Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Možnosti využití Raspberry Pi pro autonomní řízení modelu vozidla

Diplomová práce

Autor: Bc. Martin Hromádko
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.

Hradec Králové Říjen 2017

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

vlastnoruční podpis

V Hradci Králové dne 7.11.2017 Martin Hromádko

Poděkování:

Děkuji vedoucímu diplomové práce doc. Ing. Filipu Malému, Ph.D. za metodické vedení práce.

Anotace

Práce se zabývá možnostmi využití minipočítače Raspberry Pi pro autonomní řízení modelu vozidla. V teoretické části práce je věnována pozornost technologiím používaným při výzkumu autonomního řízení vozidel. Práce zmiňuje architekturu návrhu systému autonomního řízení, popisuje používané hardwarové prvky a zmiňuje i používaný software. Nedílnou součástí práce je i popis minipočítače Raspberry Pi z hlediska jeho koncepce a použití. Práce definuje kompletní návrh řešení autonomního řízení modelu vozidla s použitím minipočítače Raspberry Pi. Návrh se zabývá nejen architekturou celého systému, ale i implementačními detaily hardwarových a softwarových částí systému. Výsledkem práce je nejen samotný návrh, ale i platforma a několik nástrojů pro další rozvoj a výzkum autonomního řízení.

Annotation

Title: Usage possibilities of Raspberry Pi for autonomous control of vehicle model

The thesis deals with the possibilities of using Raspberry Pi for autonomous control of the vehicle model. In the theoretical part, the attention is paid to the technologies used in research of autonomous vehicle control. The thesis mentions the design architecture of the autonomous control system, describes used hardware elements and mentions used software. Description of Raspberry PI minicomputer for its design and use is an integral part of thesis. Thesis also defines a complete design solution for autonomous vehicle model control using the Raspberry Pi minicomputer. The design deals not only with the architecture of the whole system, but also with the implementation details of the hardware and software parts of the system. The result of the thesis is not only the proposal itself but also a platform and several tools for further development and research of autonomous vehicle control.

Obsah

Seznam obrázků

Seznam tabulek

Zdrojové kódy

1 Úvod

Umělá inteligence zažívá v poslední době renesanci a díky výkonnějšímu hardwaru se ji daří implementovat do různých odvětví běžného života. Jedním z těchto odvětví je automobilový průmysl. V posledních modelech vozidel se objevují jízdní asistenty, které využívají počítačového vidění založené na neuronových sítích. Tyto asistenty například sledují dopravní značky a řidiče upozorní na překročení aktuální maximální povolené rychlosti nebo značky oznamující nějaké nebezpečí. Úplně samostatnou kapitolou jsou vozidla, která jsou schopna se za určitých podmínek řídit sama. Tento typ vozidel může být považován za určitou předzvěst dalšího vývoje automobilů směrem k plné automatizaci.

Jeden významný hráč na poli autonomních automobilů dokázal, že pro základní míru automatizace stačí automobil vybavit kamerami a patřičným software, který umí zpracovávat obraz z těchto kamer. I v této práci se bude vycházet z jednoduchých součástek, pomocí kterých bude dosaženo autonomního chování pro malý model vozidla.

V první kapitole se práce zabývá technologiemi, které jsou používány při vývoji autonomních vozidel. Je zde zmíněno třídění typů autonomního řízení automobilů, struktura řídícího systému autonomního řízení vozidla, hardware používaný v různých prototypech a produkčních vozidlech a také základní software, který se používá v systémech autonomního řízení vozidla.

Další část práce je věnována Raspberry Pi. Tento mini-počítač se díky svým vlastnostem stal velmi oblíbeným základem pro různé hardwarové projekty. Ne jinak tomu je i v případě této práce. V kapitole je věnována pozornost samotnému Raspberry Pi, jeho historii, hardwarové výbavě, softwarovým možnostem a případy použití tohoto minipočítače.

Třetí část se zabývá samotným návrhem řešení pro autonomní řízení modelu vozidla, a to od obecné struktury až po konkrétní návrh, jehož cílem je přinést i znovupoužitelnou platformu pro testování různých přístupů k autonomnímu řízení modelu vozidla.

2 Technologie používané pro autonomní řízení vozidel

V první části této kapitoly budou popsány jednotlivé části a jejich význam v obecném systému pro autonomní řízení vozidla. Následně budou zmíněny hardwarové i softwarové prvky, které v autonomních vozidlech používají.

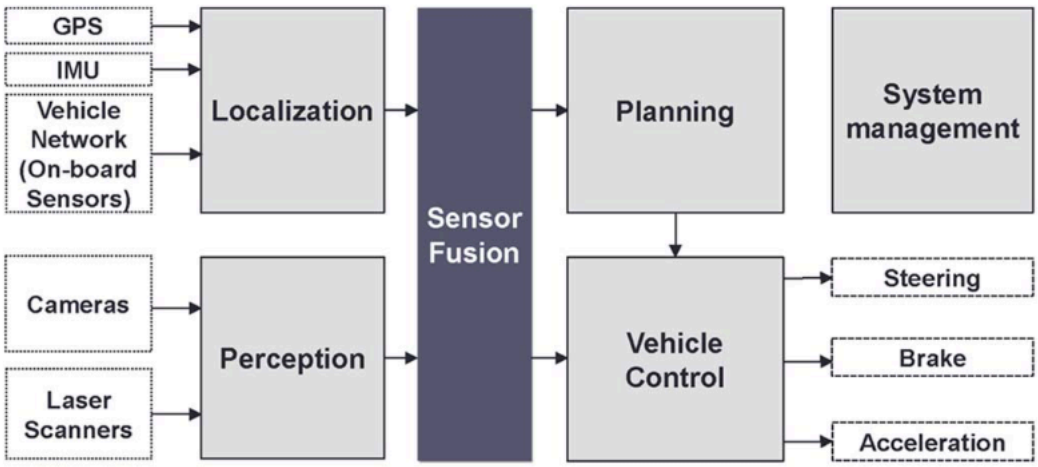
2.1 Třídy autonomního řízení vozidel

V roce 2014 vydala sdružení SAE International první ucelenou informační zprávu, která definuje klíčové koncepty spojené s autonomním řízením vozidel. Poslední aktualizace této zprávy je z roku 2016 a definuje 6 úrovní automatizace řízení:

- **Úroveň 0** – *Žádná Automatizace* – řidič se plně věnuje řízení vozidla i přes vylepšení vozidla o varovné a intervenční systémy;
- **Úroveň 1** – *Asistence řidiči* – speciální mód řízení vozidla, kdy systém řídí zrychlování, zpomalování vozidla nebo zatáčení za použití informací o prostředí a za předpokladu, že řidič bude vykonávat všechny ostatní aspekty řízení;
- **Úroveň 2** – *Částečná automatizace* – speciální mód řízení vozidla, kdy systém řídí zrychlování, zpomalování a zatáčení, za použití informací o prostředí a za předpokladu, že řidič bude vykonávat všechny ostatní aspekty řízení;
- **Úroveň 3** – *Podmíněná automatizace* – speciální mód řízení vozidla, kdy systém vykonává všechny aspekty dynamického řízení vozidla, ale předpokládá, že řidič vhodně zareaguje na žádost k zásahu;
- **Úroveň 4** – *Vysoká automatizace* – speciální mód řízení vozidla, kdy systém vykonává všechny aspekty dynamického řízení vozidla a nepředpokládá, že řidič vhodně zareaguje na žádost k zásahu;
- **Úroveň 5** – *Plná automatizace* – plně autonomní řízení, kdy systém obsluhuje všechny aspekty dynamického řízení vozidla na všech silnicích a za jakýchkoliv povětrnostních podmínek. [***1***]

2.2 Struktura systému autonomního řízení vozidla

Systém autonomního řízení je složen z pěti hlavních částí – z modulu lokalizace, senzorů okolí, modulu plánování, ovládání vozidla a správy systému.



Obrázek 1 - Schéma systému pro autonomní řízení vozidla

*zdroj: [**2**]*

2.2.1 Lokalizace (Localization)

Lokalizační systém je klíčovou částí autonomního auta. Autonomní automobily využívají tyto systémy k určování aktuální pozice vozidla a následnému výpočtu ideální trasy. Nejčastějším systémem pro určování pozice je systém GPS (Global position systém), který dokáže poskytovat nejen informace o poloze, ale i o přibližné rychlosti. Nicméně holá data z GPS systému nemohou být použita pro systém autonomního řízení, protože kvalita těchto dat se částečně odvíjí od kvality signálu z družic na oběžné dráze Země. Přesnost, kvalita a kontinuita dat se zásadně zhoršuje, když podmínky přenosu jsou nestabilní z důvodu různých stínění a odrazů signálu ze satelitů.

Pro korekce výpočtu polohy se používají další senzory, které poskytují informace o aktuálním stavu respektive pohybu vozidla. Dodatečné senzory pohybu mohou být senzory rychlosti kol, gyroskopy, akcelerometry nebo senzory magnetického pole. Pro korekce pozice jsou využívány rovněž digitální mapy nebo data ze senzorů prostředí, které poskytují komplexnější informace o pohybu a okolí vozidla. [***2***]

2.2.2 Senzory okolí (Perception)

Senzory okolí poskytují informace o okolním prostředí za pomoci několika typů senzorů jako jsou kamery, radary nebo laserové skenery.

Senzory vzdálenosti (např. radar a laserový scanner - lidar) detekují a sledují vzdálenost od pohyblivých a nepohyblivých překážek. [***2***] Senzory vzdálenosti můžeme v principu považovat za přesné, protože způsob, jakým je prováděno měření, vychází ze známých fyzikálních jevů. V případě radaru jde o odraz zvukových vln a v případě laserového scanneru o odraz světelného záření.

Senzory vidění, jako jsou například kamery, slouží k rozpoznávání objektů v zorném poli vozidla. Slouží k rozpoznávání dopravního značení i vzdálených objektů, které mohou přímo nebo nepřímo ovlivňovat dopravní situaci v okolí vozidla. Senzory vidění, které jsou implementovány kamerami, ale nemusí fungovat úplně spolehlivě. Nejde však o samotnou implementaci softwarového zpracování výstupu z kamery, ale hlavně o světelné podmínky, které mohou ovlivnit samotný senzor kamery. Mohou nastat situace, kdy slunce může oslnit kameru (stejně jako člověka) natolik, že výstup z kamery není dostatečně kvalitní a vozidlo tímto způsobem přijde o důležitý zdroj informací o prostředí, ve kterém se pohybuje.

V nedávné minulosti byl zaznamenán případ, kdy autonomní vozidlo, které používalo jako hlavní (a v dané verzi i jediný) senzor – kameru, nedokázalo zabránit dopravní nehodě, protože nízké ranní slunce oslnilo kameru.

2.2.3 Modul plánování

Modul plánování má na starost manévrování autonomního vozidla. Plánovací algoritmy pro autonomní vozidla rozdělujeme do třech fází, aby bylo dosaženo bezpečného a spolehlivého manévru vozidla v rozličných situacích provozu. Jsou to globální směrování, uvažování o chování a lokání plánovaní pohybu. [***2***]

Globální směrování

Globální směrování hledá nejrychlejší a nejbezpečnější cestu, jak se dostat z výchozí do cílové pozice. V této fázi je klíčový systém pro správu digitálních map a vyhledávací algoritmus. [***2***]

Lokální plánování pohybu

Do lokálního plánování pohybu spadají úkony „na krátkou vzdálenost“. Takový úkon můžeme považovat třeba objíždění překážky, odbočování nebo předjíždění. Jsou to situace, kdy se vozidlo rozhoduje převážně na základě dat z vlastních senzorů, a které se mohou nepravidelně měnit.

2.2.4 Ovládání vozidla

Modul pro ovládání vozidla poskytuje klíčovou funkcionalitu pro řízení autonomního vozidla po plánované trase. Modul ovládání vozidla by měl být přesný tak, aby mohl bezpečně řídit vozidlo a robustní natolik, aby se dokázal přizpůsobit různým podmínkám provozu. Aby bylo možné těmto požadavkům vyhovět, je potřeba, aby se systém uměl vypořádat s různými charakteristikami. Jsou to například holonomická omezení, dynamika vozidla nebo fyzikální omezení (omezené možnosti zatáčení nebo maximální přilnavost pneumatik). Navíc systém pro ovládání vozila musí řešit kompromis mezi výkonem a jízdním komfortem.

V praxi se k vypořádání s charakteristikami vozidla využívá rozdělení řídícího systému na boční a podélný. Boční řídící algoritmus předpokládá, že se vozidlo pohybuje podél Ackermannovy řídící geometrie Díky tomuto předpokladu je cílový úhel vypočítán z chyby předchozích bodů v generované cestě. Body generované cesty a zpětná vazby jsou plánovány s ohledem na rychlost vozidla a zakřivení cesty tak, aby bylo dosaženo přesného následování cesty.

Podélný řídící algoritmus odvozuje cílovou pozici akceleračního a brzdového pedálu z referenčních vstupů. Podélný řídící systém se skládá ze tří režimů, jsou to: řízení rychlosti, řízení vzdálenosti a nouzové zastavení. Řízení rychlosti odvozuje zrychlení z cíle a aktuální rychlosti vozidla. V režimu řízení vzdálenosti je pro řízení rychlosti přidána další výpočetní funkce. Funkce generuje požadovanou rychlost pomocí odchylky polohy mezi aktuální a cílovou polohou. Pokud se vozidlo dostane do nepředpokládané situace, jako je třeba výpadek nějakého systému nebo náhlá překážka, tak je aktivováno nouzové zastavení, které neodkladně zastaví vozidlo. [*2*]

2.2.5 System management

Aby bylo možné provádět kontrolu všech modulů, je potřeba, aby takto složitý systém měl modul správy celého systému. Tento modul má na starost hlavně informování systému o stavu modulů a případné odpojování a připojování redundantních modulů, kdyby došlo k poruše atd.

2.3 Hardware

Tato kapitola se zabývá hardwarovými technologiemi, které jsou používané při vývoji autonomních vozidel.

2.3.1 LiDAR

LiDAR (light detection and ranging) je zařízení pro 3D mapování prostoru. Jak již název napovídá, lidar používá k měření vzdálenosti laserové světlo. Aktuálně používaná technologie může skenovat vzdálenost více jak 100 metrů ve všech směrech. Generuje při tom přesnou 3D mapu prostoru kolem vozidla. V současné době je největším problémem masivního rozšíření používání lidaru cena a velké množství generovaných dat, která musejí být zpracována. [***3***]

Princip fungování LiDARu využívá fyzikálních vlastností světla. Přesněji to, že pokud vyšleme paprsek světla na nějaký předmět, tak se světlo odrazí od tohoto předmětu a my ho můžeme vidět. Díky znalosti rychlosti světla můžeme vypočítat vzdálenost předmětu na základě zpoždění světla, které se od předmětu odrazilo.

Výsledná vzdálenost se vypočítá podle vzorce (Vzorec 1).

$$s = \frac{l \cdot t}{2}$$

Vzorec 1

kde l je rychlost světla a t čas, za který světlo urazilo cestu tam a zpět.

LiDARové zařízení vysílá rychlé pulzy laserového světla, v některých případech až 150000 pulzů za sekundu. Senzor na zařízení měří množství času, za který se pulz odrazí zpět. Světlo se pohybuje se známou konstantní rychlostí, takže LiDARové zařízení může vypočítat vzdálenost mezi ním a cílem s vysokou přesností. Opakováním tohoto postupu v rychlém sledu může zařízení vytvořit komplexní mapu povrchu, který měří. U leteckých LiDARů je nutné sbírat i další data, která

zajistí dostatečnou přesnost. Při pohybu senzoru je potřeba, aby pozice a orientace zařízení byly také zaznamenány. To je potřeba k určení pozice laserového pulzu ve chvíli vyslání a ve chvíli, kdy dorazí zpět. Tyto informace jsou rozhodující pro integritu dat. Pokud používáme LiDAR na zemi, je možné ukládat GPS pozici místa, kde je umístěno zařízení.

Celkem existují dva typy detekčních metod, které LiDARy využívají. Přímá detekce energie, která je známa jako inkoherentní detekce, a koherentní detekce. Koherentní systémy jsou nejlepší pro Dopplerovské nebo fázově citlivé měření a obvykle používají optickou heterodynní detekci. Toto jim dovoluje pracovat s mnohem menší energií, ale při vyšších nárocích na vysílač, který musí být složitější. Pro obě detekční metody mohou být použity dva pulzní modely: mikropulzní a vysokoenergetický systém. Mikropulzní systémy se vyvinuly jako výsledek výkonnějších počítačů s vyšší výpočetní schopností. Tyto lasery mají nižší příkon a jsou klasifikovány jako „bezpečné pro oko“, což jim dovoluje být používány jen s malými bezpečnostními opatřeními. Vysokoenergetické systémy jsou nejčastěji využívány pro atmosférický výzkum. [***4***]

Obrázek 2 - LiDAR umístěný na střeše osobního vozu

zdroj: https://www.linkedin.com/pulse/lidar-heart-self-driving-cars-geo-plus/

Většina LiDARů má tyto části:

Lasery

Lasery jsou kategorizovány podle vlnové délky světla, které vysílají. Lasery s vlnovou délkou 600-1000nm jsou častěji využívány pro nevědecké účely, ale protože jejich vlnová délka je viditelná pro lidské oko, mají omezený maximální výkon. Lasery s vlnovou délkou 1550nm jsou hlavní alternativou, protože tato vlnová délka už není lidským okem viditelná, a jsou „bezpečné pro oko“ i při vyšších výkonech. [***4***]

Skenery a optika

Rychlost pořizování obrázků je ovlivňována rychlostí, s jakou mohou být naskenovány do systému. Existují různé metody skenování pro různé účely měření. Typ optiky určuje rozlišení a dosah, který může být detekován systémem. [***4***]

Fotodetektor a elektronika přijímače

Fotodetektor je zařízení, které čte a zaznamenává signál vracející se do systému. Jsou dva typy technologií fotodetektorů. Polovodičové detektory, jako lavinová fotodioda, a fotonásobiče. [***4***]

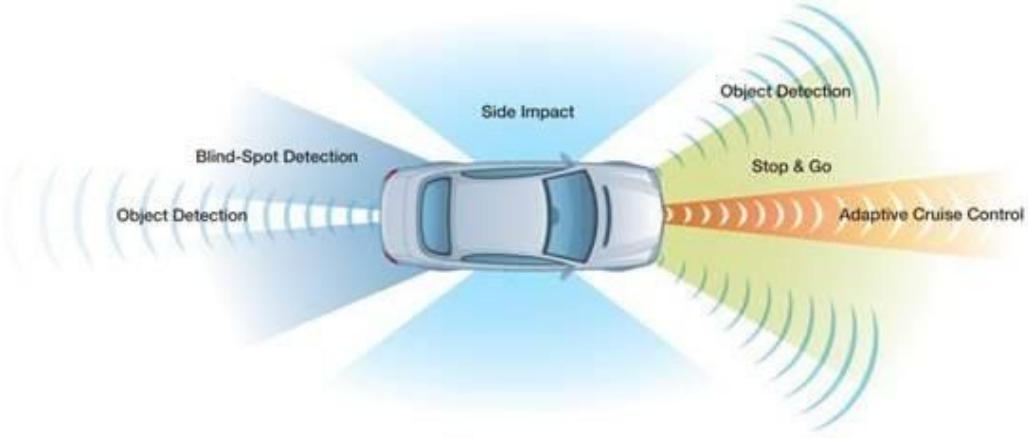
Navigační a poziční systém

Když je LiDARový senzor připevněn na mobilní platformu jako jsou satelity, letadla nebo automobily, je nutné určit absolutní pozici a orientaci senzoru k zachování použitelných dat. Systém GPS poskytuje přesné geografické informace týkající se pozice senzoru. IMU (Inertia Measurement Unit) jednotka zaznamenává přesnou orientaci senzoru na daném místě. Obě dvě zařízení poskytují metodu pro překlad senzorových dat do pevných bodů pro použití v různých systémech. [***4***]

2.3.2 Radar

Radar je senzorový systém, který používá rádiové vlny k určení rychlosti, rozsáhlosti a úhlu objektů. Radar je výpočetně méně náročný než kamera a vytváří méně dat než lidar. Je sice úhlově méně přesný než lidar, ale zato funguje za všech podmínek. Při správném využití odražených vln je možné „vidět“ za překážky. Moderní prototypy autonomních vozidel spoléhají na kombinaci radaru a lidaru pro vzájemné ověřování toho, co „vidí“. [***3***]

Obrázek 3 zobrazuje místa na moderním vozidle, kde jsou používány radarové systémy. Slouží nejen k detekci překážek před vozidlem, ale dokáží kontrolovat tzv. „mrtvý“ úhel – místo, kam řidič pomocí zrcátek nevidí. Rovněž slouží k detekci bočního nárazu, kdy pomáhají správném načasování exploze aibagu.



Obrázek 3 - Místa použití radarů na moderním voze

zdroj: https://cz.pinterest.com/pin/327848047850806966/

2.3.3 **Kamera**

Hlavním úkolem kamer při autonomním řízení vozidel je záznam obrazu pro klasifikaci a interpretaci textur. V současné době se jedná o nejlevnější a nejdostupnější senzor. Kamera produkuje velké množství dat (fullHD rozlišení = miliony pixelů nebo MB pro každý snímek). Proto je zpracování obrazu z tohoto typu senzoru výpočetně a algoritmicky náročné. Na rozdíl od lidaru a radaru, kamery mohou rozlišovat barvy, a proto se nejvíce hodí pro interpretaci scény okolí vozidla. [***3***]

2.4 Software

V této části bude věnována pozornost softwarovému vybavení systému řízení autonomního vozidla.

Dnešní vozidla mají kromě řídící jednotky motoru, která primárně zajišťuje jeho správný chod, další podpůrné počítačové systémy. Typickým příkladem takového systému je palubní počítač, který řidiči vozidla zobrazuje informace o ujetých kilometrech, průměrné spotřebě paliva nebo informuje řidiče o nutnosti navštívit

servis na výměnu oleje. V mnoha novějších vozech může řídící jednotka informovat o chybových stavech řídící jednotky pomocí displeje na palubní desce vozidla. Podobným systémem je integrovaná GPS navigace, která mimo navigaci řidiče k určenému cíli může informovat řídící jednotku automatické převodovky o profilu trasy z mapových podkladů. Automatická převodovka na základě těchto informací přizpůsobí volbu rychlostního stupně, to má za následek pružnější reakce vozidla a úspornější provoz.

Systémy autonomního řízení vozidel se staví o úroveň výše, ve snaze nahradit řidiče. To má za následek, že všechny stávající systémy, které dnes ve vozech máme, nebudou odstraněny, jen jejich rozhraní nebude o stavu vozidla informovat (jenom) řidiče, ale právě systém autonomního řízení.

2.4.1 Systémy reálného času

Hlavní softwarovou částí každého počítače je kromě zavaděče operační systém. V systémech autonomního řízení vozidel by mělo být naprostou samozřejmostí použití systému reálného času.

Obecně je operační systém odpovědný za správu hardwarových prostředků počítače a aplikací, které na tom počítači běží. Systém reálného času provádí tyto úkoly, ale zároveň je navržen tak, aby mohl provozovat aplikaci s přesným plánováním a vysokým stupněm spolehlivosti. [***5***] To je důležité hlavně v situacích, kdy prodleva programu může způsobit nějaké bezpečnostní riziko. Třeba program, který zpracovává signály ze senzorů, musí být prováděn v pravidelných intervalech, aby vozidlo nenarazilo.

Aby bylo možné operační systém považovat za „systém reálného času“ musí být znám buď maximální čas pro vykonání každé kritické operace, kterou provádí, anebo musí alespoň garantovat maximum po většinu času. Operační systémy, které absolutně garantují tento maximální čas pro tyto operace jsou běžně označovány jako „tvrdé systémy reálného času“. Systémy, které mohou garantovat maximální čas pro kritické operace pouze po většinu běhového času, jsou označovány jako „měkké systémy reálného času“. V praxi mají tato označení pouze omezenou použitelnost, protože každý řešení systému reálného času demonstruje unikátní výkonové charakteristiky a uživatel by k nim měl přistupovat opatrně. [***5***]

Aby bylo možné přenášet aplikace z jedné platformy na druhou, musí existovat standardy, kterými se operační systémy řídí. Ne jinak tomu je i u systémů reálného času. V současnosti existují čtyři hlavní standardy systémů reálného času:

- POSIX, hlavní standard operačních systému pro všeobecné použití s real-time rozšířením (RT-POSIX),
- OSEK, pro automobilový průmysl,
- APEX, pro letecké systémy,
- μITRON, pro malé embedded systémy. [***6***]

Dále bude věnována pozornost pouze standardu OSEK, který je vyvíjen pro automobilový průmysl.

OSEK/VDX je společný projekt automobilového průmyslu, který se snaží definovat standard s otevřenou architekturou pro distribuované řídící jednotky ve vozidlech. Zkratka OSEK znamená „Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug“ (Otevřené systémy a odpovídající rozhraní pro automobilovou elektroniku). Druhá používaná zkratka VDX znamená „Vehicle Distributed eXecutive“(Distribuované řízení vozidla). Cílem standardu je popsat prostředí, které podporuje efektivní využití prostředků pro automobilový aplikační software. Typické aplikace uvažované standardem jsou velké, vysoce kritické řídící aplikace s přísnými real-time omezeními. Aby se ušetřily produkční prostředky, je zde tlak na optimalizaci kódu směrem ke zmenšení paměťové stopy a zlepšení výkonu systému, jak jen to bude možné. Typický OSEK systém má následující vlastnosti:

- **Škálovatelnost** – Zamýšlené použití systému je na široké škále hardwarových platform – od 8-bitových mikrokontrolerů po výkonné procesory. Aby bylo možné podporovat takový rozsah platform, tak standard definuje 4 třídy shody s rostoucí komplexností. Ochrana paměti není vůbec podporována;
- **Přenositelnost softwaru** – Pro zjednodušení přenositelnosti softwaru je v operačním systému implementováno rozhraní. ISO/ANSI-C. Nicméně z důvodu velké variability hardwarových platforem, standard nespecifikuje

žádné rozhraní pro I/O subsystém. To snižuje přenositelnost zdrojového kódu aplikace;

- **Konfigurovatelnost** – Standard navrhuje odpovídající konfigurační nástroje, které pomohou návrháři při ladění systémových služeb a paměťových požadavků systému. Objekty, jejich instance má být v aplikaci vytvořena, mohou být specifikovány skrze jazyk nazvaný OIL (OSEK Implementation Language);
- **Statická alokace softwarových komponent** – Všechny objekty kernelu a aplikačních komponent jsou alokovány staticky. Počet úloh, jejich kód a požadované zdroje a služby jsou alokovány až při kompilaci. Tento přístup zjednodušuje vnitřní strukturu jádra a zjednodušuje možnost nasazení jádra a aplikačního kódu do ROM (Read-Only Memory);
- **Znalost jádra** – OSEK standard podporuje standardní ORTI (OSEK Run Time Interface), používané k informování debuggeru o významu jednotlivých datových struktur, takže specifická informace může být vizualizována na obrazovce správným způsobem;
- **Podpora časově spínaných architektur** – Standard OSEK poskytuje specifikaci pro OSEKTime OS - časově spínaný operační systém, který může být plně integrován do OSEK/VDX frameworku. [***6***]

3 Možnosti využití Raspberry Pi

Raspberry Pi je počítač o velikosti kreditní karty, ke kterému stačí připojit displej a klávesnici. Je to schopný malý počítač, který může být použit v elektronických projektech, ale i pro další věci, které zvládá klasický stolní počítač. Myšleny jsou tabulkový a textový procesor, prohlížeč internetu a hraní her. [*7*] Nutno ovšem podotknout, že hraní her je, vzhledem k výkonu Raspberry Pi, omezené.

Raspberry Pi bylo původně navrženo jako vzdělávací zařízení. Bylo inspirováno úspěchem počítače BBC Micro, na kterém se naučila programovat celá jedna generace. Nadace Raspberry Pi si stanovila, že chce docílit stejného úspěchu v dnešním světě. Ve světě, kde k používání počítače není potřeba, aby lidé uměli psát počítačové programy. [*8*]

Obrázek 4 - Raspberry Pi 3 model B

zdroj: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

3.1 Historie

Za podobnými projekty, jako je Raspberry Pi, stojí tým lidí, v tomto případě se tento tým rozhodl ustanovit i právní podstatu a vytvořili Nadaci Raspberry Pi. Nadace Raspberry Pi je nezisková organizace, která byla založena v roce 2006. U jejího zrodu stáli Eben Upton, Rom Mullins, Jack Lang a Alan Mycroft. Zakladatelé této nadace cítili znepokojení nad nedostatkem zařízení, která by mohli zájemci používat k učení a experimentování. Cílem této nadace je podporovat studium počítačových věd pro generaci, která nevyrostla na BBC Micro nebo na Commodore 64.

Vývoj každého produktu prochází množstvím iterací před tím, než je produkován ve velkém. Ne jinak tomu bylo i u Raspberry Pi. Vývoj začal v roce 2006, kdy bylo vytvořeno několik konceptuálních verzí Raspberry Pi založených na 8-bitovém mikrokontroléru Atmel ATMega644. Další koncept byl založen na USB paměťovém disku s procesorem ARM. Celkový vývoj k první oficiální verzi trval 6 let.

Prvních 50 zařízení s Alfa verzí Raspberry Pi bylo vytvořeno do srpna roku 2011. Tato zařízení byla o něco větší než ty dnešní a sloužila k tomu, aby je Nadace Raspberry Pi mohla otestovat a aby si potvrdili, že daný návrh bude fungovat tak, jak se předpokládalo. V prosinci roku 2011 bylo sestaveno 25 zařízení s beta verzí Raspberry Pi, které byly vydraženy, aby Nadace Raspberry Pi získala potřebné peníze. V beta verzi byla nalezena pouze jedna malá chyba, která byla opravena ještě před samotnou produkcí.

První série počítačů Raspberry Pi byla vyrobena v Číně a na Taiwanu a čítala 10000 zařízení. Naneštěstí se objevil problém s konektorem pro Ethernet, protože byl nahrazen nekompatibilní součástkou. Tento problém způsobil menší zpoždění distribuce o pár dní. Všechny objednávky však byly doručeny do týdne od slibovaného data doručení. Jako bonus se nadaci povedlo vylepšit Raspberry Pi Model A 256 MB RAM místo plánovaných 128 MB. V současné době je Raspberry Pi vyráběn ve Spojeném království. [***8***]

V únoru 2014 přichází model „A“ (původní Rasbperry označováno jako model „B“), který je odlehčenější oproti původní verzi a má nižší spotřebu energie. Deska disponovala 256 MB RAM, jedním USB portem a přišla o Ethernet konektor. Nižší

spotřeba desky pomohla k tvorbě projektů, které jsou napájeny z baterií nebo solárních panelů. [*7*]

V polovině roku 2014 je vydáno vylepšení „B+“, které přidává oproti původní verzi 2 další USB konektory (celkem jsou 4) a rozšiřuje počet GPIO pinů z původních 24 na 40. Mimo to byl původní slot na SD kartu nahrazen microSD slotem a snížena spotřeba o cca 0,5 – 1 W. [*9*]

Model „A+“ se dostal na světlo světa v listopadu 2014. Hlavní předností tohoto modelu je velikost a to cca 56×65 mm. [*10*] Ve výbavě došlo pouze k jediné úpravě - slot na SD kartu nahradil microSD slot. [*9*]

V roce 2015 přichází na trh nová verze Raspberry Pi 2, která získává výkonnější procesor BMC 2836 s taktem 900 MHz a 1 GB paměti.

Poslední vydanou verzí je Rasbperry Pi 3, které dostalo čtyřjádrový procesor s taktem 1,2 GHz a navíc Wifi a BLE čip.

3.2 Hardware

Raspberry Pi obsahuje vše potřebné k tomu, aby mohl být považován za plnohodnotný počítač. Jako konkrétní příklad je možné zmínit Raspberry Pi 3 Model B, který má následující výbavu:

- Čtyřjádrový processor Broadcom BCM2837 s taktem 1.2 GHz a 64-bitovou architekturou,
- 1 GB operační paměti,
- Wi-fi a BLE (Bluetooth Low Energy) čip BCM43438,
- porty USB 2,
- pólový stereo výstup a současně výstup kompozitního videa,
- standardní HDMI port. [*7*]

Mimo tyto běžné součásti počítače disponuje Raspberry Pi i speciálními vstupně-výstupními konektory:

- CSI (Camera Serial Interface) pro připojení speciální Raspberry Pi kamery,
- DSI (Display Serial Interface) pro připojení speciálního displeje pro Raspberry Pi,

- 40-pinový GPIO (General-purpose input/output) port. [*7*]

Kvůli svým rozměrům nepoužívá Rasperry Pi klasické diskové nebo SSD úložistě, ale veškerá data jsou uložena na výměnné MicroSD kartě.

3.3 Software

Jelikož je Raspberry Pi plnohodnotný počítač, tak je v principu možné nainstalovat jakýkoliv operační systém, který podporuje architekturu procesoru dané verze Raspberry Pi. Oficiálně podporovaným systémem je speciální distribuce Linuxu, která se jmenuje Raspbian, a je založená na linuxové distribuci Debian, ale existují i jiné systémy, které Raspberry Pi podporují.

NOOBS

NOOBS není plnohodnotný operační systém. Je to instalátor, skrze který je možné nainstalovat nejen Raspbian, ale i jiné operační systémy. Na oficiálních stránkách Raspberry Foundation je možné stáhnout dvě verze. Verzi *NOOBS*, která obsahuje i soubory pro offline instalaci systému Raspbian, a verzi *NOOBS Lite*, která slouží pouze k online instalaci systémů. [*7*]

Raspbian

Jak již bylo zmíněno, systém Raspbian je oficiálně podporovaným operačním systémem pro Raspberry Pi. Obsahuje spoustu předinstalovaného softwaru pro vzdělávání, programování a obecné používání, jako Python, Scratch, Sonic Pi, Java, Mathematice a další. [*7*]

Raspbian vznikl jako neoficiální port linuxové distribuce *Debian Wheezy armhf* s upravenými parametry kompilace tak, aby operace s plovoucí řádovou čárkou mohly fungovat na procesoru Raspberry Pi. Toto poskytlo výrazně vyšší výkon pro aplikace, které provádí velké množství operací s plovoucí řádovou čárkou. Ostatní aplikace také získají vyšší výkon díky použití pokročilých instrukcí procesoru architektury ARMv6 v Raspberry Pi. [*11*]

Windows 10 IoT Core

Na počítači Raspberry Pi je možné provozovat operační systém Windows 10. Samozřejmě není možné provozovat Windows taková, jaká můžeme znát z běžných osobních počítačů.

Windows 10 IoT Core je verze operačního systému Windows 10 optimalizovaná pro menší zařízení s displejem i bez něj, která pohání procesor klasické (x86/x64) nebo ARM architektury. [***12***]

Ubuntu

Pro Raspberry Pi jsou na oficiálních stránkách [***7***] zmíněny dvě odvozené verze linuxové distribuce Ubuntu.

První z nich je *Ubuntu MATE*, která na rozdíl od standardní distribuce využívá grafické prostředí MATE s ve výchozím stavu se načítá i méně podpůrných nástrojů a díky tomu nepotřebuje k svému běhu tolik systémových prostředků.

Další zmiňovanou verzí je *Ubuntu Core*. Stejně jako u Windows 10 IoT Core se jedná o odlehčenou verzi původního systému určenou pro zařízení ve světě *Internet of Things* (IoT).

OSMC

OSMC (Open Source Media Center) je open-source mediální přehrávač k volnému užití založený na Linuxu. Projekt OSMC byl založen v roce 2014 a umožňuje přehrávání mediálních souborů z lokální sítě, připojeného uložiště nebo Internetu. Na vývoji tohoto přehrávače se kromě týmu OSMC podílejí i lidé z projektu Debian nebo Kodi. [***13***]

PiNET

PiNET je systém, který byl vydán Adrewem Mulhollandem, studentem počítačových věd na Queens Univerzity v Belfastu. Systém byl navržen pro škola a organizace k nastavení a správě sítě s počítači Raspberry Pi. Systém má reflektovat podobné systémy pro systém Windows. Veškerý software a dokumentace jsou kompletně zdarma s open-source licencí a byly vytvořeny za dohledu školitelů z celého světa. [***14***]

RISC OS Pi

Na Raspberry Pi je možné provozovat i jiné systémy, než jsou systémy založené na Linuxu, případně Windows. Jedním ze zástupců takových systémů je RISC OS.

RISC OS je Britský operační systém, který byl navržen speciálně pro ARM procesory stejným týmem, který vytvořil originální ARM. Není to verze Linuxu ani není žádným způsobem spřízněný s Windows. Má množství unikátních vlastností a aspektů jeho návrhu. [***15***]

3.4 Použití Raspberry Pi

Díky svým vlastnostem (především rozměrům) je Raspberry Pi zajímavý nástroj pro projekty z různých oborů. V následující části budou zmíněny nejpoužívanější oblasti, kde se Raspberry Pi používá.

3.4.1 **Výuka**

Při vývoji Raspberry Pi chtěli autoři navázat na úspěchy BBC Micro a Commodore 64 a vytvořit nástroj pro výuku počítačových věd resp. přiblížit svět počítačů širší veřejnosti. Dnes je ve světě Raspberry Pi používán na množství základních škol jako počítač pro výuku programování. [***7***]

3.4.2 **Automatizace**

Prostor, kde se dá využít potenciálu malého počítače, je rozhodně automatizace. Na Internetu je možné najít spoustu návodů jak na Raspberry PI zprovoznit například senzor vlhkosti nebo teploty, případně ovládat relé nebo rozsvěcet diody.
Firma Faster CZ přišla s rozšiřujícím obvodem pro Rasbperry Pi, který obsahuje relé s digitální vstupy a výstupy vhodné pro menší projekty. Mimo to vytvořili i řadu PLC jednotek, které mohou být používány při implementaci „chytrých“ domů, tak i v komerčních aplikacích a systémech. Umožňují lokální vzdálenou kontrolu připojených systémů i jednotlivých zařízení přes vstupy a výstupy. [***16***]

Obrázek 5 - UniPi Neuron S10x

*zdroj [**16**]*

3.4.3 **Multimediální zařízení**

Dalším oblíbeným použitím počítače Rasbperry Pi je stavba malého multimediálního zařízení. Raspberry Pi již od prvních verzí disponuje konektorem HDMI pro připojení displeje, který lze využít k připojení dnes běžných televizí. Grafická karta v Raspberry Pi disponuje dostatečným výkonem pro přehrávání videa v rozlišení 1920x1080 px. Jako operační systém je možné použít OSMC, který je k tomu přímo vytvořen.

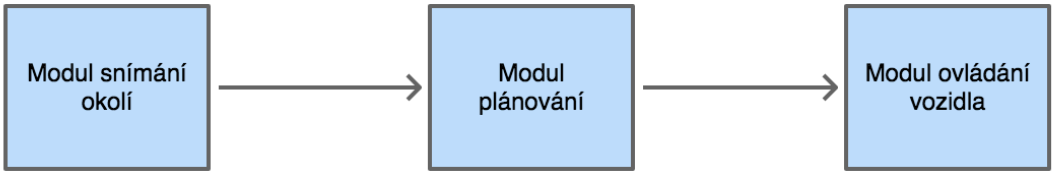
Další možnou alternativou je systém Musicbox, který se zaměřuje na streamování hudby skrze Raspberry Pi. Podporuje streamování hudby přímo z hudebních služeb a jako jeden z mála umožňuje přenos hudby přes Apple AirPlay. [***17***]

4 Návrh řešení

Návrh řešení je inspirován projektem, jehož autorem je Zheng Wang [***18***]. Oproti jeho přístupu je v tomto řešení kladen důraz na systematičnost přístupu s ohledem na další použití a rozvoj. Cílem této kapitoly je přinést řešení, které bude sloužit jako platforma pro testování různých přístupů pro autonomní řízení modelu vozidla. První část této kapitoly se bude zaměřovat na komunikaci, která bude probíhat mezi jednotlivými hardwarovými částmi systému. Od tohoto návrhu se odvíjí následná implementace softwarových modulů. Druhá část kapitoly se věnuje implementačním detailům softwarové části systému.

4.1 Struktura systému

Struktura systému vychází z obecné struktury systému autonomního řízení vozidel, který je zmiňován v druhé části práce. Oproti obecné struktuře bude zjednodušen o část lokalizace a systémového managementu, které nejsou v případě modelu v laboratorních podmínkách důležité.



Obrázek 6 - Obecné schéma navrhovaného systému

zdroj: vlastní tvorba

Vizuální podoba obecné struktury je znázorněna na obrázku (Obrázek 6) a skládá se z následujích částí:

- **Modul snímání okolí**

  V tomto modulu se budou sbírat data z přímého okolí modelu. Zde návrh počítá s použitím běžné webkamery pro snímání obrazu před modelem vozidla. Systém rovněž bude připraven pracovat se stereoskopickou kamerou, která usnadní odhady vzdálenosti objektů a další pokročilé operace.

Součástí návrhu je senzor vzdálenosti od objektů v okolí modelu. Tento senzor bude umístěn na přídi modelu a bude zaznamenávat informace o přítomnosti a vzdálenosti objektů před modelem.

Nasbíraná data bude modul poskytovat dalším modulům ve společném formátu, aby bylo možné podporovat více typů kamer a senzorů.

Jelikož se v případě tohoto systému nepočítá s lokalizací, tak tento modul bude rovněž zastávat funkci rozhraní pro senzorická data.

- **Modul plánování**

  Modul plánování bude na základě získaných dat ze senzorů rozhodovat o tom, jak má model vozidla zrychlovat nebo zpomalovat a jak moc má zatáčet. Tato část bude nejzajímavější z hlediska implementace, protože modul může být implementován pomocí různých přístupů a algoritmů. Systém bude natolik robustní, že si bude schopen poradit s modulem plánování založeném na neuronových sítích anebo na řešení, které bude využívat exaktních výpočtů pro určení směru a rychlosti pohybu.

- **Modul ovládání vozidla**

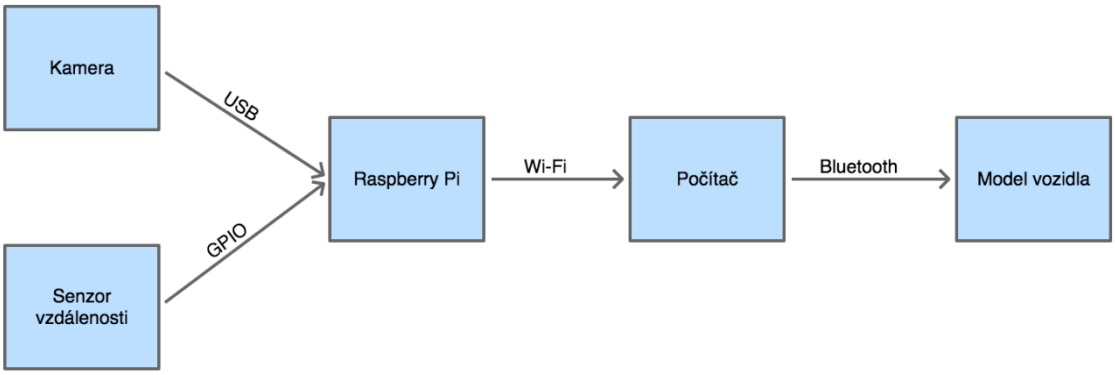
  Tento modul má na starost přenos informací o změně směru a zrychlení do modelu vozidla. Jednotlivé instrukce ze systému bude transformovat do komunikačního protokolu dané hardwarové platformy vozidla. Díky tomu bude možné podporovat široké množství hardwarových platforem nezávisle na použitém rozhodovacím algoritmu. To umožní i paralelní testování stejného algoritmu na více (různých) modelech.

4.2 Hardware a komunikace

Při návrhu tohoto systému je stejně nutné se zaměřit na návrh hardwarové části stejně jako v případě softwaru.

4.2.1 **Schéma a popis**

Hardwarové schéma navrhovaného řešení je znázorněno na následujícím obrázku.



Obrázek 7 - Schéma komunikace mezi hardwarovými moduly

zdroj: vlastní tvorba

Schéma (Obrázek 7) obsahuje 5 hardwarových modulů, mezi kterými bude probíhat komunikace pomocí různých technologií:

- **Kamera**

  Implementace počítá s použitím běžné webové kamery s rozlišením 640x480 px, která je vybavena USB rozhraním. Skrze toto rozhraní bude posílat obrazová data do Raspberry Pi.
- **Senzor vzdálenosti**

  Pro snímání překážek bude použit ultrazvukový senzor HC-SR04. Tento senzor bude připojen stejně jako kamera k Rasbperry Pi. Na rozdíl od kamery tento senzor nemá USB rozhraní a je nutné ho ovládat přímo přes GPIO sběrnici na úrovni digitálního signálu.
- **Raspberry PI**

  V tomto návrhu řešení figuruje Rasbperry Pi pouze jako zprostředkovatel komunikace. Jeho hlavním úkolem je přenos obrazových dat z kamery a zpřístupnit tato data pro počítač. Mimo to bude počítač informovat o překážce zjištěné senzorem vzdálenosti.
- **Počítač**

  Centrální částí navrhovaného systému je počítač, který bude vybaven softwarovými moduly popisovanými v předchozí podkapitole. Počítač musí být vybaven Wi-fi kartou s podporou technologie Bluetooth pro ovládání modelu vozidla.

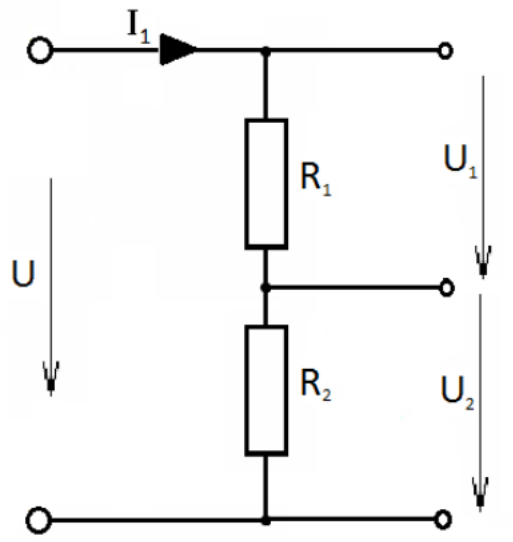
- **Model vozidla**

  Návrh počítá s použitím modelu vlastní výroby, který je postaven na základě programovatelného autíčka pro děti. V původním nefunkčním autíčku byla nahrazena veškerá elektronika vlastním řešením založeném na vývojové desce Arduino UNO. Ovládání modelu je možné realizovat skrze Bluetooth rozhraní a za použití speciálního, pro tyto účely vytvořeného textového protokolu.

Komunikace mezi jednotlivými hardwarovými moduly není homogenní. Využívá různých technologií pro přenos informací. Přenos obrazu z kamery do Raspberry Pi probíhá přes USB rozhraní za použití standardních driverů pro webové kamery. Jak již bylo zmíněno, tak součástí řešení je nutnost použití bezdrátové sítě. Tu bude distribuovat standardní bezdrátový směrovač poskytující komunikační standard IEEE 802.11n na frekvenci 2,4 GHz. Pro testovací účely poskytuje dostatečně dobré řešení, pro přenos obrazových dat do počítač a zároveň je možné implementovat i další komunikační kanály pro přenos informací z model.

4.2.2 Senzor vzdálenosti

Informace o překážce před modelem se bude získávat ze senzoru vzdálenosti. Ten bude přímo připojen k GPIO portům na Raspberry PI. Pro měření vzdálenosti je použit senzor HC-SR04. Tato součástka pracuje s 5 V logikou a Raspberry Pi sice poskytuje napájecí výstup s 5 V, ale vstupně-výstupní piny GPIO sběrnice pracují pouze s napětím 3,3 V. Provozování zařízení bez použití podpůrných obvodů a s různým napětím operační logiky je možné pouze směrem od zařízení s 3,3 V logikou směrem k zařízení s 5 V logikou. Toto je možné díky tomu, že minimální vstupní úroveň napětí pro logickou hodnotu „1“ v 5 V logice jsou 2 V. Běžné zařízení pracující s 3,3 V logikou vysílá logickou „1“ napětím o přibližné hodnotě 3 V. Ovšem komunikace od zařízení s 5 V logikou je možná pouze za použití podpůrného obvodu, a to tvz. děliče napětí. Dělič napětí je jeden ze základních elektrotechnických obvodů, jehož účelem je snižovat vstupní napětí na předem určenou úroveň. Úroveň výstupního napětí je ovlivněna hodnotami odporů v obvodu. Zapojení obvodu je ilustrováno na obrázku (Obrázek 8).



Obrázek 8 - Schéma zapojení napěťového děliče

*zdroj: [**19**]*

Dělič napětí pro tento projekt musí být konstruován tak, aby snižoval vstupní napětí z 5 V logiky na napětí pro 3,3 V logiku. Pro určení hodnot odporů v děliči napětí se používá vzorec (Vzorec 2).

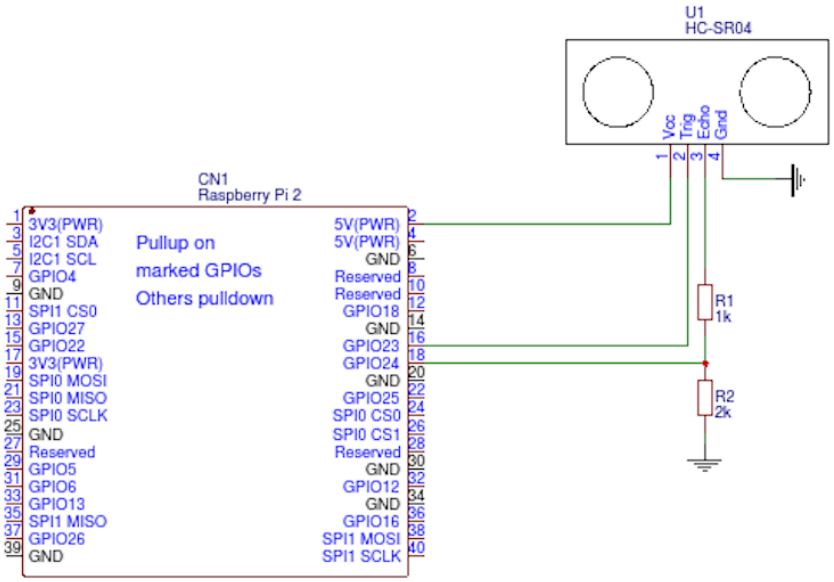
$$V_{výstupní} = V_{vstupní} \frac{R_2}{R_1 + R_2}$$

Vzorec 2

Po dosazení vstupního a výstupního napětí a následných úpravách vznikne:

$$0.66(R_1 + R_2) = R_2$$

Nyní je potřeba dosadit hodnotu za jeden z odporů. Je vhodné dosadit takovou hodnotu, která je běžně k dostání v obchodech s elektronikou. Proto byla pro R_1 zvolena hodnota 1000 Ω. Po dosazení vychází, že hodnota odporu R_2 by měla být přibližně 1941 Ω. Jelikož konstrukce odporu s přesnou hodnotou by byla technologicky náročná, je možné výslednou hodnotu zaokrouhlit nahoru na 2000 Ω. Díky tomu bude opět možné použít součástku s běžně dostupnou hodnotou. Celkové schéma zapojení senzoru vzdálenosti k Raspberry PI je znázorněno na obrázku (Obrázek 9).

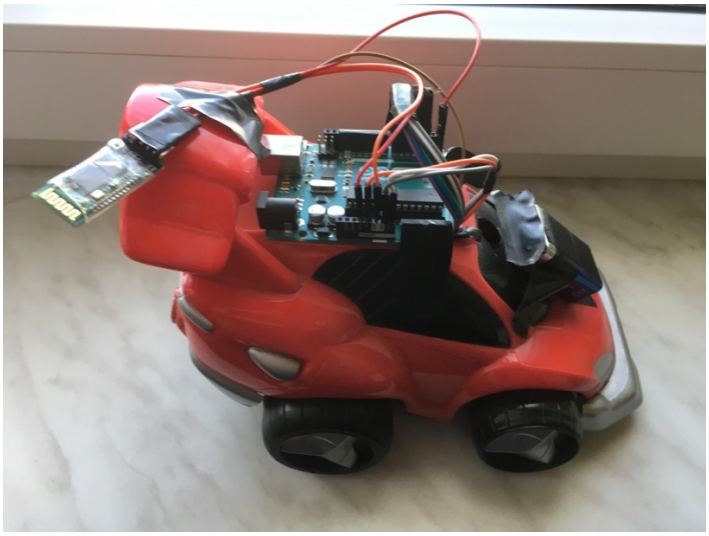


Obrázek 9 - Schéma zapojení senzoru vzdálenosti k Raspberry Pi

zdroj: vlastní tvorba

4.2.3 Model vozidla

Návrh počítá s využitím jednoduchého modelemu dálkově ovládaného vozidla. Model je založen na modelu programovatelného vozítka z roku 2000. Původním účelem tohoto vozítka byla podpora algoritmického myšlení dětí v předškolním věku. Toto vozítko bylo vybaveno jednoduchým tlačítkovým rozhraním, pomocí kterého byla vozítku nastavena posloupnost pohybů a jejich opakování.



Obrázek 10 - Model vozidla

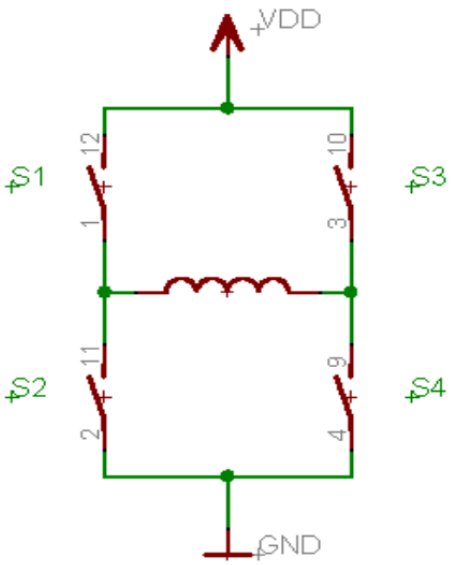
zdroj: vlastní tvorba

Přestavba byla realizována na základě zkušeností z vývoje podobného vozítka na předmětu Výpočetní inteligence. Úprava vozítka spočívala ve vyjmutí veškeré elektroniky a některých konstrukčních částí vozítka. Z původního vozítka zbyly v modelu pouze motory. Na základě předchozích zkušeností ze studia byla navržena nová řídící elektronika založená na vývojové desce Arduino UNO.

Arduino je open-source platforma pro vývoj elektroniky s jednoduchým hardwarem a softwarem. Arduino bylo vytvořeno v Ivrea Interaction Design Institut jako jednoduchý nástroj pro rychlé vytváření prototypů, zaměřené na studenty bez předchozích znalostí elektroniky a programování. Po rozšíření mezi širší komunitu se začalo Arduino měnit tak, aby adaptovalo nové potřeby svých uživatelů. Jedná se například o nová odvětví jako například svět IoT (Internet of Things), nositelné elektroniky a 3D tisku. Všechny návrhy Arduino desek jsou stejně jako software plně open-source. [***20***]

Arduino UNO je vývojová deska založená na 8-bitovém mikro kontroléru ATmega328P. Je vybavena čtrnácti digitálními vstupně-výstupními piny - šest z těchto pinů může být použito jako tzv. PWM výstupy, například pro řízení servomotorů nebo řízení rychlosti stejnosměrného motoru.. Mimo to je vybaveno 6 analogovými vstupy a ISCP piny pro připojení dalších kompatibilních modulů. Pro přenos informací z vývojové desky je možné použít USB rozhraní. Pro napájení slouží buď speciální konektor, nebo k tomu určené piny na desce. Deska potřebuje napájení 5 V, ale má integrovaný i dělič napětí, takže je možné napájet i zařízení pracující s 3 V logikou. [***21***]

Samotná vývojová deska Arduino umí ovládat i motory, ale není možné ovládat směr jejich otáčení, proto bylo přistoupeno k použití speciálního zapojení:



Obrázek 11 - Schéma zapojení H-můstku

*zdroj [**22**]*

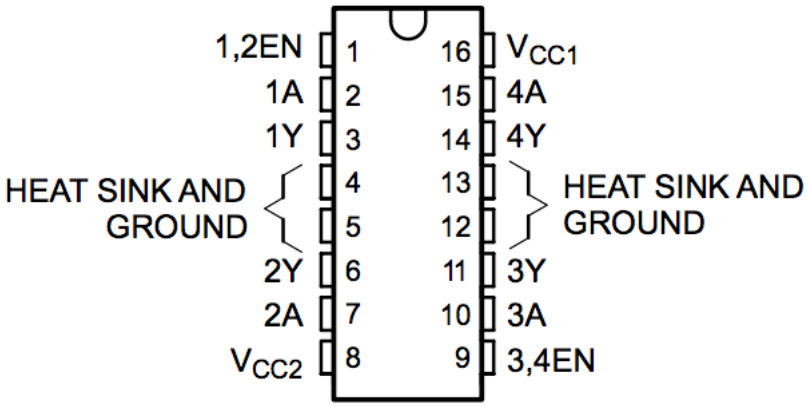
Tomuto zapojení se kvůli podobnosti s písmenem „H“ říká H-můstek. Jako spínací prvky bývají použity tranzistory. Vhodnou kombinací řídících signálů přiváděných na spínací prvky lze dosáhnout stavu, kdy bude na svorky motoru přivedeno napětí s jednou nebo druhou polaritou. Tím lze řídit rychlost a směr otáčení. Tato možnost řízení směru otáčení platí pouze pro motory s permanentním magnetem. Při použití H-můstku je nutné zajistit, aby v jednom okamžiku nebyly sepnuty oba spínací prvky na jednom rameni můstku (například S1 a S2 na obrázku). [***22***]

Díky existenci integrovaných obvodů není potřeba vytvářet toto zapojení z jednotlivých součástek. Je možné zakoupit již hotová zapojení integrovaná v jednom pouzdře (tzv. integrovaný obvod).

Integrovaný obvod je elektronická součástka realizující určité množství obvodových prvků neoddělitelně spojených na povrchu nebo uvnitř určitého spojitého tělesa, aby se dosáhlo ucelené funkce elektronického obvodu. [***23***]

Použití integrovaných obvodů usnadňuje použití různých zapojení v praxi. V případě H-můstku je možné zakoupit jeho integrovanou verzi označovanou jako L293D, která obsahuje H-můstky, kde jsou jako spínače použity tranzistory. Tento integrovaný obvod může být použit buď k ovládání dvou stejnosměrných motorů s permanentním magnetem nebo jednoho krokového motoru. Obvod dokáže

napájet motory napětím až 36 V. Hlavním omezením tohoto obvodu je maximální proud, který může skrze obvod protékat. Nominální hodnota proudu, který může protékat obvodem, činí 600 μA. Maximální špičkový neopakovatelný proud, který obvod nezničí, je maximálně 1,2 A po dobu maximálně 100 μs. Při delším průtoku proudu tímto obvodem nebo i kratší vystavení obvodu proudu vyššímu způsobí nevratné zničení tohoto obvodu.



Obrázek 12 - Integrovaný obvod L293D

http://www.ti.com/lit/ds/symlink/l293.pdf

Na obrázku (Obrázek 12) je zobrazeno schéma rozložení pinů integrovaného obvodu L293D v pouzdře DIP 16. DIP pouzdra se používají pro integrované obvody s nízkým stupněm integrace a malým počtem obvodů.

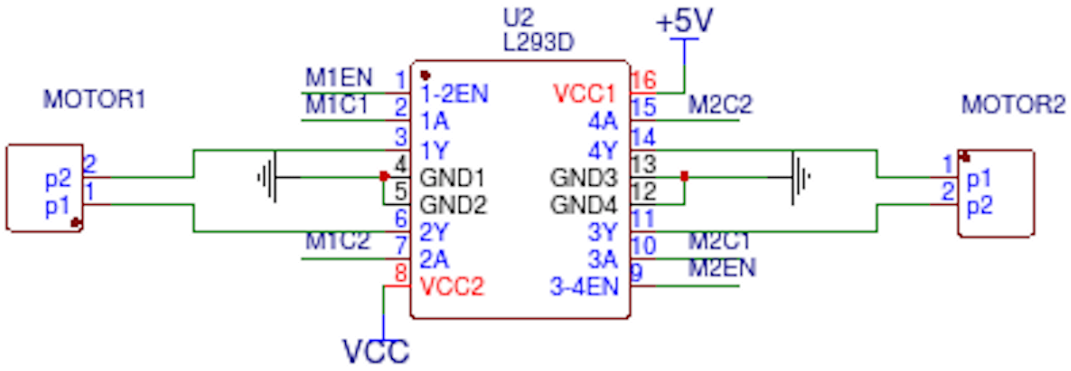
Určení jednotlivých pinů je popsáno v tabulce:

Tabulka 1 - Popis výstupů integrovaného obvodu L293D

Název pinu	Číslo pinu	Popis
1,2 EN	1	Řízení protékajícího proudu kanály 1 a 2.
1 – 4 A	2, 7, 10, 15	Ovládání směru proudu protékajícího můstkem.
1 – 4 Y	3, 6, 11, 14	Výstupy můstku pro připojení motoru.
3,4 EN	9	Řízení protékajícího proudu kanály 3 a 4.
GROUND	4, 5, 12, 13	Uzemnění a chlazení integrovaného obvodu.
V_{CC1}	16	Napájení logických obvodů – 5 V.
V_{CC2}	8	Napájení výstupů můstku pro napětí 4,5 – 36 V.

zdroj: http://www.ti.com/lit/ds/symlink/l293.pdf

Na základě těchto informací je možné vytvořit zapojení pro ovládání motorů modelu vozidla. Motory jsou připojeny k integrovanému obvodu podle následujícího schématu:



Obrázek 13 - Schéma zapojení L293D pro ovládání dvou motorů

zdroj: vlastní tvorba

Na schématu (Obrázek 13) je zobrazeno připojení motorů k integrovanému obvodu. Motor 1 je připojen k pinům 3 a 6. Jeho rychlost bude řízena PWM pinem Arduina, který bude připojen k pinu 1. Směr otáčení prvního motoru bude záviset na nastavených logických hodnotách na pinech 2, 7. Druhý motor je připojen k pinům 11 a14, rychlost bude ovládána pomocí pinu 9 a směr otáčení bude řízen pomocí pinů 10 a 15. Jak již z dokumentace vyplývá, integrovaný obvod potřebuje napájení logických struktur, to je realizováno pomocí připojení napětí o hodnotě 5 V k pinu 16. Napájení motorů zajišťuje zdroj napětí o hodnotě 9 V připojený k pinu 8.

Na vývojové desce Arduino jsou k ovládání motorů použity piny s označené čísly 2 až 7. Jejich role bude popsána při vysvětlení programu pro Arduino. K vývojové desce je rovněž připojen obvod s čipem HC-05, který je použit pro ovládání vozítka pomocí technologie Bluetooth. Celé schéma zapojení obvodů je znázorněno v příloze 1.

Programování 8-bitového mikroprocesoru nacházejícího se na vývojové desce Arduino je možné pomocí programu napsaném v jazyce C. Pro vývoj těchto programů je možné na webových stránkách výrobce stáhnout speciální editor,

který obsahuje kompilátory a další software, který slouží například pro nahrání výsledného kódu do vývojové desky.

Každý program pro Arduino musí nutně obsahovat dvě metody. První z nich je metoda *setup*, která se po spuštění programu na desce zavolá pouze jednou. Tato metoda slouží k inicializaci a úvodnímu nastavení vývojové desky. V této fázi se nastaví role jednotlivých pinů, inicializují se komunikační protokoly a podobně. Druhá metoda se nazývá *loop* a spouští se cyklicky po celou dobu běhu programu na vývojové desce. V této metodě program může reagovat na vstupy, na interní časovač nebo například na signály ze sériové linky.

```
1 | const int motorsCount = 2;
2 | const int controlPins1[motorsCount] = {2, 7};
3 | const int controlPins2[motorsCount] = {4, 5};
4 | const int enablePins[motorsCount] = {3, 6};
5 |
6 | int motorsEnabled[motorsCount] = {0, 0};
7 | int motorsSpeed[motorsCount] = {255, 255};
8 | int motorsDirection[motorsCount] = {1, 1};
9 | int actualMotorsDirection[motorsCount] = {1, 1;
10|
11| char message[4];
12| int messageIndex = 0;
13| char inChar;
```

Zdrojový kód 1 - Inicializace programu pro Arduino

První část programu je zobrazena v ukázce (Zdrojový kód 1). Na prvním řádku je nastavena konstanta určující počet motorů, který bude program ovládat. Při testování programu v rámci přechozího studia bylo reálně otestováno zapojení čtyř motorů, které bylo možné jednotlivě ovládat. Na dalších třech řádcích se nastavují pole obsahující jednotlivá čísla pinů do skupin, které reprezentují jejich určení. Každém poli jsou čísla seřazena tak, aby čísla pinů pro jeden motor měla stejný index v poli. Toto řešení usnadňuje přístup k jednotlivým pinům na základě čísla motoru. Na řádcích označených čísly 6 až 9 jsou pomocná pole pro uchování aktuálního stavu všech motorů. Celý program se chová jako konečný automat a jeho stavy jsou reprezentovány hodnotami nastavenými v těchto polích. Na posledních třech řádcích ukázky jsou připraveny pomocné proměnné pro komunikaci pomocí sériové linky.

```
1 | void setup() {
2 |  Serial.begin(9600);
3 |
4 |  for (int i = 0; i < motorsCount; i++) {
5 |    pinMode(controlPins1[i], OUTPUT);
6 |    pinMode(controlPins2[i], OUTPUT);
7 |    pinMode(enablePins[i], OUTPUT);
8 |    digitalWrite(enablePins[i], LOW);
9 |  }
10| }
```

Zdrojový kód 2 - Implementace metody setup v prograu pro Arduino

Následuje metoda *setup* (Zdrojový kód 2), která připraví Arduino k ovládání motorů. Na druhém řádku se nastavuje rychlost komunikace pro sériovou linku na rychlost 9600 b/s. Následuje nastavení módu pinů každého motoru. V případě ovládání motorů se všechny potřebné piny nastaví jako výstupní. A pin sloužící k ovládání rychlosti se nastaví na logickou „0“, aby se žádný z motorů nepohyboval. Nyní bude po částech rozebrána metoda *loop*, která je rozsáhlá, protože řeší nejen projekci aktuálních stavů do reálného chování, ale i komunikaci po sériové lince a s ní spojené operace.

```
1 | if (motorsDirection[i] == 1) {
2 |   if (actualMotorsDirection[i] == 0) {
3 |     digitalWrite(enablePins[i], LOW);
4 |     digitalWrite(controlPins1[i], HIGH);
5 |     digitalWrite(controlPins2[i], LOW);
6 |     digitalWrite(enablePins[i], motorsSpeed[i]);
7 |   } else {
8 |     digitalWrite(controlPins1[i], HIGH);
9 |     digitalWrite(controlPins2[i], LOW);
10|   }
11|  actualMotorsDirection[i] = 1;
12| }
```

Zdrojový kód 3 - Ukázka změny otáčení motorů

Ukázka (Zdrojový kód 3) zobrazuje část kódu, která se stará o změnu směru otáčení motorů na základě nastavené hodnoty v poli *motorsDirection*. Zobrazená část řeší přechod do směru otáčení „1“. Pro reprezentaci směru byla zvolena čísla, protože jsou praktická pro reprezentaci stavu otáčení. Kdyby byla zvolena písmena, mohlo by docházet k určitému nedorozumění v nastaveném směru otáčení oproti reálnému stavu. Toto nedorozumění může být způsobeno

obráceným zapojením motorů na piny. Na druhém řádku ukázky se kontroluje aktuální stav otáčení. Pokud se tento stav liší od aktuálního, tak se motor zastaví nastavením *enable* pinu na hodnotu LOW. Obrátí se polarity pomocí kontrolních pinů, které ovládají H-můstek. A znovu se motoru nastaví jeho rychlost a tím se spustí jeho otáčení. V případě, že je směr aktuálního otáčení stejný, tak se pouze zachová směr otáčení. Po úspěšném nastavení následuje uložení aktuálního stavu do proměnné reflektující aktuální stav otáčení motorů.

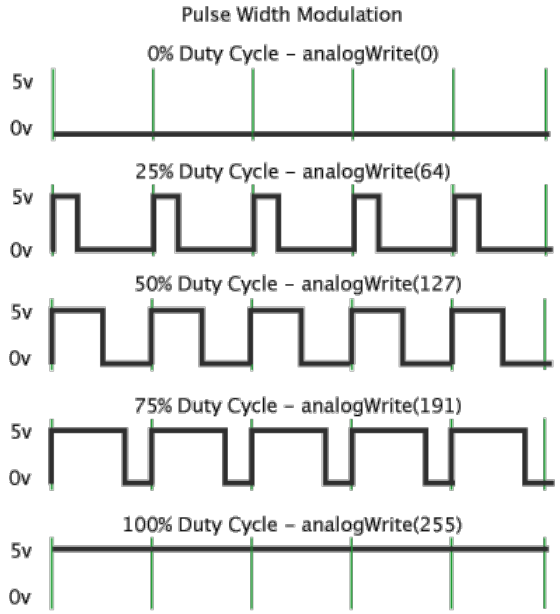
```
1| if (motorsEnabled[i] == 1) {
2|   analogWrite(enablePins[i], motorsSpeed[i]);
3| } else {
4|   analogWrite(enablePins[i], 0);
5| }
```

Zdrojový kód 4 - Ovládání enable pinů Arduina

V ukázce (Zdrojový kód 4) je zobrazeno ovládání *enable* pinů. Piny použité pro ovládání rychlosti otáčení motorů umožňují modulaci šířky pulzu.

Pulzně-šířková modulace neboli PWM (Pulse Width Modulation) je technika umožňující dosažení analogového výsledku při použití digitálních prostředků. Pomocí generátoru se vytvoří obdélníková vlna, kdy je signál zapnut a zase vypnut. Pomocí frekvence těchto vln je možné simulovat napětí od 0 do 5 V. Tohoto efektu je dosaženo časovými prodlevami mezi jednotlivými vlnami. Na vývojové desce Arduino se pro práci s PWM používá funkce *analogWrite*, která má dva parametry. První z nich je číslo pinu a druhý je celočíselná hodnota od 0 do 255. [***24***]

Obrázek 14 znázorňuje průběh pulzů při různých nastavených hodnotách.



Obrázek 14 - Průběh napěťových pulzů u PWM

*zdroj: [**24**]*

V metodě *loop* se kromě ovládání motorů rovněž zajišťuje komunikace skrze sériovou linku. Přenos dat pomocí sériové linky do vývojové desky je možný dvěma způsoby. První z nich je skrze USB rozhraní a druhá možnost je připojení k TX a RX pinům, které se nacházejí pod čísly 1 a 0. Část kódu zajišťující komunikaci směrem do zařízení pomocí sériové linky je zobrazena v následující ukázce:

```
1| if (Serial.available() > 0) {
2|   inChar = Serial.read();
3|   message[messageIndex] = inChar;
4|   messageIndex++;
5| }
```

Zdrojový kód 5 - Načítání znaků ze sérové linky na Arduinu

V ukázce (Zdrojový kód 5) je znázorněno načítání jednotlivých znaků do předem připraveného bufferu, protože v každé iteraci je možné načíst pouze jeden znak ze sériové linky. Komunikace s programem ve vývojové desce probíhá pomocí zpráv složených ze čtyř znaků. V aktuálním stavu je program schopen přijímat pouze zprávy sloužící k ovládání motorů.

Tabulka 2 - Popis příkazů pro ovládání modelu vozidla

MXE1	Zapnutí motoru
MXE0	Vypnutí motoru
MXSF	Nastavení plné rychlosti motoru, PWM hodnota 255
MXSH	Nastavení poloviční rychlosti motoru, PWM hodnota 127
MXDD	Nastavení směru „dopředu“
MXDR	Nastavení směru „dozadu“

Tabulka 2 zobrazuje všechny podporované zprávy, které je program pro vývojovou desku schopný zpracovat. Místo písmena „X“ je možné doplnit číslo motoru nebo 0. Ve zprávách jsou motory číslovány od 1 a číslo 0 slouží k ovládání všech motorů. Díky tomu je možné příkazem „M0E1“ spustit všechny motory nebo příkazem „M1SH“ zpomalit jeden motor a tím vyvolat pomalé smykové zatáčení a tak dále.

```
1 | if (messageIndex == 4) {
2 |   if (message[0] == 'M') {
3 |     char s = message[1];
4 |     int motorIndex = s - '0';
5 |     motorIndex--;
6 |     if (motorIndex < 0) {
7 |       for (int i = 0; i < motorsCount; i++) {
8 |         changeMotorStatus(i, message[2], message[3]);
9 |       }
10|     } else {
11|       changeMotorStatus(motorIndex, message[2], message[3]);
12|     }
13|   }
14|   messageIndex = 0;
15| }
```

Zdrojový kód 6 - Zpracování příchozí zprávy pro ovládání modelu

V ukázce (Zdrojový kód 6) je zobrazeno zpracování zprávy v okamžiku, kdy její délka dosáhne čtyř znaků. Následuje jednoduché zpracování, kdy se zpráva rozsekává na jednotlivé znaky a na jejich základě se provádí změna stavu jednoho nebo všech motorů. To, jestli bude ovládán jeden nebo všechny motory, je ovládáno druhým znakem za předpokladu, že první znak je písmeno „M“. Následuje převedení znaku na celé číslo. Toto číslo se zmenší o jedničku, protože komunikační jazyk čísluje motory od čísla 1, ale indexy pole v programu jsou číslovány od 0. Následuje rozhodnutí, zda získané číslo je menší než 0 a zda se

budou ovládat všechny motory, nebo jenom jeden. Pro změnu stavu motoru je připravena funkce *changeMotorStatus*, které je předán index motoru a zbylé dva znaky zprávy. Tato funkce jednoduše pomocí příkazu *switch* rozhodne o změnách v nastavení zapnutí, rychlosti a směru otáčení motoru. Kompletní program pro vývojovou desku Arduino je obsažen v elektronických přílohách práce.

4.3 Software

V této části se práce věnuje návrhu softwarového řešení pro navrhovaný systém. Pro implementaci softwarových částí systému budou použity dva programovací jazyky, a to Java a Python. Jazyk Java byl zvolen pro svoji obecnou rozšířenost a dobrou použitelnost na běžných počítačích. Python byl zvolen k použití na Raspberry Pi hlavně kvůli nízké paměťové náročnosti. Jazyk Python je na Raspberry Pi všeobecně podporovaným jazykem.

4.3.1 **Modul snímání okolí**

Modul snímání okolí má na starost příjem a zpracování dat z kamer a senzorů. V navrhovaném systému je tento modul rozdělen mezi dvě hardwarové části systému. První část, která bude komunikovat přímo s kamerou a senzory, bude umístěna v Raspberry Pi a druhá, která bude hlavně zpracovávat obrazová data z kamery, poběží jako součást aplikace na počítači.

O získávání informace o vzdálenosti předmětů před modelem vozidla se bude starat program pro Rasbperry Pi. Při měření vzdálenosti pomocí senzoru HC-SR04 se využívá fyzikálních vlastností zvuku. Senzor je vybaven vysílačem i detektorem ultrazvukových vln a při měření vzdálenosti se měří doba mezi vysláním signálu a návratem odražených vln zpět k senzoru. Pro výpočet se může použít standardní vzorec (Vzorec 3).

$$s = v \cdot t$$

Vzorec 3

kde s značí vzdálenost, v rychlost a t čas. Pro měření vzdálenosti pomocí ultrazvuku je potřeba vzorec upravit. Výsledná dráha by totiž zahrnovala cestu zvuku od senzoru k předmětu a zároveň i cestu zpět. Proto je potřeba výslednou rychlost vydělit dvěma. A tím získáváme vzorec (Vzorec 4).

$$s = \frac{v \cdot t}{2}$$

Vzorec 4

Při výpočtu bude použita přibližná rychlost šíření ultrazvuku ve vzduchu. Při pokojové teplotě může být jako výchozí hodnota považována konstanta 343 m/s [***25***]. Program, který bude pracovat se senzorem vzdálenosti a provádět potřebné výpočty, je napsán v jazyce Python. Tento jazyk byl zvolen, protože jsou pro něj připravené oficiální knihovny pro práci s GPIO sběrnice.

Inicializace programu, který bude pracovat s GPIO, je znázorněna v ukázce (Zdrojový kód 1).

```
1| import RPi.GPIO as GPIO
2|
3| GPIO.setmode(GPIO.BCM)
4|
5| GPIO.setup(23, GPIO.OUT)
6| GPIO.setup(24, GPIO.IN)
```

Zdrojový kód 7 - Inicializace programu pro práci s GPIO

Na řádku 1 se provádí import knihovny pro práci s GPIO Raspberry Pi. Na řádku 3 se nastavuje mód číslování pinů sběrnice. V tomto případě se jedná o BCM číslování, které je specifikováno GPIO sběrnicí. Druhý použitelný mód je BOARD, při němž čísla pinů vycházejí přímo z definice na desce. [***26***] V posledních dvou řádcích se nastavuje jednotlivým pinům, zda jsou vstupní nebo výstupní.

```
1 | GPIO.output(TRIG, False)
2 | time.sleep(1)
3 |
4 | GPIO.output(TRIG, True)
5 | time.sleep(0.00001)
6 | GPIO.output(TRIG, False)
7 |
8 | while GPIO.input(ECHO) == 0:
9 |   start_time = time.time()
10|
11| while GPIO.input(ECHO) == 1:
12|   end_time = time.time()
13|
14| duration = end_time - start_time
15|
16| distance = (SPEED_OF_SOUND * duration) / 2.0
```

Zdrojový kód 8 - Implementace měření vzdálenosti na Raspberry Pi

Samotné měření vzdálenosti zobrazuje ukázka (Zdrojový kód 8). Na prvním řádku je nastavena hodnota výstupu ovládající TRIG pin senzoru. Následuje pozastavení programu na jednu vteřinu, aby se senzor připravil. Následuje vyslání ultrazvukového signálu po dobu 10 μs. Na osmém řádku je cyklus, který čeká, dokud signál nedorazí zpět k senzoru. Ve chvíli, kdy signál dorazí k senzoru, program přechází do dalšího cyklu, který změří konečný čas, kdy signál dorazí celý zpět. Rozdíl těchto časů je doba trvání cesty signálu od senzoru k překážce a zpět. Na posledním řádku probíhá samotný výpočet odhadu vzdálenosti.

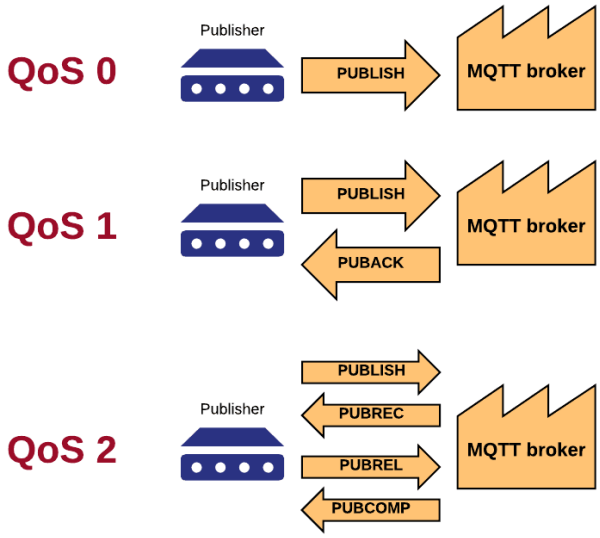
Po úspěšném výpočtu vzdálenosti je potřeba výsledek zpřístupnit dalším modulům. Pro publikaci informace o vzdálenosti je použit protokol MQTT.

MQTT je komunikační protokol pro komunikaci „machine-to-machine“. Je navržen jako extrémně nenáročný protokol pro předávání zpráv pomocí centrálního brokera fungující na principu „publish/subscribe”. Princip návrhu MQTT protokolu si zakládá na minimalizaci nároků na datový tok v síti a na hardwarové prostředky koncových zařízení při zachování určité míry spolehlivosti doručení zpráv. [***27***]

Jak již bylo zmíněno komunikace probíhá prostřednictvím brokera. To je centrální bod, který se stará o výměnu zpráv. Předávání zpráv je rozděleno do tzv. témat (topic). Jednotlivá zařízení mají možnost buď publikovat zprávy (režim „publisher“) v daném tématu, anebo se přihlásit k odběru jednoho nebo více témat (režim „subscriber“). Jedno zařízení samozřejmě může najednou v některých tématech publikovat zprávy a v jiných být přihlášeno k odběru. Samotný obsah zpráv není protokolem vyžadovaný. Obsah zprávy je protokolem vnímán jako standardní binární data, která je potřeba přenést. Pomocí protokolu je možné přenášet zprávy ve formátu JSON, text nebo jiná data v binární podobě. [***28***]

MQTT protokol rovněž podporuje řízení Quality of Service, a to na 3 úrovních. První úroveň je 0 označovaná jako „at most one“ nebo také „fire and forget“. Při takovém nastavení pošle publisher zprávu typu PUBLISH a na nic nečeká. Broker ji stejným způsobem pošle odběratelům daného tématu. Na úrovni 1 (také označována jako „at least once“) pošle publisher brokeru zprávu typu PUBLISH a čeká. Broker zprávu přijme a pošle ji odběratelům (také PUBLISH) a čeká na potvrzení od příjemce. Ve chvíli, kdy odběratelé potvrdí přijetí zprávou typu PUBACK, broker zprávu odstraní a pošle PUBACK publisherovi zpět. Publisher ví,

že zpráva prošla brokerem, a může ji zahodit. Při nastavení úrovně 2 (exactly once) pošle publisher brokeru zprávu PUBLISH. Ten ji, stejně jako v předchozím případě, přijme, pošle ji odběratelům a publisherovi vrátí zprávu PUBREC (tedy potvrzení přijetí). Publisher odpoví zprávou PUBREL, broker zprávu smaže a potvrdí zprávou PUBCOMP. [***28***]



Obrázek 15 - Quality of Service v MQTT protokolu

*zdroj: [**28**]*

Publikace informace o vzdálenosti objektů před modelem bude implementována do stejného programu jako měření vzdálenosti. Následovat bude popis částí programu, které se starají o přenos infor mace skrze MQTT protokol.

```
1 | import paho.mqtt.client as mqtt
2 |
3 | # Handlers
4 | def on_connect(mqttc, obj, flags, rc):
5 |   print("Connected to MQTT server")
6 |
7 | # MQTT Client setup
8 | mqtt_client = mqtt.Client(MQTT_CLIENT_NAME)
9 | mqtt_client.on_connect = on_connect
10|
11| mqtt_client.connect(MQTT_BROKER_HOST, MQTT_BROKER_PORT, 60)
12|
13| mqtt_client.loop_start()
14|
15| mqtt_client.publish(MQTT_TOPIC, distance_string, qos=0)
```

Zdrojový kód 9 - Implementace publikování zpráv MQTT protokolu

Ukázka (Zdrojový kód 9) zobrazuje základní implementaci publikování zpráv do tématu v MQTT serveru. Na řádku 1 se nachází import potřebné knihovny. V návrhu se počítá s použitím MQTT knihovny od tvůrců známého IDE Eclipse. Na řádcích 4 a 5 je definována handler funkce, která je zavolána po úspěšném připojení k MQTT brokeru. Následuje nastavení klienta. Řádek 6 znázorňuje vytvoření nové instance třídy Client z importované knihovny. Název klienta sice pro vytvoření instance není povinný, ale pro komunikaci skrze MQTT protokol ve verzi „3.1.1“ je nutný. Následuje přiřazení event handleru pro událost „on_connect“, který bude informovat o úspěšném připojení k serveru. Na řádku 11 je volání funkce „connect“, kterému jsou předány potřebné parametry. Tyto parametry jsou URL serveru, port a počet vteřin pro připojení (tzv. „timeout“). Následuje spuštění komunikační smyčky vlákna. Na řádku 15 je zobrazena samotná publikace textové informace o vzdálenosti a v neposlední řadě nastavení „quality of service“ dané zprávy. V tomto případě je optimální nastavení „0“ - „fire and forget“, protože zjišťování vzdálenosti bude v probíhat 2x za vteřinu a potvrzovací zprávy by zbytečně vytěžovaly přenosovou linku.

Pro přenos obrazu je použit nástroj mjpg-streamer. Mjpg-streamer je aplikace pro příkazovou řádku, která kopíruje snímky v JPEG formátu z jednoho nebo více vstupních rozšíření do více výstupních rozšíření. Původně byl navržen pro embedded zařízení s velmi omezenými prostředky ve smyslu paměti RAM a výkonu CPU. Jeho předchůdce „uvc_streamer“ byl vytvořen, protože kamery

kompatibilní s Linux-UVC produkovaly data ve formátu JPEG. To umožnilo rychlé a výkonné M-JPEG streamy dokonce ze zařízení, na kterých běžel OpenWRT. [***29***]
Pro spuštění streamu z webové kamery je použit následující příkaz:

```
$ mjpg_streamer -o "output_http.so" -i "input_uvc.so -r "VGA" -fps 30"
```

Zdrojový kód 10 - Shell příkaz pro spuštění mjpg_streamer

Použitím přepínače „–o“ je nastaven výstupní plugin. V tomto případě je použit plugin „output_http.so“, který zpřístupní MJPEG stream na portu 8080. Přepínačem „i“ se nastavuje vstupní plugin. Pro zachycení streamu z kamery slouží plugin „input_uvc.so“. Tomuto pluginu je však potřeba nastavit další parametry. Prvních z nich „-r“ určuje rozlišení obrazu, které bude z kamery získáváno. V tomto případě je nastaveno rozlišení VGA (640x480px), které je pro účely tohoto projektu dostačující. Druhým parametrem je přepínač „fps“, pomocí kterého je nastavena snímková frekvence.
K získávání obrazových dat ze streamu, který poskytuje kamera z Rasbperry Pi, je použita knihovna *Webcam Capture API*. Tuto knihovnu vytvořil a spravuje Bartosz Firyn, který na webovém portálu GitHub používá přezdívku „sarxos“. Mezi hlavní přednosti této knihovny patří podpora hlavních operačních systému na 32-bitových, 64-bitových i ARM architekturách. Díky tomu jme možné tuto knihovnu používat i na Raspberry Pi. Knihovna umí získávat obrazová data z vestavěných kamer, USB kamer, ale i síťových kamer, které vysílají data jako MJPEG stream. [***30***]
Hlavní důvodem pro použití zmiňované knihovny byla možnost získávání obrazových dat z MJPEF streamu. Modul snímání okolí je navržen obecně jako rozhraní, které lze implementovat různými způsoby. Definice tohoto rozhraní v jazyce Java je znázorněna v ukázce (Zdrojový kód 11).

```
1 | public interface PerceptionModule {
2 |
3 | interface PerceptionListener {
4 |   void onImageCaptured(BufferedImage image);
5 |
6 |   void onDistanceChange(float distance);
7 |  }
8 |
9 |  void startPerception();
10|
11|  void setPerceptionListener(PerceptionListener listener);
12| }
```

Zdrojový kód 11 - Implementace rozhraní modullu snímání okolí v jazyce Java

Rozhraní nazvané *PerceptionModule* definuje dvě povinné metody. První z nich je metoda *startPerception*, která slouží k zahájení snímání. Druhá metoda je nazvána *setPerceptionListener* a přijímá instanci objektu, který implementuje rozhraní *PerceptionListener*. Jak již název napovídá, tak na modul snímání okolí bude aplikován návrhový vzor Observer (Pozorovatel). Modul snímání se napojí na obrazový stream z kamery a připojí se jako subscriber k MQTT serveru. Následně bude informovat svého pozorovatele o těchto událostech s potřebnými daty. Proto rozhraní Perception Listener definuje dvě metody, které musí pozorovatel nutně implementovat. První z nich je metoda *onImageCaptured*, která přijímá objekt typu *BufferedImage*. Tato metoda slouží k předaní obrazových dat pro další zpracování. Druhou metodou je metoda *onDistanceChange* sloužící k předání informace o vzdálenosti naměřené senzorem.

Implementace modulu snímání okolí je obsažena v elektronických přílohách pod názvem *RaspiPerceptionModule.java*. Konstruktor této třídy přijímá dva parametry typu *String*. V prvním parametru očekává adresu, na které se nachází MJPEG stream z kamery. Druhým parametrem se nastavuje server, na kterém je spuštěn MQTT broker. Součástí konstruktoru je i inicializace připojení ke streamu z kamery. Metoda *startPerception* spouští dvě vlákna. První vlákno každých 200ms stáhne aktuální obrázek z kamery. Následně provede úpravu získaného obrázku pro zpracování neuronovou sítí. Úprava obrázku probíhá pomocí třídy *ImageTransformer*. Tato třída byla vytvořena speciálně pro proudové zpracování obrazu. Při inicializaci této třídy se připraví instance transformačních operací. Díky předpřipraveným operacím se při volání metody *transform* nemusí tyto operace

vytvářet znovu. To šetří cenný strojový čas, který je při proudovém zpracování dat kritický. Úprava obrazu spočívá v jeho převedení odstínů šedi a ořezu pouze na spodní polovinu. Výsledný obraz má rozměry 640 na 240 pixelů Po úpravě je obrázek předán instanci listeneru pomocí metody *onImageCaptured.* Druhé vlákno se připojí k MQTT brokeru a zaregistruje se k odběru tématu „obstacle“.

Pro práci s MQTT protokolem je použita knihovna *mqtt-client*, která byla vytvořena firmou FuseSource. Tato knihovna poskytuje API k protokolu MQTT. Je licencována Apache Licence 2.0. Mezi její výhody patří automatická znovu-připojení k MQTT serveru a obnovení *session* klienta v případě chyby spojení se serverem. Aplikační rozhraní knihovny podporuje klasický (blokující) přístup k událostem, přístup za použití *futures* i přístup založený na *callback* funkcích. [***31***]

V implementovaném modulu snímání okolí je aplikován přístup založený na *callback* funkcích. Při každé přijaté zprávě předá naměřenou vzdálenost instanci listeneru pomocí metody *onDistanceChange.*

4.3.2 **Modul plánování**

Úkolem modulu plánování je na základě dat získaných ze senzorů, rozhodnout o následujícím pohybu vozidla. Modul plánování je v návrhu jako jediný čistě softwarový a kompletně implementovaný pouze v jednom jazyce. Pro účel návrhu systému je modul plánování definován pouze jako rozhraní. Díky tomu je možné jednoduše vyměňovat rozhodovací algoritmy pro model, aniž by muselo být nutné zasahovat do ostatních modulů. Definice tohoto rozhraní v jazyce Java je zobrazena v ukázce (Zdrojový kód 12).

```
1| public interface PlanningModule {
2|   KeyConfiguration plan(BufferedImage image);
3| }
```

Zdrojový kód 12 - Definice rozhraní modulu plánování v jazyce Java

Rozhraní definuje pouze jednu povinnou metodu. Metoda je nazvána *plan* a přijímá jediný parametr typu *BufferedImage*. Metoda vrací instanci třídy K*eyConfiguration* obsahující rozhodnutí neuronové sítě.

V první fázi návrh předpokládá použití modulu plánování založeném na umělých neuronových sítích. Umělá neuronová síť je výpočetní model používaný v umělé

inteligenci. Umělé neuronové sítě jsou inspirovány chováním biologických neuronových sítí, jako třeba lidského mozku. Umělé neurony a umělé neuronové sítě mohou provádět aritmetické operace, kde jednotlivé umělé buňky odpovídají neuronům, aktivace odpovídají míře vysílání signálů neuronů, propojení odpovídají synapsím a váhy propojení odpovídají síle jednotlivých synapsí biologické neuronové sítě. V neuronových sítích jsou neurony propojeny přímými propojeními. Neurony a propojení určují topologii sítě. Každé propojení má svoji váhu, která specifikuje vliv mezi neurony. Pozitivní hodnota váhy indikuje posílení a negativní váha reprezentuje inhibici. Váhy jednotlivých propojení určují chování sítě. [***32***]

Hlavním významem neuronových sítí je jejich schopnost učit se z jejich prostředí. V kontextu neuronových sítí je učení definováno jako proces, kterým se volné parametry neuronových sítí adaptují pomocí kontinuální simulace prostředí. Obecně existují dva přístupy k učení neuronových sítí:

- **Učení s učitelem** - Během učení je do neuronové sítě poslán vstup a následně je získána odpověď sítě. Získaná odpověď je porovnána s předpokládanou odpovědí. Pokud se získaná odpověď liší od správného řešení, tak neuronová síť vygeneruje chybový signál. Tento signál je použit k výpočtu úpravy, která by měla být udělána ve váhách jednotlivých synapsí neuronové sítě tak, aby se aktuální odpověď shodovala se správným řešením. [***32***]
- **Učení bez učitele** – Jak již název napovídá, tak tento typ učení nepotřebuje „učitele“ – předpokládanou odpověď pro každý vstup. Během učící fáze neuronová síť přijímá vstupní vzory a svévolně je organizuje do kategorií. Když jsou následně do sítě odeslána vstupní data, neuronová síť poskytne odpověď, která indikuje kategorii, do které vstupní data spadají. I když učení bez učitele nepotřebuje předpokládanou odpověď pro daný vstup, tak ale potřebuje pokyny k určení, jak formovat jednotlivé skupiny. Seskupování může být založeno na tvaru, barvě, konzistenci materiálu nebo jiných vlastností objektu. [***32***]

Návrh řešení předpokládá aplikaci učení neuronové sítě s učitelem. Testovací data budou získávána pomocí speciálního programu. Tento program je součástí elektronických příloh pod názvem *TrainingDataCapturer.java*. Tento program bude sloužit jako ovladač vozidla, kde obsluha uvidí obraz z kamery a bude moci pomocí klávesnice ovládat pohyby vozítka. Při spuštění nahrávání jsou ukládány jednotlivé snímky z kamery na vozítku a k nim asociována konfigurace kláves, které byli při daném snímku aktivní. Při ukončení nahrávání se tato konfigurace uloží do speciálního XML souboru, jehož struktura je zobrazena v ukázce (Zdrojový kód 13).

```
<imageKeyStorage>
 <image name="image7.jpg">
  <key-config up="false" left="true" right="false" down="false"/>
 </image>
 <image name="image26.jpg">
  <key-config up="false" left="false" right="false" down="true"/>
 </image>
 ...
</imageKeyStorage>
```

Zdrojový kód 13 - Výstup nástroje pro získávání testovacích dat

Vygenerovaný soubor je uložen společně se zaznamenanými obrázky ve specifikovaném adresáři.

V dalším kroku učení je potřeba pomocí těchto dat začít učit neuronovou síť. K tomuto účelu je připraven další program, který je rovněž součástí elektronických příloh pod názvem *Teacher.java*. Tento program načte data z XML souboru se specifikací kláves. Následně připraví a nakonfiguruje novou neuronovou síť. Program používá implementaci neuronové sítě, která je součástí knihovny OpenCV. OpenCV (Open Source Computer Vision Library) je open-source knihovna pro počítačové vidění a strojové učení. Knihovna OpenCV byla vytvořena, aby poskytovala společnou infrastrukturu pro aplikace využívající počítačové vidění a urychlila použití strojového vnímání v komerčních produktech. Knihovna obsahuje více než 2500 optimalizovaných algoritmů, které zahrnují nejen klasické, ale i nejmodernější algoritmy pro počítačové vidění a strojové učení. Tyto algoritmy mohou být použity pro detekci a rozpoznávání tváří, identifikaci objektů a pro mnoho dalších aplikací pracujících s obrazem, videem, 3D prostorem nebo

rozšířenou realitou. Knihovna je implementována nativně v jazyce C++ a poskytuje rozhraní pro C, C++, Python, Java a MATLAB. [***33***]

V programu je použito řešení pomocí knihovny *javacv* od tvůrce *bytedeco*, která propojuje nativní knihovnu s JVM (Java Virtual Machine) pomocí JNI (Java Native Interface).

```
1 | Mat layers = new Mat(5,1, CV_32SC1);
2 | IntRawIndexer ki = layers.createIndexer();
3 | ki.put(0,0,153600)
4 |   .put(1,0,256)
5 |   .put(2,0,64)
6 |   .put(3,0,32)
7 |   .put(4,0,4);
8 |
9 | TermCriteria criteria = new TermCriteria();
10| criteria.maxCount(500);
11| criteria.epsilon(0.0001);
12| criteria.type(CV_TERMCRIT_NUMBER | CV_TERMCRIT_EPS);
13|
14| ANN_MLP mlp = ANN_MLP.create();
15| mlp.setLayerSizes(layers);
16| mlp.setTermCriteria(criteria);
17| mlp.setTrainMethod(ANN_MLP.BACKPROP);
18| mlp.setBackpropWeightScale(0.001);
19| mlp.setBackpropMomentumScale(0.0);
```

Zdrojový kód 14 - Implementace vytvoření neuronové sítě

Samotná implementace vytvoření neuronové sítě pomocí knihovny OpenCV je zobrazena v ukázce (Zdrojový kód 14). První část ukázky je věnována konfiguraci jednotlivých vrstev neuronové sítě. Hlavní stavební jednotkou je v případě knihovny OpenCV matice. Všechna data, která se mají zpracovávat jsou knihovně předávána ve formě matic. Ať už se jedná o konfiguraci vrstev neuronové sítě nebo obrazová data. Na první řádku je vytvořena instance matice o pěti řádcích a jednom sloupci. Každý prvek v matici je tvořen celým číslem o bitové délce 32 bitů. Přímý přístup k jednotlivým položkám matice není možný. Je potřeba vytvořit instanci tzv. *indexeru*, který zajišťuje přímý přístup k prvkům matice. Pomocí instance *indexeru* je možné na zadané adresy v matici nastavit hodnoty jednotlivých prvků matice. V případě implementace neuronové sítě pro ovládání modelu vozidla byla zvolena konfigurace čítající pět vrstev. První vrstva obsahuje

153600 buněk. Tento počet vychází z rozměrů obrazu, který je získáván z kamer na modelu vozidla.

Rozměr obrazu získávaný z kamery má rozměry 640 na 240 pixelů. Jednoduchým výpočtem $640 \cdot 240 = 153600$ získáme velikost první vrstvy neuronové sítě. Rozměry následujících vrstev byly určeny tak, aby byly násobkem čísla čtyři. Poslední pátá vrstva již reprezentuje řešení, které se skládá ze čtyř hodnot. Tyto hodnoty reprezentují směr plánovaného pohybu vozidla. Na řádcích 9 až 12 je zobrazeno nastavení zastavovacích kritérií pro iterační algoritmus učení neuronové sítě. Instanci neuronové sítě je nastaven maximální počet iterací každého učení (*maxCount*) a požadovaná přesnost změny v parametrech, které algoritmus učení zastaví (*epsilon*). Na řádku 12 je nastaven typ kritérií. V případě této neuronové sítě je zvolen typ, který reflektuje předchozí nastavení. Jedná se o kombinaci omezení počtu iterací a velikosti změny ve váhách učené sítě. Ve třetí části ukázky je samotná inicializace instance neuronové sítě. Ta se provádí pomocí metody *create*, která je součástí třídy ANN_MLP. Následuje nastavení vrstev sítě a zmiňovaných kritérií. Důležitou součástí počátečního nastavení je specifikace metody učení, kterou bude neuronová síť používat. V případě neuronové sítě použité pro navrhovaný model vozidla bude použit *Backpropagation* algoritmus.

Algoritmus *Backpropagation* byl vyvynut pro trénovaní vícevrstvých Perceptronových sítí. Perceptron je nejjednodušší forma neuronové sítě, která umí klasifikovat data do dvou tříd. Skládá se z jednoho neuronu s více nastavitelnými vahami. Idea *Backpropagation* algoritmu je v trénovaní sítě pomocí propagace výstupní chyby zpět skrze všechny vrstvy. Chyba slouží k vyhodnocení derivátů chybové funkce vzhledem k váhám, které lze pak upravit. Algoritmus *Backpropagation* se skládá ze dvou fází. První z nich se nazývá *feedforward* funkce a během ní se aplikují vstupy, vyhodnotí aktivační funkce a uloží chyba. Ve druhé části se spočítají úpravy a aktualizují se váhy. Viditelné jsou pouze chyby ve výstupní vrstvě, respektive mohou být spočítány přímo. Chyby ve skrytých vrstvách jsou propagovány z okolních vrstev. [***32***]

Na řádku 18 a 19 ukázky (Zdrojový kód 14) je zobrazeno nastavení parametrů samotného *Backpropagation* algoritmu. První z nich slouží k nastavení váhového gradientu a druhý k nastavení velikosti momenta. Momentum je používáno u

Backpropagation algoritmu ke zrychlení učení neuronové sítě. V případě navrhované neuronové sítě budou použity stejné hodnoty, které použil Zheng Wang ve svém projektu. To pro začátek umožní dosáhnout dobrých výsledků a při dalším výzkumu bude možné více optimalizovat řešení založené na neuronových sítích.

Ve chvíli, kdy je inicializována neuronová sít, je potřeba připravit data pro učení. Obrazová data je potřeba připravit do formy matice. Knihovna OpenCv nabízí dvě možnosti, jak obrazová data v matici reprezentovat. První z nich je, že veškerá obrazová data budou v jednom řádku matice. Druhá možnost je ta, při které budou veškerá obrazová data v jednom sloupci matice. V programu byla zvolena možnost první, tedy celý obrázek bude na jednom řádku matice. Výhoda tohoto řešení je v tom, že je možné do jedné matice načíst více řádků dat. Mimo vstupní matici obrazových dat je potřeba rovněž připravit matici předpokládaných výstupů. U té je možné vybírat také ze sloupcové nebo řádkové reprezentace výsledků. Je nutné zdůraznit, že systémy reprezentace musí být u obou matic totožné. Kompletní implementace přípravy dat je obsažena v elektronických přílohách pod názvem *Teacher.java*.

```
1 | Mat imageMat = new Mat(storage.size(),153600, CV_32F);
2 | FloatRawIndexer imageMatIndexer = imageMat.createIndexer();
3 |
4 | Mat values = new opencv_core.Mat(storage.size(), 4, CV_32F);
5 | FloatRawIndexer valueIndexer = values.createIndexer();
6 |
7 | // naplneni vstupnich a vystupních dat
8 |
9 | mlp.train(imageMat, opencv_ml.ROW_SAMPLE, values);
10| mlp.save("output/model.xml");
```

Zdrojový kód 15 - Implementace učení neuronové sítě

Následuje samotné učení neuronové sítě. To se provádí pomocí volání metody *train* instance neuronové sítě, je znázorněno na řádku 9 v ukázce (Zdrojový kód 15). Jako první parametr se metodě předává matice vstupů. Vstupní matice je inicializována na prvním řádku. Obsahuje počet řádků, který odpovídá počtu obrázků v testovacích datech. Počet sloupců je pevně daný a ten odpovídá násobku rozměrů zpracovaných obrázků. Datový typ pole je vyžadován implementací neuronové sítě. Je nutné, aby obě matice (vstupní i výstupní) měly prvky tvořené

čísly s plovoucí desetinnou čárkou o délce 32 bitů. Druhým parametrem funkce *train* je způsob reprezentace dat ve vstupní a výstupní matici. Třetím parametrem je matice předpokládaných výstupních hodnot. Tato matice má stejný počet řádků jako matice vstupů, ale jen čtyři sloupce reprezentující řešení.

Na řádku 10 ukázky (Zdrojový kód 15) je znázorněna metoda, která exportuje konfiguraci vah všech perceptronů a uloží je do souboru ve formátu XML. Tento soubor obsahuje nejen konfiguraci vah, ale nastavení celé neuronové sítě. Díky tomu je možné jednoduše rekonstruovat celou neuronovou síť pro další použití. Velikost exportovaného souboru závisí na konfiguraci vrstev sítě. Velikost generovaného souboru u navrhované sítě je kolem 1 GB.

Navrhované rozhraní pro plánovací modul definuje pouze jednu povinnou metodu. Tato metoda je nazvána *plan*, přijímá jeden parametr typu BufferedImage a vrací instanci třídy *KeyConfiguration.* Třída *KeyConfiguration* se rovněž používá při učení neuronové sítě pro ukládání stisknutých kláves pro jednotlivé obrázky. Implementace plánovacího modulu založeném na neuronové síti je demonstrována v souboru *NeuralNetworkPlanningModule.java*. Konstruktor této třídy přijímá jeden parametr – cestu k souboru s uloženými vahami naučené neuronové sítě. Přímo v konstruktoru dochází k inicializaci neuronové sítě a načtení uložených vah ze souboru. Vzhledem k velikostem neuronových sítí je načítání neuronové sítě časově a paměťově náročné. Vytvořená třída má kromě konstruktoru pouze jednu metodu, kterou předepisuje rozhraní *PlanningModule*. Implementaci metody *plan* zobrazuje ukázka (Zdrojový kód 16).

```
1| public KeyConfiguration plan(BufferedImage image) {
2|   Mat result = new Mat(1,4, CV_32F);
3|   model.predict(
4|     ImageUtils.imageToRowMat(image),
5|     result,
6|     0
7|   );
8|   // zbytek implementace
9| }
```

Zdrojový kód 16 - Implementace metody plan modulu plánování

Před samotnou predikcí je potřeba připravit instanci matice, která bude sloužit pro uložení výsledků predikce. Následuje volání metody *predict*, která jako první

parametr přijímá vstupní data. Vstupními daty je v případě implementované metody obrázek, který je pomocí vlastní knihovní funkce *imageToRowMat* převeden do matice vhodné pro zpracování neuronovou sítí. Druhým parametrem funkce pro predikci je inicializovaná matice pro výsledky. Třetí parametr je určen pro nastavení příznaků funkce. Tento parametr funkce bohužel není nikde zdokumentován a nativní funkce knihovny OpenCV, která je volána, tento parametr nemá. Proto byla zvolena hodnota 0, která by dle všeobecného názoru neměla nijak ovlivňovat predikci. V další části kódu, která je v ukázce z praktických důvodů vynechána, se najde pozice maximální hodnoty v matici výsledků. Na základě této pozice se inicializuje instance třídy *KeyConfiguration*, která je metodou navrácena.

4.3.3 Modul ovládání vozidla

Účelem modulu ovládání vozidla je vytvořit abstraktní rozhraní pro ovládání modelu vozidla tak, aby bylo možné v budoucnu používat rozdílná vozidla s různými komunikačními rozhraními. Definice rozhraní modulu v jazyce Java je zobrazena v ukázce (Zdrojový kód 17). Rozhraní definuje pouze jednu povinnou metodu *drive*. Metoda přijímá jeden parametr. Tímto parametrem je instance třídy *KeyConfiguration*, která reprezentuje směr, kterým řidič (neuronová síť) chce jet.

```
1| public interface VehicleControlModule {
2|   void drive(KeyConfiguration keyConfiguration);
3| }
```

Zdrojový kód 17 - Definice rozhraní modulu ovládání vozidla v jazyce Java

Modul ovládání vozidla využívá pro ovládání modelu technologii Bluetooth. Implementace modulu využívá knihovnu BlueCove. Tato knihovna implementuje standard JSR-82 (Java APIs for Bluetooth), který byl navržen komunitou v roce 2000 a jeho poslední aktualizace byla provedena v roce 2010. Knihovna BlueCove je multiplatformní a lze ji používat na všech hlavních platformách. [***34***]

Připojení k hardwarovému modulu HC-05, který je použit na modelu vozidla, se skládá ze dvou částí. V první části je potřeba dané zařízení vyhledat v okolí. Následně je potřeba se připojit k službě zařízení, která nám umožní přenášet binární data do zařízení.

```
1| LocalDevice localDevice = LocalDevice.getLocalDevice();
2| DiscoveryAgent agent = localDevice.getDiscoveryAgent();
3| agent.startInquiry(DiscoveryAgent.GIAC, listener);
4|
5| synchronized(lock) {
6|   lock.wait();
7| }
```

Zdrojový kód 18 - Implementace objevování okolních Bluetooth zařízení

Ukázka (Zdrojový kód 18) znázorňuje implementaci první fáze. Na prvním řádku se získá instance třídy *LocalDevice*. Ta reprezentuje referenci na aktivní Bluetooth zařízení v počítači, na kterém je program spuštěn. Na dalším řádku je získána reference na agenta, pomocí kterého se provádí objevování okolních zařízení. K tomuto účelu se používá služba poskytovaná operačním systémem. Na třetím řádku se volá metoda agenta *startInquiry*, která spustí dotazovaní na okolní Bluetooth zařízení. Metoda přijímá dva parametry. První parametr je typ dotazování, který bude agent používat. V případě této implementace bude použit typ GIAC (General/Unlimited Inquiry Access Code). Druhým parametrem je listener, který implementuje rozhraní *DiscoveryListener*. Jelikož je metoda *startInquiry* asynchronní, je potřeba před dalším postupem opět synchronizovat vlákna. To se prování konstruktem *synchronized*, kterému se předá instance synchronizovaného objektu. A v kódovém bloku se nad tímto objektem zavolá metoda *wait*, která zajistí to, že kód bude pokračovat, až jedno nebo více vláken skončí svoji činnost.

```
1 | @Override
2 | public void deviceDiscovered(
3 |       RemoteDevice device,
4 |       DeviceClass deviceClass) {
5 |   String address = remoteDevice.getBluetoothAddress()
6 |   if (address.equals(carBluetoothAddress)) {
7 |     carBluetoothDevice = remoteDevice;
8 |     System.out.println("Car bluetooth device found");
9 |     }
10|   }
11|
12| @Override
13| public void inquiryCompleted(int i) {
14|   synchronized(lock){
15|     lock.notify();
16|   }
17| }
```

Zdrojový kód 19 – Implementace dvou metod rozhraní Discovery Listener

Ukázka (Zdrojový kód 19) zobrazuje implementaci dvou metod rozhraní *DiscoveryListener*, které jsou volány v průběhu životního cyklu objevování okolních zařízení.

První metoda nazvaná *deviceDiscovered* je volána v případě, že agent našel nějaké zařízení. Prvním parametrem této metody je reference na instanci třídy *RemoteDevice*, která obsahuje informace o zařízení. Zásadní informací o zařízení je jeho Bluetooth adresa. Tato adresa je neměnná a pokaždé se toto zařízení bude objevovat pod stejnou adresou. Toho je využito právě při hledání, kdy se adresa nalezeného zařízení porovnává s předem zjištěnou adresou hledaného modulu. Pokud se tato adresa shoduje s požadovanou, je reference na objekt uložena pro další krok. Metoda *inquiryCompleted* je volána v okamžiku, kdy skončí prohledávaní. V této metodě se notifikuje synchronizační objekt o ukončení prohledávaní.

Druhá fáze, která musí být provedena před komunikací, spočívá v prohledání nabízených služeb zařízení. Pro komunikaci, která bude fungovat stejně jako sériové rozhraní, je potřeba najít službu SPP (Serial Port Profile). Ukázka (Zdrojový kód 20) zobrazuje implementaci druhé fáze. Pro nalezení služeb se použije metoda *searchServices*. Tato metoda přijímá jako první parametr pole atributů, která chceme zjistit. V případě implementace hledání SPP služby nám pouze stačí její název, který je reprezentován hexadecimální hodnotou 0x0100. Druhým

parametrem této funkce je hledaný typ služby. Služba SPP je reprezentována hexadecimální hodnotou 0x1101. Třetím parametrem funkce je reference na objekt zařízení, ve kterém chceme službu vyhledat. V případě této implementace je použita reference nalezeného Bluetooth modulu. Posledním parametrem funkce *searchServices* je opět reference na objekt implementující rozhraní *DiscoveryListener*.

```
1| UUID[] uuidSet = new UUID[1];
2| uuidSet[0] = new UUID(0x1101);
3|
4| int[] attrIDs = new int[]{
5|   0x0100
6| };
7|
8| agent.searchServices(attrIDs, uuidSet, carDevice, listener);
```

Zdrojový kód 20 - Implementace prohledávání služeb na Bluetooth zařízení

Po volání metody *searchServices* je opět potřeba provést synchronizaci vláken, protože metoda je asynchronní.

V případě metody *searchServices* budou v životním cyklu hledání služeb volány funkce *servicesDiscovered* a *servicesSearchComplete*. Implementace metody *servicesDiscovered* je znázorněna v ukázce (Zdrojový kód 21).

```
1 | @Override
2 | public void servicesDiscovered(
3 |         int i,
4 |         ServiceRecord[] serviceRecords) {
5 |   ServiceRecord record = serviceRecords[0];
6 |   carBluetoothDeviceUrl = record.getConnectionURL(
7 |     ServiceRecord.AUTHENTICATE_NOENCRYPT,
8 |     False
9 |   );
10| }
```

Zdrojový kód 21 - Implementace metody servicesDiscovered

Metoda *servicesDiscovered* je volána ve chvíli, kdy agent nalezne služby podle specifikace. Metoda přijímá dva parametry, první z nich je počet nalezených služeb a druhý je pole objektů obsahující reference na jednotlivé služby. V implementaci pro model vozidla je získaná služba pod prvním indexem v poli. Pro komunikaci s Bluetooth modulem v modelu stačí k otevření komunikačního proudu URL služby.

URL je možné získat pomocí metody *getConnectionURL*, která přijímá dva parametry. Prvním parametrem je požadované zabezpečení přenosu. V případě komunikace s Bluetooth module HC-05 je možné použít pouze zabezpečení na základě autentifikace. Při otevření komunikace je vyvolán systémový dialog, do kterého se zadá autentifikační kód. Potom je možné s modelem komunikovat. Šifrování modul HC-05 nepodporuje. Kdyby byl použit pro komunikaci modul s podporou šifrování, stačilo by použít konstantu *AUTHENTICATE_ENCRYPT* a knihovna BlueCove ve spolupráci se systémem zajistí potřebné kroky ke spuštění šifrované komunikace. Druhý parametr funkce *getConnectionURL* určuje, zda aplikace požaduje, aby se počítač k modulu nutně připojil jako *master*.

Implementace metody *servicesSearchComplete* je totožná s implementací *inquiryCompleted* – notifikuje se synchronizační objekt.

Po úspěšném získání adresy služby se pomocí metody *open* třídy *Connector* otevře spojení s Bluetooth modulem. Tato metoda navrací objekt implementující rozhraní *Connection*, ze kterého se získá odchozí stream, do kterého je možné zapisovat binární data.

Přímou komunikaci s modelem má na starost třída *CarController*. Ta v konstruktoru přijímá instanci třídy *OutputStream*, skrze kterou vysílá příkazy pro Arduino na modelu vozidla. Třída CarController byla převzata z vlastního projektu – aplikace pro ovládání modelu vozidla pomocí telefonu. Tato aplikace byla vytvořena pro telefony s operačním systémem Android a je přizpůsobena pro komunikaci s vytvořeným modelem vozidla. Jak již bylo zmíněno v kapitole zabývající se použitým hardwarem, tak software ve vývojové desce Arduino je navržen jako konečný automat. Tomu byla přizpůsobena i komunikace. Model vozidla se může nacházet v jednom z osmi stavů. Každý stav je reprezentován aktuálním nastavením směru otáčení motorů a jejich stavem zapnutí. Definice přechodu do jednotlivých stavů je obsažena v privátní funkci *setState*. Ovládání modelu probíhá prostřednictvím pěti veřejných funkcí – *forward*, *left*, *right*, *stop* a *reverse.* Tyto funkce obsahují definice možných přechodů mezi jednotlivými stavy. Přechody jsou optimalizovány tak, aby byl minimalizován přenos informací směrem od počítače k modulu. Rovněž byl při návrhu kladen důraz na správnou posloupnost příkazů, aby nedošlo k poškození elektroniky modelu vozidla.

5 Shrnutí výsledků

Výsledkem této práce je hotový návrh řešení, který propojuje více oblastí informatiky s elektronikou. Mimo samotný návrh vznikla znovupoužitelná platforma a sada nástrojů pro další rozvoj a výzkum autonomního řízení. Implementační část zahrnuje softwarové komponenty pro Raspberry Pi, které zajišťují přenos senzorických dat. Rovněž je implementována aplikace pro vytváření vzorových dat, která je propojena s ovládáním testovacího modelu vozidla. Součástí hotového softwaru je i nástroj pro učení neuronové sítě ze získaných vzorových dat a aplikace, která propojuje všechny navržené modely, pro autonomní řízení modelu vozidla.

Při práci na testovací implementaci se objevilo hned několik problémů, které bylo potřeba řešit. První problém se objevil při implementaci přenosu dat ze senzoru vzdálenosti. Zamýšlený MQTT broker, který by byl pro provoz na Raspberry Pi díky nižší paměťové náročnosti lepší, bohužel nepracoval s aktuální verzí protokolu. Proto byl zvolen aktuální, ale paměťově náročný, MQTT broker implementovaný v jazyce Java. Druhým zásadním problémem z hlediska implementace byla knihovna pro Bluetooth komunikaci, která v aktuální stabilní verzi nepodporuje 64-bitovou verzi jazyka Java na platformě macOS. Tento problém byl vyřešen nalezením vývojové verze použité knihovny, která již potřebnou architekturu podporuje.

6 Závěry a doporučení

Navrhované řešení by mělo sloužit jako základ pro další výzkum autonomního řízení modelů vozidel. Další rozvoj tohoto projektu by měl spočívat ve zprovoznění plnohodnotného testovacího modelu vozidla, který zajistí dostatečný prostor pro umístění všech hardwarových komponent včetně potřebného napájení. Nový model by měl být co nejvíce podobný klasickému vozidlu. Tím je myšleno hlavně to, aby se směr zatáčení ovládal mírou natočení předních kol. Současný model, vzhledem ke smykovému ovládání, není v tomto ohledu ideální.

Mimo samotný model je z hlediska hardwaru možné použití dokonalejších kamer a senzorů. V rámci dalšího rozvoje je možné vyzkoušet i malý LiDAR a prozkoumat jeho použitelnost v podobných projektech.

Vhodné by rovněž bylo vyzkoušet, zda by bylo možné provozovat všechny softwarové moduly na Raspberry Pi a tím dosáhnou nezávislosti modelu na bezdrátových sítí, které dnes nutně ke svému provozu potřebuje. Tyto optimalizace by měli spočívat hlavně v paměťové optimalizaci celého softwarového řešení.

Po softwarové stránce je možné rozvíjet projekt tak, aby se model dokázal chovat jako řidič (člověk) v běžném provozu. Další výzkum by se měl zaměřit na vývoj plánovacího modulu, který dokáže rozpoznávat vozovku a na základě vizuálních informací dokonale zvládne ovládání modelu vozidla. Následujícím krokem je rozpoznávání dopravního značení. K tomu by bylo vhodné vytvořit i model města, na kterém se model bude učit a aplikovat naučené chování.

Velmi zajímavým směrem dalšího rozvoje by byla i možnost zapojení více modelů, které by se pohybovali modelem města. K tomu bude však nutný rozvoj nejen plánovacího modulu, ale i nutné komunikace mezi jednotlivými modely.

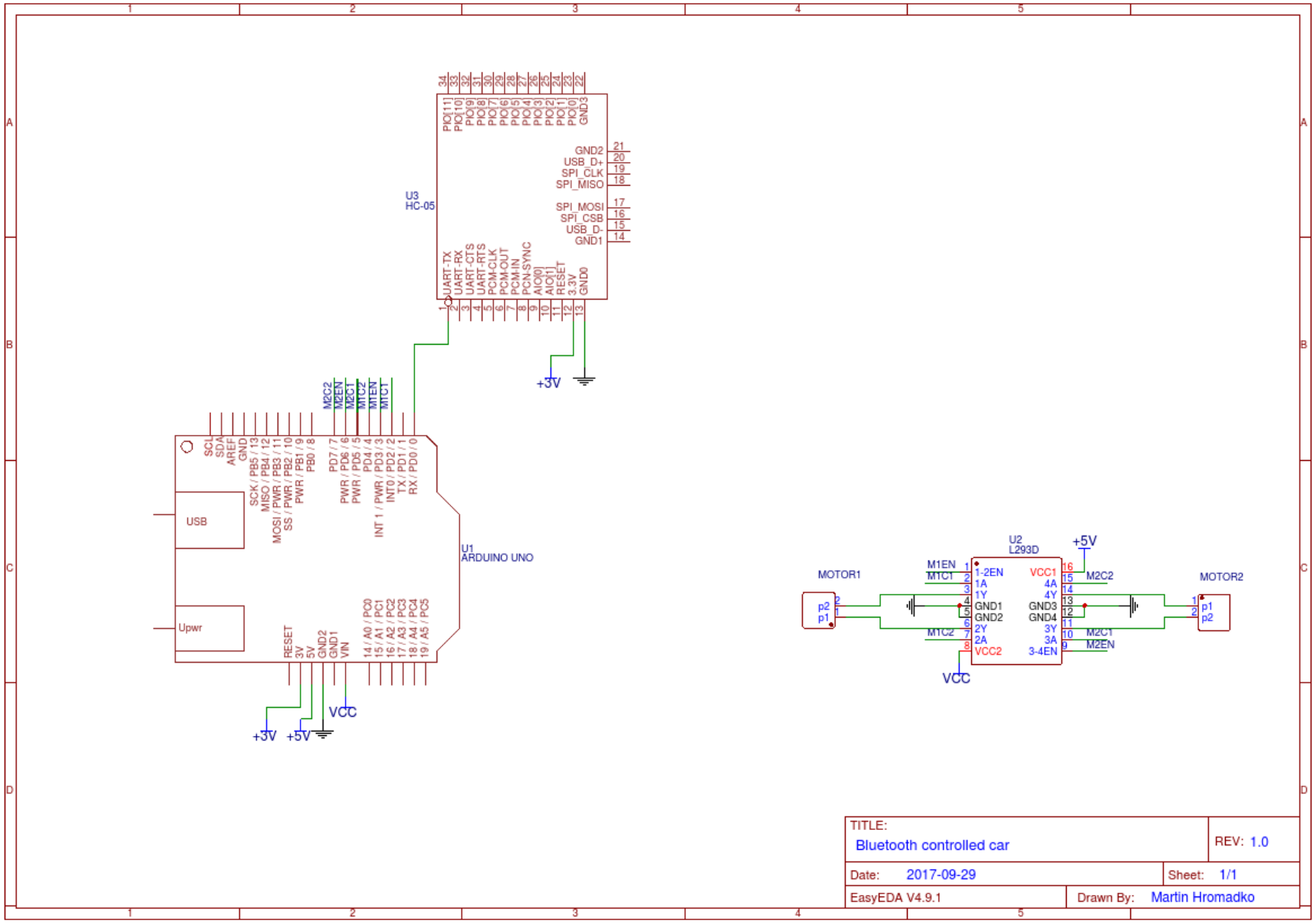
7 Seznam použité literatury

1. SMITH, B. W. Center for Internet and Society. In: *SAE Levels of Driving Automation* [online]. 18. 12. 2013 [cit. 2017-08-01]. Dostupné z: http://cyberlaw.stanford.edu/blog/2013/12/sae-levels-driving-automation
2. KICHUN, J. et al. Development of Autonomous Car—Part II: A Case Study on the Implementation of an Autonomous Driving …. IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, 2015, **62** (8).
3. SANTO, D. EETimes. In: *Autonomous Cars' Pick: Camera, Radar, Lidar?* [online]. 7. 7. 2016 [cit. 2017-06-05]. Dostupné z: http://www.eetimes.com/author.asp?section_id=36&doc_id=1330069
4. LIDAR-UK. LiDAR-UK. In: *How does LiDAR work?* [online]. 2017 [cit. 2017-06-28]. Dostupné z: http://www.lidar-uk.com/how-lidar-works/
5. NATIONAL INSTRUMETS. National Instrumets. In: *What is a Real-Time Operating System (RTOS)?* [online]. 22. 11. 2013 [cit. 2017-08-04]. Dostupné z: http://www.ni.com/white-paper/3938/en/
6. BUTTAZO, G. C. Real-time operating systems and standards. In: BUTTAZO, G. C. *Hard Real-Time Computing Systems*. Boston (MA): Springer, 2011. ISBN 978-1-4614-0676-1.
7. Raspberry Pi. *Raspberry Pi* [online]. [cit. 2017-07-20]. Dostupné z: https://www.raspberrypi.org/
8. HARRINGTON, W. *Learning Raspbian*. Birmingham: Packt Publishing Ltd. 2015. ISBN 978-1-78439-219-2.
9. BOŘÁNEK, R. Root.cz. In: *Raspberry Pi B+ přidává USB porty a snižuje spotřebu* [online]. 14. 7. 2014 [cit. 2017-07-26]. ISSN 1212-8309
10. KRČMÁŘ, P. Root. In: *Raspberry Pi A+ bude menší a nabídne micro SD slot a konektor pro displej* [online]. 10. 11. 2014 [cit. 2017-07-28]. ISSN 1212-8309
11. RASPBERRY PI FOUNDATION. Raspbian. In: *About Raspbian* [online]. [cit. 2017-07-21]. Dostupné z: https://www.raspbian.org/RaspbianAbout

12. MIRCROSOFT CORPORATION. Windows Dev Center. In: *Learn about Windows 10 IoT Core* [online]. [cit. 2017-07-24]. Dostupné z: https://developer.microsoft.com/en-us/windows/iot/Explore/IoTCore
13. OSMC. OSMC. In: *About* [online]. 2017 [cit. 2017-07-24]. Dostupné z: https://osmc.tv/about/
14. PINET TEAM. PiNet, a centralised user accounts and file storage system for a Raspberry Pi classroom. In: *Introduction* [online]. [cit. 2017-07-24]. Dostupné z: http://pinet.org.uk/articles/guides.html
15. LEE,. RISC OS OPEN. In: *Welcome to RISC OS Pi* [online]. 2011 [cit. 2017-10-01]. Dostupné z: https://www.riscosopen.org/wiki/documentation/show/Welcome%20to%20RISC%20OS%20Pi
16. UNIPI TECHNOLOGY. UniPi Technology. In: *Neuron* [online]. [cit. 2017-07-10]. Dostupné z: https://www.unipi.technology/cs/produkty/neuron-3?categoryId=2&categorySlug=unipi-neuron
17. Pi MusicBox [online]. [cit. 2017-08-12]. Dostupné z: http://www.pimusicbox.com/
18. WANG, Z. Zheng Wang. In: *Self Driving RC Car* [online]. 8. 8. 2015 [cit. 2017-08-19]. Dostupné z: https://zhengludwig.wordpress.com/projects/self-driving-rc-car/
19. BEŠTA, M. studijní materiály elektro. In: *Základy elektrotechniky* [online]. 2017 [cit. 2017-09-10]. Dostupné z: http://www.mbest.cz/wp-content/uploads/2013/10/T2.9-Dělič-napět%C3%AD.pdf
20. ARDUINO. Arduino. In: *Introduction* [online]. 2017 [cit. 2017-09-25]. Dostupné z: https://www.arduino.cc/en/Guide/Introduction
21. ARDUINO. Arduino Store. In: *ARDUINO UNO REV3* [online]. 2017 [cit. 2017-09-25]. Dostupné z: https://store.arduino.cc/usa/arduino-uno-rev3
22. DUNDÁČEK, M. *Mikroprocesorový modul řízení SS motoru se zpětnou vazbou* [Dokument]. Brno: 2008.

23. PELIKÁN, J. Pouzdra integrovaných obvodů. In: *Pouzdra integrovaných obvodů* [online]. [cit. 2017-09-29]. Dostupné z: https://www.fi.muni.cz/usr/pelikan/ARCHIT/TEXTY/POUZDRA.HTML
24. HIRZEL, T. Arduino. In: *PWM* [online]. 2017 [cit. 2017-10-03]. Dostupné z: https://www.arduino.cc/en/Tutorial/PWM
25. CURTIS, K. DPS. In: *Použití ultrazvukových měničů pro měření vzdálenosti* [online]. 2015 [cit. 2017-09-21]. Dostupné z: http://www.dps-az.cz/mereni/id:20858/pouziti-ultrazvukovych-menicu-pro-mereni-vzdalenosti
26. MATT. Raspberry Pi Spy. In: *Simple Guide to the RPi GPIO Header and Pins* [online]. 9. 6. 2012 [cit. 2017-09-21]. Dostupné z: https://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/
27. MQTT.ORG. MQTT. In: *Frequently Asked Questions* [online]. [cit. 2017-09-23]. Dostupné z: http://mqtt.org/faq
28. MALÝ, M. Root.cz. In: *Protokol MQTT: komunikační standard pro IoT* [online]. 29. 6. 2016 [cit. 2017-09-23]. Dostupné z: https://www.root.cz/clanky/protokol-mqtt-komunikacni-standard-pro-iot/
29. JACKSON, L. GitHub. In: *mjpg-streamer* [online]. 28. 8. 2017 [cit. 2017-09-13]. Dostupné z: https://github.com/jacksonliam/mjpg-streamer
30. FIRYN, B. GitHub. In: *Webcam Capture API* [online]. 30. 9. 2017 [cit. 2017-10-10]. Dostupné z: https://github.com/sarxos/webcam-capture
31. FUSESOURCE. GitHub. In: *A Java MQTT Client* [online]. 2015 [cit. 2017-10-13]. Dostupné z: https://github.com/fusesource/mqtt-client
32. YEUNG, D. S. et al. Introduction to Neural Networks. In: *Sensitivity Analysis for Neural Networks*. Berlin: Springer, 2010. ISBN 978-3-642-02532-7.
33. OPENCV TEAM. OpenCV. In: *About* [online]. 2017 [cit. 2017-10-09]. Dostupné z: http://opencv.org/about.html
34. BlueCove [online]. 2008 [cit. 2017-10-13]. Dostupné z: http://www.bluecove.org/index.html

8 Přílohy


U3
HC-05
U1
ARDUINO UNO
U2
L293D
MOTOR1
MOTOR2
+5V
+3V
VCC
TITLE:
Bluetooth controlled car
REV: 1.0
Date: 2017-09-29
Sheet: 1/1
EasyEDA V4.9.1
Drawn By: Martin Hromadko

Obsah CD

- *Arduino* –zdrojové kódy použité ve vývojové desce Arduino
- *Raspberry Pi* – zdrojový kód programu pro výpočet vzdálenosti a přenos dat přes MQTT protokol
- *Computer* – zdrojové kódy softwarových modulů a nástrojů

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2016/2017

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai2-p)

Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Bc. Hromádko Martin	Třída SNP 861/65, Hradec Králové - Slezské Předměstí	I1500687

TÉMA ČESKY:

Možnosti využití Raspberry Pi pro autonomní řízení modelu vozidla

TÉMA ANGLICKY:

Usage possibilities of Raspberry Pi for autonomous control of vehicle model

VEDOUCÍ PRÁCE:

doc. Ing. Filip Malý, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Analyzovat aktuální stav vývoje autonomních vozidel a používané technologie, prozkoumat možnosti využití Raspberry Pi jako řídící jednotky pro autonomní řízení modelu vozidla, navrhnout vhodné řešení a pokusit se toto řešení realizovat.

Osnova:

1. Úvod
2. Technologie používané pro autonomní řízení vozidel
3. Možnosti využití Raspberry Pi
4. Návrh řešení
5. Závěr
6. Literatura

SEZNAM DOPORUČENÉ LITERATURY:

Podpis studenta: ..... **Datum:** 12.10.2016

Podpis vedoucího práce: ..... **Datum:** 12.10.2016