# UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE

## FAKULTA PRÍRODNÝCH VIED


# V2V KOMUNIKÁCIA POUŽITÍM MINIPOČÍTAČA RASPBERRY PI

**DIPLOMOVÁ PRÁCA**


**Nitra 2020** **BC. DÁVID BOJNANSKÝ**

# UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE

## FAKULTA PRÍRODNÝCH VIED


# V2V KOMUNIKÁCIA POUŽITÍM MINIPOČÍTAČA RASPBERRY PI

## DIPLOMOVÁ PRÁCA

| | |
|---|---|
| Študijný odbor: | 9.2.9 Aplikovaná informatika |
| Študijný program: | Aplikovaná informatika |
| Školiace pracovisko: | Katedra informatiky |
| Školiteľ: | PaedDr. Peter Švec, PhD. |


Nitra 2020 Bc. Dávid Bojnanský


Univerzita Konštantína Filozofa v Nitre
Fakulta prírodných vied

# ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:** Bc. Dávid Bojnanský
**Študijný program:** aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
**Študijný odbor:** aplikovaná informatika
**Typ záverečnej práce:** Diplomová práca
**Jazyk záverečnej práce:** slovenský
**Sekundárny jazyk:** anglický

**Názov:** V2V komunikácia použitím minipočítača Raspberry Pi

**Anotácia:** Komunikácia V2V (vehicle-to-vehicle) umožňuje jednoduché, rýchle a spoľahlivé odovzdanie informácií medzi vozidlami s cieľom zvýšiť bezpečnosť jazdy. Počítač Raspberry je možné pripojiť buď priamo k OBD-II rozhraniu prostredníctvom ELM327 Bluetooth adaptéru alebo ELM327 USB káblu. Vzájomnú komunikáciu RPi je možné realizovať v ad-hoc wifi sieti. Základné informácie je možné nájsť na http://www.instructables.com/id/OBD-Pi/

Ciele práce:
Cieľom práce je navrhnúť a zrealizovať distribuovaný systém V2V komunikácie pomocou minipočítača Raspberry Pi. Komunikácia medzi počítačom a vozidlom je realizovaná rozhraním OBD-II (knižnica pyOBD), vzájomná komunikácia minipočítačov je realizovaná v ad-hoc wifi sieti. Odporúčaným programovacím jazykom je Python.

Požiadavky na vedomosti a zručnosti:
Schopnosť programovať v jazyku C, Java alebo Python, základné znalosti z elektroniky, možnosť zabezpečiť si doplnky k počítaču Raspberry Pi

**Kľúčové slová:** V2V, Raspberry Pi, OBD-II

**Školiteľ:** PaedDr. Peter Švec, Ph.D.
**Oponent:** doc. Ing. Štefan Koprda, PhD.
**Katedra:** KI - Katedra informatiky

**Dátum zadania:** 30.09.2016

**Dátum schválenia:** 10.10.2016

prof. Ing. Milan Turčáni, CSc., v. r.
vedúci/a katedry

## POĎAKOVANIE

Touto cestou vyslovujem poďakovanie pánovi PaedDr. Petrovi Švecovi, PhD. za odporúčania, rady a pripomienky počas písania tejto práce, ako aj nasmerovania a uvedenia do problematiky na začiatku.

Poďakovať sa chcem aj mojej priateľke Števke, ktorá ma dokopala k tomu, aby som túto prácu vôbec začal písať, ako aj za jej neúnavnú podporu v priebehu toho nekonečného písania.

# ABSTRAKT

BOJNANSKÝ, Dávid: V2V komunikácia použitím minipočítača Raspberry Pi. [Diplomová práca]. Univerzita Konštantína Filozofa v Nitre. Fakulta prírodných vied. Školiteľ: PaedDr. Peter Švec, PhD. Stupeň odbornej kvalifikácie: Magister odboru Aplikovaná informatika. Nitra: FPV, 2020. 84 s.

Diplomová práca sa venuje problematike komunikácie (to jest výmene informácií) medzi viacerými vozidlami (takzvaný V2V systém), vytvoreniu takéhoto systému pomocou minipočítača Raspberry Pi, programovacieho jazyka Python a rozhrania OBD-II, otestovaniu vytvoreného systému v praxi a vyhodnoteniu dosiahnutých výsledkov. Diplomová práca je rozdelená na teoretickú a praktickú časť. Teoretická časť je zameraná na analýzu súčasného stavu a je rozdelená na štyri ďalšie časti. Prvá časť sa zaoberá V2V systémom v jeho rannom štádiu, kdežto druhá časť sa zaoberá V2V systémom v súčasnosti – v nej autor porovnáva minulosť a súčanosť, rozoberá dostupné bezdrôtové technológie vhodné pre V2V systém, rozoberá bezpečnosť tohto systému a prekážky brániace zavedeniu takéhoto systému do praxe. Tretia časť poskytuje stručnú charakteristiku minipočítačov Raspberry Pi a štvrtá časť zase rozhrania OBD-II. Praktická časť je rozdelená na tri ďalšie časti. Prvá časť je zameraná na vytvorenie návrhu V2V systému po hardvérovej a softvérovej stránke. Druhá časť sa zaoberá realizáciou tohto návrhu krok za krokom. Tretia časť opisuje testovanie v praxi a vyhodnocuje dosiahnuté výsledky. V závere práce autor navrhuje možné vylepšenia vytvoreného V2V systému.

Kľúčové slová: V2V. Raspberry Pi. OBD-II.

# ABSTRACT

BOJNANSKÝ, Dávid: V2V communication using minicomputer Raspberry Pi. [Diploma Thesis]. Constantine the Philosopher University in Nitra. Faculty of Natural Sciences. Supervisor: PaedDr. Peter Švec, PhD. Degree of Qualification: Magister of Applied Informatics. Nitra: FNS, 2020. 84 p.

Diploma thesis describes the issue of communication (that is information exchange) between several vehicles (so-called V2V system) and creating such a system using Rasperry Pi minicomputer, Python programming language and OBD-II interface. It also deals with testing the system in practise and evaluating the results. The thesis is divided into theoretical and practical part. Theoretical part is about analysis of the current state of the problem and is divided into four other parts. The first part deals with the V2V system in its early stage while the second part describes the V2V system at the modern times where the author compares the past with the present, discusses the available wireless technologies suitable for the V2V system, analyse the security of this system and obstacles to its implementation. The third part provides a brief description of Rasberry Pi minicomputers and the fourth part is about the OBD-II interface. The practical part of the thesis is divided into three other parts. The first part is focused on creating the V2V system design according to hardware and software side. The second part describes the step by step implementation of this proposal. The third part is about testing in practise and evaluates the achieved results. In conclusion of the thesis the author suggests possible improvements to the created V2V system.

Keywords: V2V. Raspberry Pi. OBD-II.

# OBSAH

# ZOZNAM ILUSTRÁCIÍ

# ZOZNAM ZDROJOVÝCH KÓDOV

# ÚVOD

V dnešnej dobe začína byť pre ľudí hlavnou prioritou ich bezpečnosť a začali si to uvedomovať aj výrobcovia vozidiel. Vozidlá disponujú airbagmi, systémom ABS, ktoré kontroluje prudké brzdenie na povrchoch s rôznou priľnavosťou, rôznymi asistenčnými systémami, ktoré vedia pomocou sady senzorov predvídať rôzne situácie a zlepšujú tak pôžitok z jazdy a podobne. Výrobcovia vozidiel chcú ísť v dohľadnej budúcnosti ešte ďalej. Chcú umožňiť vozidlám, aby sa medzi sebou „rozprávali." Nemyslíme tým, že teraz každé vozidlo bude mať v jeho prednej časti ústa a bude vydávať zvuky. Myslíme tým, neviditeľnú a nepočuteľnú bezdrôtovú komunikáciu. Komunikácia v tomto prípade predstavuje výmenu užitočných informácií ako napríklad rýchlosť vozidla pred nami, či vozidlo pred nami brzdí a podobne. Takáto komunikácia sa začala označovať ako systém V2V. Je to naozaj skvelá myšlienka, už len pri predstave, že sa neskutočne zredukuje množstvo dopravných nehôd a zachráni sa tak množstvo nie len ľudských životov.

My sme si vybrali tému diplomovej práce, ktorá s takouto komunikáciou medzi vozidlami súvisí. Jej hlavný cieľ bude vytvoriť funkčný V2V systém, otestovať ho v praxi a vyhodnotiť dosiahnuté výsledky. Diplomová práca bude rozdelená do troch kapitol.

V prvej kapitole sa budeme zaoberať analýzou súčasného stavu V2V systému, kde sa bližšie pozrieme aj do minulosti, porovnáme minulosť a súčasnosť, rozoberieme dostupné bezdrôtové technológie vhodné pre V2V systém, rozoberieme bezpečnosť tohto systému a prekážky brániace zavedeniu takéhoto systému do praxe. Ďalej sa pozrieme aj na stručnú charakteristiku minipočítačov Raspberry Pi a rozhrania OBD-II, ktoré budú neoddeliteľnou súčasťou nami vytvoreného V2V systému.

V druhej kapitole si definujeme ciele, ktoré sa budeme snažiť v rámci tretej kapitoly splniť.

V tretej kapitole sa budeme zaoberať praktickou časťou, kde sa budeme snažiť vytvoriť návrh nášho V2V systému po hardvérovej a softvérovej stránke, zrealizovať tento návrh krok za krokom, otestovať ho v praxi a nakoniec vyhodnotiť dosiahnuté výsledky.

# 1 ANALÝZA SÚČASNÉHO STAVU V2V SYSTÉMU

Prvé náznaky a ambície na realizáciu V2V systému boli už okolo roku 2007. Označenie V2V predstavuje skratku pre anglický výraz Vehicle-to-Vehicle, teda po našom Vozidlo-Vozidlo. Keď k tejto skratke pridáme slovo systém, tak spolu to predstavuje najmä systém, v ktorom komunikujú viaceré vozidlá medzi sebou.

V rámci tejto kapitoly (1) sme v prvej podkapitole (1.1) rozobrali pohľad na V2V systém v jeho začiatkoch – čo viedlo k jeho vytvoreniu, aká bola predstava jeho fungovania, z akých technológií mal byť poskladaný. Pozreli sme sa aj na skúšku vtedajšieho prototypu v praxi (oddiel 1.1.1). V závere sme uviedli vtedajší plán očakávaného zavedenia tohto systému do praxe (oddiel 1.1.2). V druhej podkapitole (1.2) sa venujeme jeho súčasnému stavu. V tretej podkapitole (1.3) rozoberáme minipočítač Raspberry Pi, na ktorom je založená naša diplomová práca. V štvrtej podkapitole (1.4) sa zaoberáme rozhraním OBD (II), ktorý je taktiež dôležitou súčasťou našej diplomovej práce.

## 1.1 PRVÉ NÁZNAKY UŽ V ROKU 2007

Podľa článku „Komunikácia medzi autami - vyskúšali sme technológiu budúcnosti“ publikovaného v roku 2007 autorom (Karpat, 2007), dôvodom na vôbec začatie rozmýšľania o komunikácii medzi vozidlami bola nemecká štatistika o nehodovosti na cestách z roku 2003. Tá hovorila jasnými číslami, a teda, že nehodovosť má rastúcu tendenciu. Autor v článku uviedol: „26,1 % havárií bolo spôsobených jazdou neprimeranou rýchlosťou a nedodržiavaním bezpečných odstupov medzi vozidlami, 22,8 % príčin tvorili prekážky na ceste, v zákrute, kľučkovania medzi autami a otáčania a 11,1 % pripadlo na jazdu v zlom pruhu a nesprávne predbiehanie.“ Autor článku ďalej uviedol, že jednou z možností ako znížiť počet nehôd a úmrtí na cestách je nehodám predchádzať.

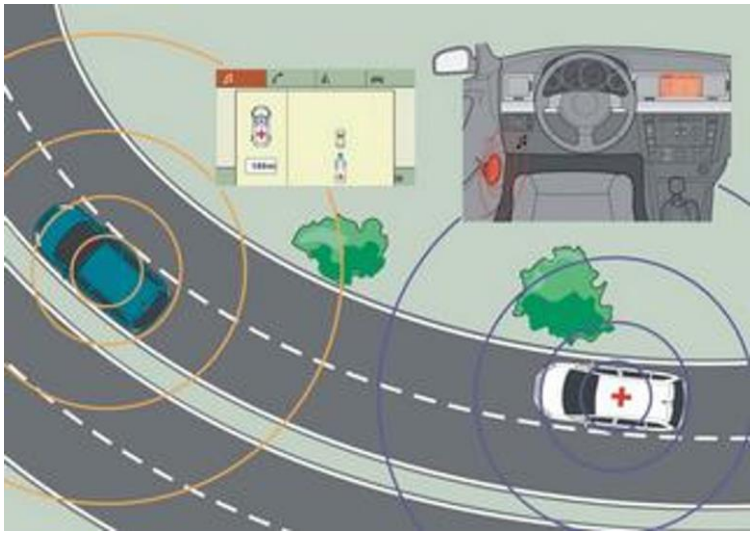
Ďalšou možnosťou by bolo zníženie počtu vozidiel. Keď sa však zamyslíme nad dnešnou dobou, tak klesanie, dokonca ani stagnácia tohto počtu sa nejaví ako reálna. V dnešnej dobe nie je nič nezvyčajné, ak napríklad štvorčlenná rodina vlastní aj päť áut. Z toho nám vyplýva, že počet vozidiel na cestách bude stále len narastať. S tým však súvisia aj ďalšie problémy ako napríklad nadmerná produkcia $CO_2$ alebo kolaps dopravy v každom väčšom meste. To je však nad rámec tejto témy.

V roku 2007 predstavila automobilka General Motors telematický systém prenosu informácií medzi vozidlami. V tých časoch bolo cieľom systému V2V včas upozorniť vodiča vozidla na nepredvídané nebezpečenstvo a v predstihu sa naň správne pripraviť a zareagovať. Medzi takéto nebezpečenstvá sa zaraďovalo riskantné konanie vodičov (odstavenie vozidla v zákrute, otáčanie sa za zákrutou), vozidlá zoraďujúce sa za sebou pri spájaní viacerých pruhov do jedného, práce na ceste alebo zlé poveternostné podmienky (námraza, sneh, hmla). Sústava systému V2V mala pozostávať z mikroprocesora, GPS prijímača a modulov realizujúcich bezdrôtovú komunikáciu. Táto sústava mala dokázať sledovať polohu vozidla, jeho rýchlosť a smer. Tieto údaje si potom mala vymieňať s okoloidúcimi vozidlami do vzdialenosti 2 až 5 kilometrov. Ak by systém zistil neštandardnú dopravnú situáciu, mal odoslať informácie cez bezdrôtovú WiFi sieť. Ostatné vozidlá v dosahu mali tieto informácie prijať a akusticky (cez reproduktory vo vozidle) aj opticky (cez displej vo vozidle) na ne vodiča upozorniť. Nie všetky informácie zachytené vozidlom, mali byť aj vodičovi interpretované. Dôvodom malo byť, že nie všetky informácie sú v danom čase pre vodiča dôležité. Systém mal upozorniť len na nebezpečné situácie na trase vozidla. Výhodou použitia bezdrôtovej WiFi siete malo byť to, že táto technológia už existovala a nevyžadovala si žiadne náklady na prevádzku, čo platí dodnes. V roku 2009 mala byť Európskou úniou na účel komunikácie medzi vozidlami schválená jedna spoločná frekvencia (Karpat, 2007).

### 1.1.1 Skúšanie v praxi

Autor (Karpat, 2007) v spomínanom článku vysvetľuje, ako to v tom čase malo fungovať v praxi. Vysvetlil to na príklade hmly: „Ak vodič vojde do hmly, prvé čo urobí, zníži rýchlosť a zapne hmlové svetlá. Keď to zopakuje desať áut po sebe, dá sa táto informácia považovať za dôveryhodnú v porovnaní s tým, keď mladí zapnú hmlovky bezdôvodne len tak pre frajerinu.“ Inžinieri automobilky General Motors im umožnili si celý systém názorne vyskúšať. K dispozícii mali štyri vozidlá.

Prvá ukážka spočívala v tom, ako vie systém V2V upozorniť na blížiace sa policajné alebo záchranné vozidlo. Problém, ktorý nastal a platí dodnes je, že sirénu síce počujeme, ale nemusíme vždy tušiť odkiaľ dané vozidlo prichádza, čo môže spôsobiť vodičovi stres. Systém ukázal, z ktorej strany sa dané vozidlo blíži a ako je od nich ďaleko. Myšlienku znázorňuje obrázok 1.



*Obrázok 1 Blížiace sa záchranné vozidlo signalizované vo V2V systéme*

Druhá ukážka bolo stojace vozidlo v neprehľadnej zákrute alebo dopravné obmedzenie na neprehľadnom mieste. Aj keď prekážka nebola v ich jazdnom pruhu a nehrozila kolízia, tak systém V2V aj tak včas upozornil, akusticky aj vizuálne, že na ceste je prítomná prekážka.

Ďalšia ukážka spočívala v tom, ako systém funguje pri vznikajúcej zápche na diaľnici. Systém na pomaly idúce alebo stojace vozidlo pred nimi upozrnil najprv akusticky a vizuálne na displeji. Ak vodič nezareagoval na žiadne upozornenie, napríklad kvôli telefonovaniu, tak ich vozidlo poslalo informáciu vozidlu tomu pred ním, že sa blíži a to zaplo brzdové svetlá.

Autor v článku ďalej vysvetľuje, že okrem upozorňovania mal systém V2V vedieť zasiahnuť aj v reálnom čase. Popísal to na príklade, kde blížiace vozidlo prichádzajúce do križovatky z bočnej cesty si Vás nevšíma a nespomaľuje, tak naše vozidlo to zaregistruje a ešte pred vjazdom do križovatky prudko zabrzdí. Vďaka vzájomnej komunikácii medzi sebou by však mohli obe vozidlá zastať súčasne.

### 1.1.2 Uvedenie do praxe

Predstavitelia automobilky General Motors chceli tento systém uviesť do praxe v priebehu 5 až 10 rokov. Aby tento systém fungoval v praxi, tak predpokladali, že minimálne 15 % jazdiacich vozidiel by muselo byť vybavených týmto systémom.

Jedným zo základných predpokladov úspechu tohto systému bolo aj to, že by mal byť dostupný aj v lacnejších verziách vozidiel. Viceprezident pre vývoj automobilky General Motors v EÚ v tom čase povedal: „Čím viac vozidiel bude vybavených týmto systémom, tým efektívnejšia bude jeho účinnosť. Preto je dôležité, aby systém V2V bol cenovo dostupný (Karpat, 2007).“

## 1.2 V2V SYSTÉM V SÚČASNOSTI

„Automobilový priemysel kladie na dnešné bezdrôtové komunikačné systémy súbor nových požiadaviek. Komunikačné aplikácie vozidiel zabezpečujúce bezpečnosť nemôžu tolerovať dlho trvajúce nadväzovanie spojenia predtým ako sa vozidlá stretnú na ceste a začnú medzi sebou komunikovať. Podobne aj komunikačné aplikácie, ktoré nesúvisia s bezpečnosťou, taktiež závisia na efektívnom nadviazaní spojenia so zariadeniami poskytujúcimi služby, ako napríklad aktualizácia mapy, pretože čas, počas ktorého prejde vozidlo cez oblasť pokrytú signálom, je limitovaný. Navyše, rýchlo sa pohybujúce vozidlá a zložité okolie cesty predstavujú výzvy na úrovni fyzickej vrstvy OSI modelu,“ uvádzajú autori (Jiang, a iní, 2008) v konferenčnom príspevku „IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments.“

National Highway Traffic Safety Administration (NHTSA) je americký úrad zodpovedený za bezpečnosť ľudí na amerických cestách. Prostredníctvom presádzania štandardov a partnerstiev so štátom a miestnymi vládnymi organizáciami, redukuje úmrtia, zranenia a ekonomické straty pri nehodách motorových vozidiel (nhtsa.gov, 2019).

Na stránke (nhtsa.gov, 2019) sa ďalej uvádza, že NHTSA spolupracuje s automobilovým priemyslom a akademickými inštitúciami už vyše dekádu, aby sa V2V systém stal realitou kvôli jeho potenciálu zachraňovať životy.

Od prvých náznakov a ambícií z roku 2007 na komunikáciu vozidiel medzi sebou ubehlo už viac ako desaťročie. V rámci tejto podkapitoly (1.2) sme v prvom oddieli (1.2.1) stručne porovnali rozdiely súčasnosti a v krátkosti rozobratej minulosti v podkapitole 1.1, a síce čo pribudlo, od čoho sa upustilo, čo sa zmenilo, vylepšilo alebo jednoducho ostalo rovnaké. V druhom oddieli (1.2.2) sme sa zamerali na bezdrôtové technológie, pomocou ktorých sa dnes dá, pri niektorých aspoň teoreticky, V2V systém realizovať. V treťom oddieli (1.2.3) sa zameriavame na samotnú bezpečnosť V2V

systému. V poslednom oddieli (1.2.4) sa venujeme prekážkam, ktoré v súčasnosti bránia zavedeniu V2V systému do praxe.

### 1.2.1 Porovnanie rozdielov minulosť/súčanosť

Oblasť systému V2V sa rozšírila do viacero smerov. V článku „Talking Cars: A Survey of Protocols for Connected Vehicle Communication“ autorka (Jain, 2018) vysvetľuje podobné systémy ako V2V systém: „Vozidlá sa nepochybne stávajú chytrejšími a technológia vo vozidle sa zlepšuje. Jedným z aspektov, ktorým sa venuje veľká pozornosť, je schopnosť inteligentného vozidla komunikovať s cestujúcimi, ako aj s jeho okolím.“ Tieto systémy môžu byť kategorizované nasledovne:

- V2E (Vehicle-to-Environment) – Vozidlo-Prostredie – predstavuje komunikáciu medzi vozidlom a dynamicky sa meniacim prostredím (V2V, V2P) ako aj komunikáciu medzi vozidlom a statickým prostredím (V2I, V2G). Patrí sem:
  - V2V,
  - V2I (Vehicle-to-Infrastructure) – Vozidlo-Infraštruktúra – semafory, mýtne brány, cesty, a iné,
  - V2P (Vehicle-to-Pedestrian) – Vozidlo-Chodec – chodci, cyklisti, a iné,
  - V2G (Vehicle-to-Grid) – Vozidlo-Sieť – elektrická sieť,
- V2U (Vehicle-to-User) – Vozidlo-Používateľ – predstavuje komunikáciu medzi vozidlom a jeho posádkou vrátane vodiča. Jedná sa o prispôsobenie vozidla a spríjemnenie tak cestovania,
- V2N (Vehicle-to-Network) – Vozidlo-Sieť – predstavuje komunikáciu vozidla a sieťového operátora, ktorý poskytuje užitočné informácie ako dočasné uzavretie cesty alebo tvoriaca sa kolóna na našej trase. Môže predstavovať aj prístup k internetovej sieti,
- V2D (Vehicle-to-Device) – Vozidlo-Zariadenie – predstavuje komunikáciu vozidla s elektronickým zariadením pripojeným k vozidlu (telefóny a iné),
- V2X (Vehicle-to-Everything) – Vozidlo-Všetko – predstavuje všetky vyššie spomenuté systémy v jednom.

Koncept bezdrôtovej siete sa samozrejme zachoval. Tak isto, hlavné informácie, ktoré si vozidlá bezdrôtovo vymieňajú, sú rýchlosť, GPS lokácia a smer jazdy. V minulosti sa hovorilo o dosahu 2 až 5 kilometrov. V súčasnosti sa však hovorí

približne o dosahu viac ako 300 metrov (nhtsa.gov, 2019). Niektoré zdroje uvádzajú 1 kilometer, no všetko je závislé od použitej technológie.

Podľa článku „Will Vehicle-to-Vehicle Communication Ever Take Off?“ od autora (Koon, 2019) sú v kurze dve konkurenčné a vzájomne sa vylučujúce technológie na implementáciu V2V systému. Prvou z nich je Dedicated Short-Range Communication (DSRC) založenej na známom protokole WiFi siete a druhou je 5G celulárna sieť, ktorú autor opisuje ako rýchlejšiu, s nižšou odozvou a väčším dosahom. Najväčšou prekážkou je, že obe tieto technológie používajú rovnaké frekvenčné pásma a nevedia medzi sebou spolupracovať. Výhodou technológie DSRC je, že nie je závislá na žiadnej externej celulárnej sieti. Autor uvádza aj to, že podľa priemyselného analytika sa automobilový priemysel prikloní skôr k 5G sieti kvôli jej lepším vlastnostiam, hoci niektoré automobilky sú naopak za DSRC technológiu, aby sa vyhli poplatkom za jej používanie. NHTSA explicitne nepresadzuje ani jedno, ani druhé, avšak častejšie sa odkazuje na technológiu DSRC.

Tvrdenie, že sa na prenos informácií použije technológia bezdrôtovej WiFi siete sa stalo čiastočne pravdivé. Ako sme spomenuli vyššie, v hre je aj ďalšia technológia. V roku 2010 bol schválený štandard IEEE 802.11p, ktorý predstavuje vylepšenú bezdrôtovú WiFi sieť oproti jej predchodcom, pridáva bezdrôtový prístup v dopravnom prostredí (Wireless Access in Vehicular Environment – WAVE) vyžadovaný aplikáciami ITS (Intelligent Transportation Systems), a ktorý je základom pre DSRC (standards.ieee.org, 2010). Bližšie tento štandard rozoberieme v podkapitole (1.2.2.4).

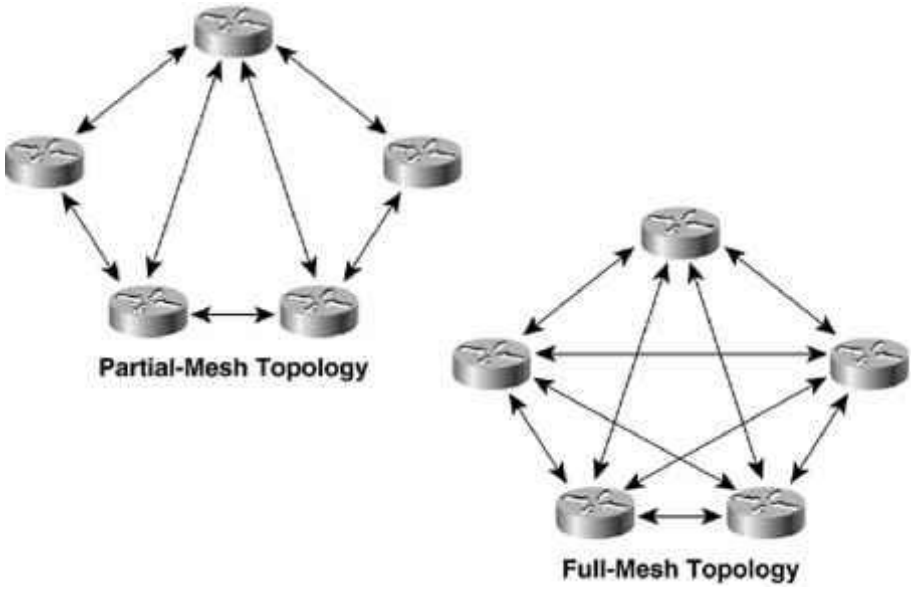
Koncept upozornenia vodiča V2V systémom na nebezpečenstvo ostal. Okrem toho vie tento systém aj sám zareagovať. Dnes sa však začalo hlasnejšie hovoriť aj o plne autonómnych vozidlách, čo je vyšší level V2V systému. Pri týchto vozidlách si len zadáme cieľ, kam chceme ísť. Pre zaujímavosť, ako prvá spoločnosť takéto auto začala predávať spoločnosť Tesla v roku 2014, konkrétne vozidlo Tesla Model S. Systém sa volá Autopilot a dnes už je reálne inštalovaný do všetkých vozidiel Tesla. Prvenstvo si však drží spoločnosť Google, ktorá ako vôbec prvá začala na takomto vozidle pracovať, a to v roku 2009 (vozidlo Waymo) (Dormehl, a iní, 2019).

Automobilka General Motors mala v pláne zaviesť technológiu V2V systému v najbližších 5 až 10 rokoch v tej dobe. Dnes však vieme, že sa to realitou nestalo. Pre viaceré automobilky sa v súčasnosti ako zlomový rok javí 2023 (Hall-Geisler, 2017).

Pri analýze týchto rozdielov sme dospeli k tomu, že za toto desaťročie sa V2V systém posunul určite vpred, no vyskytli sa nové problémy a výzvy, pričom veľa z nich nie je jednoduché vyriešiť.

### 1.2.2 Bezdrôtové technológie

V článku „An Overview of Vehicular Communications“ autori (Arena, a iní, 2019) uvádzajú, že V2V systém používa na prenos informácií medzi vozidlami bezdrôtovú Ad-Hoc Mesh sieť. Jedná sa o decentralizovaný systém spojenia medzi zariadeniami. Mesh topológia môže byť plne prepojená medzi uzlami alebo len čiastočne. Na obrázku 2 môžeme vidieť rozdiel medzi týmito topológiami.




*Obrázok 2 Čiastočne a plne prepojená Mesh sieťová topológia*

Pri čiastočne prepojenej Mesh topológii sú len niektoré uzly priamo pospájané medzi sebou, a teda môžu medzi sebou priamo komunikovať bez sprostredkovateľa (single-hop). Ak priame spojenie neexistuje, komunikácia môže ísť prostredníctvom uzla, s ktorým vedia komunikovať oba koncové uzly (multi-hop). Spojenia medzi uzlami môžu vznikať aj zanikať dynamicky podľa toho, ako často a v akom objeme si dáta vymieňajú. Pri plne prepojenej Mesh topológii sú priamo pospájané všetky uzly (Arena, a iní, 2019).

#### 1.2.2.1 Bluetooth

V dnešnej dobe existuje viacero technológií a protokolov na bezdrôtovú komunikáciu medzi zariadeniami vo všeobecnosti. Bluetooth je jednou z týchto technológií. Slúži na výmenu informácií medzi zariadeniami na krátku vzdialenosť. Slúži na vytvorenie tzv. Personal Area Network (PAN), čiže na osobnú sieť dostupnú na malom rozsahu v okolí. Pracuje na UHF (Ultra High Frequency) rádiovej frekvencii v rozsahu od 2.402 do 2.480 GHz. Bluetooth má proprietárny otvorený štandard označený IEEE 802.15.1, avšak už nie je udržiavaný. Vývoj zastrešuje zoskupenie Bluetooth SIG (Special Interest Group), ktoré má 35 000 členských organizácií. (Singh, a iní, 2015).

Bluetooth bol predstavený v roku 1994 ako náhrada pre RS-232 káble. Využiť sa dá napríklad na prehrávanie hudby v bezdrôtových slúchadlách púšťanej cez telefón, prenos súborov medzi dvoma telefónmi a podobne (techopedia.com, 2019).

V dobe písania tejto práce je najnovšou verziou Bluetooth 5.1. Dosah signálu je až do 200 metrov (niekde uvádzajú aj 400, respektíve 600 metrov) v exteriéri a 50 metrov v interiéri. Rýchlosť prenosu dát je 2 Mb/s. Je spätne kompatibilný s predchádzajúcimi verziami. Veľkosť paketu je 255 bajtov. Dokáže eliminovať rušivé vplyvy a tým pádom aj znížiť počet neočakávaných odpojení (10najs.sk, 2018).

*Obrázok 3 Anglický názov Bluetooth pri doslovnom preklade predstavuje modrý zub*

#### 1.2.2.2 ZigBee

Ďalšiou bezdrôtovou technológiou je ZigBee. Stránka (techopedia.com, 2019) uvádza, že ZigBee má otvorený štandard označený IEEE 802.15.4, a teda môže byť implementovaný hocikým. Používa nízko energetické rádiové signály na vytváranie PAN sietí, ktoré pracujú s nižšou rýchlosťou prenosu dát, sú energeticky efektívne a bezpečné. Využitie si táto technológia našla v inteligentných domácnostiach, automatizovaných systémoch, medicínskych zariadeniach, a podobne. Na rozdiel od Bluetooth-u je navrhnutý jednoduchším spôsobom. Funguje na princípe Mesh sieťovej topológii. Poskytuje vysokú spoľahlivosť a rozumné rozpätie dostupnosti v okolí.

Komunikácia je šifrovaná 128 bitovými kryptografickými kľúčmi. Obe strany používajú na šifrovanie a dešifrovanie rovnaké kľúče.

Stránka (electronics-notes.com, 2019) uvádza, že technológia ZigBee môže byť spoľahlivo použitá v prostredí, kde existuje viacero rádiových signálov. Vyvíjaný a udržiavaný je alianciou Zigbee Alliance, ktorá bola založená v roku 2002. Prenos údajov spoľahlivo funguje na vzdialenosť 70 metrov medzi dvoma uzlami, avšak pokiaľ sa komunikácia realizuje prostredníctvom viacerých medziuzlov, tak vzdialenosť môže byť oveľa väčšia. Technológia vysiela na frekvenčných pásmach 2.4 GHz, respektíve 915 MHz pre severnú Ameriku a 868 MHz pre Európu. Na frekvencii 2.4 GHz má k dispozícii celkovo 16 kanálov s maximálnou rýchlosťou prenosu 250 Kb/s. Frekvencia 915 MHz má k dispozícii celkovo 10 kanálov s rýchlosťou prenosu 40 Kb/s a frekvencia 868 MHz má k dispozícii 1 kanál s rýchlosťou prenosu 20 Kb/s. Dáta sú prenášané v paketoch s maximálnou veľkosťou 128 bajtov, pričom samotný obsah paketu (takzvaný payload) môže mať maximálne 104 bajtov. Technológia podporuje až 64 bitové adresy. V dobe písania tejto práce je najnovšou verziou ZigBee 3.0.

Autor (Blank, 2016) uvádza, že technológia ZigBee umožňuje neobmedzené množstvo hop-ov medzi uzlami a podporuje až 65 536 zariadení v jednej sieti.

#### 1.2.2.3 Wireless Local Area Network

Ďalšiou bezdrôtovou technológiou je Wireless Local Area Network (WLAN), známejšia však ako WiFi sieť. WLAN sieť je štandardizovaná v skupine s označením IEEE 802.11.

„802.11 sa vzťahuje na skupinu špecifikácií vyvinutých asociáciou IEEE pre WLAN technológiu. 802.11 špecifikuje vzdušné rozhranie medzi bezdrôtovým klientom a základňou (napríklad router) alebo dvoma bezdrôtovými klientami. Asociácia IEEE schválila špecifikáciu 802.11 v roku 1997," uvádza autor (Beal, 2019) vo svojom článku „802.11 IEEE wireless LAN standards."

Autor (Beal, 2019) ďalej uvádza, že v skupine 802.11 nájdeme tieto špecikácie:

- 802.11 – poskytuje prenosovú rýchlosť 1 až 2 Mb/s na frekvencii 2.4 GHz podľa toho, či používa metódu vysielania rádiového signálu Frequency Hopping Spread Spectrum (FHSS) alebo Direct Sequence Spread Spectrum (DSSS),
- 802.11a – vylepšená 802.11, ktorá poskytuje prenosovú rýchlosť až do 54 Mb/s na frekvencii 5 GHz. Namiesto FHSS a DSSS používa metódu digitálneho

kódovania dát na viacerých nosných frekvenciách Orthogonal Frequency-Division Multiplexing (OFDM),

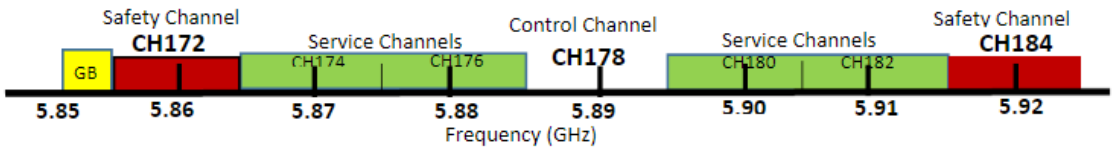
- 802.11b – označovaná tiež ako 802.11 High Rate alebo WiFi. Založená je na 802.11 a poskytuje prenosovú rýchlosť 11 Mb/s na frekvencii 2.4 GHz. Používa iba metódu DSSS. Poskytuje funkcionalitu porovnateľnú s Ethernet protokolom,
- 802.11e – definuje koncept Quality of Service (QoS) a vylepšuje 802.11a a 802.11b. Pridáva podporu pre multimédiá a poskytuje spätnú kompatibilitu s uvedenými špecifikáciami,
- 802.11g – používa sa na prenos na krátke vzdialenosti s rýchlosťou do 54 Mb/s na frekvencii 2.4 GHz,
- 802.11n – stavia na predchádzajúcich špecifikáciach a pridáva princíp Multiple Input/Multiple Output (MIMO). Pridaný vysielač a prijímač umožňuje vyššiu dátovú priepustnosť prostredníctvom nezávislého a oddeleného vysielania zakódovaných dátových signálov (spatial multiplexing), známych ako prúdy (streams). Poskytuje aj väčší dosah. Prenosová rýchlosť je 100 Mb/s a na fyzickej vrstve až 250 Mb/s,
- 802.11ac – vylepšuje 802.11n. Prenosové rýchlosti dosahujú až 433 Mb/s cez spatial multiplexing, prípadne až 1.3 Gbps pri použití troch antén. Pracuje len na frekvencii 5 GHz a podporuje širšie kanály od 80 MHz do 160 MHz.

Okrem toho autor (Beal, 2019) spomína aj menej známe špecifikácie, ktoré stoja za zmienku:

- 802.11ac Wave 2 – vylepšuje jej predchodkyňu, ktorá používa Multi User MIMO (MU-MIMO) technológiu, kde viacero používateľov s jednou alebo viacerými anténami komunikujú medzi sebou. Teoretická prenosová rýchlosť dosahuje až 6.93 Gb/s,
- 802.11ad – je vo vývoji, operuje na frekvencii 60 GHz a poskytuje prenosovú rýchlosť až do 7 Gb/s,
- 802.11ah – známa ako WiFi HaLow, operuje na frekvencii 900 MHz, teda nižšie ako všetky ostatné. Dosah má takmer dvakrát väčší ako ostatné WiFi technológie a je schopná preniknúť cez steny a iné bariéry.

### 1.2.2.4 IEEE 802.11p - DSRC

V konferenčnom príspevku „IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments“ autori (Abdelgader, a iní, 2014) uvádzajú, že IEEE 802.11p je štandard z roku 2010, ktorý umožňuje bezdrôtový prístup v dopravnom prostredí (Wireless Access in Vehicular Environment – WAVE). WAVE je špecifikované v štandarde s označením IEEE 1609. IEEE 802.11p je základom pre DSRC (Dedicated Short-Range Communication). Vylepšuje IEEE 802.11 a pridáva požadovanú podporu aplikáciám ITS (Intelligent Transportation Systems). Patrí sem napríklad výmena dát medzi vozidlami jazdiacimi vysokou rýchlosťou (V2V) a medzi vozidlami a cestnou infraštruktúrou (V2I). Pracuje na rádiových frekvenciách od 5.850 do 5.925 GHz s rozostupom medzi kanálmi o veľkosti 20, 10 alebo 5 MHz. Umožňuje výmenu informácií na vzdialenosť 1 kilometra s prenosovou rýchlosťou od 3 do 27 Mbit/s pri rýchlosti 260 km/s. Obrázok 4 ukazuje, že môže operovať až na siedmich kanáloch, pričom frekvenčné pásma 5.860 GHz (kanál 172) a 5.920 GHz (kanál 184) sú určené pre bezpečnosť – prvý poskytuje seriózne bezpečnostné riešenia a druhý predstavuje ochranu voči preťaženiu na ostatných kanáloch. Kanál 178 (5.890 GHz) je zodpovedný za nadviazanie spojenia a vysielanie. Zvyšné kanály sú alokované pre obojsmernú komunikáciu.




*Obrázok 4 Frekvenčné pásma kanálov štandardu IEEE 802.11p*

### 1.2.2.5 Porovnanie

Všetky spomenuté technológie pracujú na alebo okolo frekvenčného pásma 2.4 GHz. WiFi, akú poznáme z každodenného života, najčastejšie pracuje vo forme hviezdicovej topológie, to znamená, že komunikácia medzi zariadeniami je realizovaná pomocou sprostredkovateľa (routra). Vie však pracovať aj v Mesh topológii, čo je pri V2V systéme základ. Bluetooth a ZigBee sú energeticky úspornejšie oproti WiFi, avšak poskytujú aj nižšie prenosové rýchlosti. Bluetooth vie podľa uvedených informácií poskytnúť najlepší dosah signálu, avšak štandard 802.11p poskytujúci garantovaný

minimálne 300 metrový dosah je dostatočný na včasnú reakciu vodiča pri primeranej rýchlosti. Pri dvoch vozidlách idúcich vysokou rýchlosťou (130 km/h) priamo proti sebe (spolu 260 km/h) je to však asi málo. Takéto vozidlo prejde za 1 sekundu 36 metrov, 300 metrov dosiahne za ani nie 8,5 sekundy. Obe však majú k dispozícii 150 metrovú brzdnú dráhu, čiže približne 4,25 sekundy. Približne 4,25 sekundy by trvalo, kým by sa vozidlá stretli. Otázka je, či by obe vozidlá stihli včas zabrzdiť. To záleží od toho, pri akej vzdialenosti od seba začnú medzi sebou komunikovať (čím viac, tým lepšie) a ako rýchlo dokážu vyhodnotiť, či treba začať brzdiť.

#### 1.2.2.6 Ostatné bezdrôtové technológie

Za zmienku stojí aj bezdrôtová technológia Z-Wave. Je dosť podobná ZigBee. Rozdielom je, že Z-Wave je vyvíjaný a udržiavaný súkromnou organizáciou a poskytuje nižšie prenosové rýchlosti. Pracuje s maximálne 232 zariadeniami. Z-Wave je považavaný však o niečo spoľahlivejšiu technológiu (Blank, 2016).

Okrem vyššie spomenutých bezdrôtových technológií sme sa dopátrali aj k ďalším, pre nás menej známym technológiam ako ANT, Thread, DASH7, SigFox, nWave, 6LoWPAN. Niektoré z nich majú dosah len niekoľko metrov, iné naopak aj niekoľko kilometrov, všetky však ponúkajú nízke prenosové rýchlosti. Kvôli tomu nemusia byť vhodné na implementáciu V2V systému pri posielaní/prijímaní väčšieho objemu dát.

### 1.2.3 Bezpečnosť V2V systému

„Pridanie komunikácie medzi vozidlami a cestnou infraštruktúrou umožňujú skvelé nové služby (napríklad bezpečnú jazdu, rýchejší presun, a podobne). Z bezpečnostného hľadiska to však vytvára nové hrozby. Je pravdepodobné, že „zlý človek“ sediaci pri ceste s bezdrôtovým prístupom sa môže nabúrať do nášho vozidla alebo do semaforov pred nami (Vamosi, 2017).“

Autor (Vamosi, 2017) vo svojom blogu „New V2V communication could give hackers a free ride“ ďalej uvádza, že v roku 2015 poukázal výskumník Cesar Cerrudo na značnú zraniteľnosť tisícok cestných bezdrôtových senzorov. Senzory používali bezdrôtovú technológiu ZigBee, kde chýbalo šifrovanie a autentifikácia. Prístup k zopár cestným senzorom sa nemusí zdať ako nebezpečné. Avšak, čo ak útočník zmení

semafory na červenú a zastaví dopravu vo všetkých smeroch alebo nebodaj opačne? Takéto niečo sa stalo v Los Angeles v roku 2009. Použitá bola síce iná technológia, ale na niekoľko hodín boli odstavené strategické križovatky. Keďže sa vozidlá hýbu, aj v opačnom smere, nie je čas na nadviazanie/akceptovanie spojenia a autentifikáciu, ktoré poznáme zo štandardnej WiFi siete. Preto štandard IEEE 802.11p umožňuje okamžitú komunikáciu hneď ako je vozidlo v dosahu. Prístupovým bodom odovzdáva dočasné tokeny, ktoré zabezpečia hladký a relatívne anonymný prechod.

Autor (Koon, 2019) vo svojom článku hovorí, že NHTSA tiež stanovuje, že systémy V2V nebudú zhromažďovať, vysielať ani zdieľať osobné informácie medzi vozidlami, a ani nebudú umožňovať sledovanie konkrétneho vodiča alebo vozidla.

### 1.2.4 Prekážky brániace zavedeniu V2V systému do praxe

V2V systém stále nie je zavedený v praxi. Čelí viacerým prekážkam, ktoré tomu bránia, a ktoré musia byť vyriešené aby sa tak stalo.

Autor (Graham, 2017) uvádza vo svojom článku „Vehicle-to-vehicle communication tech going nowhere“: „Jedným z problémov brániacich implementácii V2V systému je chýbajúca dohoda výrobcov automobilov o jednotných normách, financovaní a ochrane osobných údajov.“ Ďalej dodáva: „Na zabezpečenie silnej komunikačnej siete je potrebná povinná štandardizácia spracovania, analytických schopností a technických možností. Existuje však obava, že dosiahnutie dohody o štandardoch (komunikačného) protokolu, bezpečnostných otázkach a protiopatreniach môže zastaviť akýkoľvek posun trhu vpred.“

V našej práci sme doteraz opisovali V2V systém ako systém pracujúci na úrovni LAN siete. Ak by V2V systém bol napojený aj na internet (teda WAN), tak autor (Graham, 2017) tvrdí: „Systém V2V produkuje veľa údajov, ktoré vyžadujú škálovateľné cloudové úložisko schopné tieto dáta uložiť.“

„Ďalším veľkým problémom s posunom V2V je nesúlad štátnych a miestnych zákonov, ktorými sa dnes V2V systém riadi. Počas Obamovej administratívy sa regulačné orgány snažili urýchliť uvedenie V2V systému na trh, ale pod administratívou Trumpa sa to zastavilo,“ vo svojom článku uvádza autor (Graham, 2017).

## 1.3 MINIPOČÍTAČ RASPBERRY PI

Raspberry Pi je cenovo dostupný počítač veľkosti kreditnej karty, ktorý sa pripája na TV alebo monitor na zobrazovanie obsahu a používa klasickú klávesnicu a myš na ovládanie. Jedná sa o šikovné malé zariadenie, ktoré umožňuje ľuďom rôznych vekových kategórií skúmať výpočtovú vedu a naučiť sa ako programovať v programovacích jazykoch ako Scratch, Python a iných. Je schopné zvládnuť všetko čo dokáže klasický desktopový počítač od prehliadania webu až po prehrávanie videí a hrania videohier. Ba čo viac, Raspberry Pi má široké využitie v rôznych oblastiach po celom svete ako zdravotníctvo, priemysel, automatizácia a podobne. Projekt vedie vzdelávacia nadácia Raspberry Pi Foundation so sídlom vo Veľkej Británii (raspberrypi.org, 2019).

Na obrázku 5 je znázornený opisovaný minipočítač. Na obrázku sa konkrétne nachádza jeho štvrtá generácia.

*Obrázok 5 Minipočítač Raspberry Pi 4*

### 1.3.1 Raspberry Pi 3

Stránka (thepihut.com, 2016) prezentuje Raspberry Pi 3 (model B) v poradí ako tretiu generáciu tohto kultového minipočítača. Tak ako jeho predchodcovia, aj tento model má veľkosť kreditnej karty a využitie nájde v rôznych oblastiach. Oproti jeho predchodcom ponúka väčší výkon a síce je desaťkrát rýchlejší ako jeho prvá generácia. Taktiež má v sebe zabudované moduly pre bezdrôtovú sieť WiFi a Bluetooth. Technická špecifikácia tohto minipočítača je nasledovná:

- čipset Broadcom BCM2387,

- štvorjadrový procesor s taktom 1,2 GHz a 64 bitovou architektúrou ARM Cortex-A53,
- zabudovaný modul pre bezdrôtovú sieť WiFi štandardu IEEE 802.11b/g/n,
- zabudovaný modul pre bezdrôtovú sieť Bluetooth 4.1,
- pamäť RAM o veľkosti 1 GB,
- štyri USB 2.0 konektory,
- 4-pólový stereo výstup a kompozitný video port,
- HDMI výstup v HD kvalite,
- 10/100 Mbit Ethernet konektor pre LAN sieťový kábel,
- CSI konektor pre kameru,
- DSI konektor pre dotykový displej,
- slot pre pamäťovú microSD kartu, ktorá slúži ako diskové úložisko,
- microUSB konektor pre napájanie,
- 40-pinový GPIO (General-Purpose Input/Output) konektor,

V rámci našej diplomovej práce chceme použiť práve túto generáciu minipočítača.

### 1.3.2 Raspberry Pi 4

V roku 2019 vyšla štvrtá generácia tohto minipočítača. Stránka (thepihut.com, 2019) uvádza viacero vylepšení oproti tretej generácii. A dosť výrazných. Medzi najhlavnejšie patrí zvýšenie taktu procesora na 1,5 GHz, pri pamäti RAM máme na výber 1 GB, 2 GB alebo 4 GB verziu, moduly WiFi štandardu IEEE 802.11a/c a Bluetooth 5.0, 10/100/1000 Mbit Ethernet konektor pre LAN sieťový kábel, dva USB 3.0 konektory a mnoho ďalších.

## 1.4 ROZHRANIE OBD (II)

OBD je skratkou pre On-Board Diagnostics. Je to rozhranie, ktoré je zabudované v takmer každom vozidle jazdiacom po cestách. Výnimku tvoria vozidlá spred roku 1996. Dnes z výroby nevyjde vozidlo bez tohto rozhrania. V sedemdesiatych a začiatkom osemdesiatych rokov začali výrobcovia používať elektronické zariadenia na kontrolu funkčnosti motora a diagnostiku jeho problémov. V priebehu rokov sa

rozhranie OBD stalo sofistikovanejším a v polovici 90. rokov bolo predstavené rozhranie OBD-II. To dokáže skontrolovať takmer úplne celý motor a monitoruje aj podvozok, trup vozidla, príslušenstvo a diagnostickú sieť. Jednoducho povedané, vie nám zhodnotiť zdravotný stav vozidla.

Hlavný dôvod, prečo rozhranie OBD vzniklo, bol nadmerný smog v Los Angeles v roku 1966. Kongres v tom čase schválil „Clean Air Act in 1970“ a založil agentúru na ochranu životného prostredia Environmental Protection Agency (EPA). EPA vydala emisné normy a požiadavky na údržbu vozidiel. Aby výrobcovia splnili tieto nároky, začali do svojich vozidiel inštalovať elektronicky riadené systémy ako vstrekovanie paliva alebo zapaľovanie. Senzory potom merali výkon motora a znečistenie. Každý výrobca si najskôr vytvoril vlastný systém, a preto v roku 1988 spoločnosť Society of Automotive Engineers (SAE) štandardizovala konektor známy ako OBD vrátane testovacích signálov. Na obrázku 6 je znázornený OBD-II konektor používaný v dnešnej dobe.

Celkovo dnes existuje päť základných protokolov s minimálnymi rozdielmi, ktoré OBD-II používa na komunikáciu medzi vozidlom a skenovacím prístrojom. Sú to ISO 9141, KWP2000, SAE J1850 VPW, SAE J1850 PWM a najnovší z roku 2008 – CAN. Európske vozidlá používajú protokol štandardu ISO 9141 alebo KWP2000. Príkazy pri každom protokole sú však definované podľa štandardu SAE J1979.

OBD-II konektor je štandardne umiestnený pod palubnou doskou na strane vodiča. Cez rozhranie OBD-II dokážeme aj preprogramovať výkonnostné parametre vozidla, nejedná sa však o takzvané prečipovanie vozidla, pretože rozhranie OBD-II to už nedovoľuje (obdii.com, 2011).

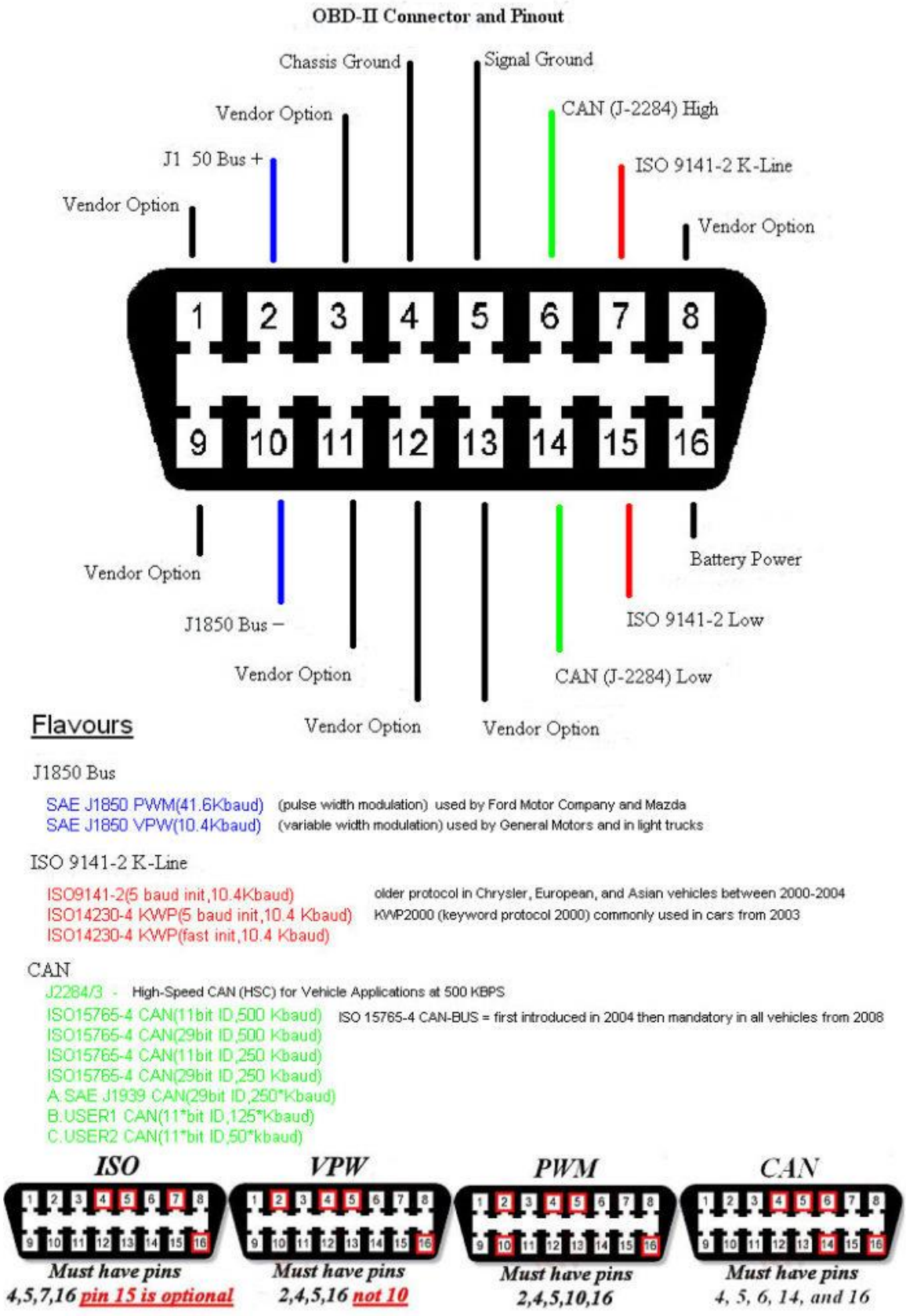
Aby sme získali určitú informáciu o vozidle, musíme dať vozidlu príkaz. Rozhranie OBD-II definuje širokú škálu príkazov označovanú ako PIDs (Parameter IDs). Príkaz je číselný kód, pomocou ktorého získame požadovanú informáciu o vozidle, ako napríklad: rýchlosť vozidla, počet otáčok motora za minútu a podobne. Príkazy sú kategorizované na diagnostické služby, a to nasledovne:

- 01 – aktuálne informácie o vozidle,
- 02 – rámcové informácie o vozidle,
- 03 – uložené chybové kódy,
- 04 – resetuje uložené chybové kódy a hodnoty,
- 05 – výsledky testu monitorovania senzoru kyslíka,

- 06 – výsledky testu monitorovania palubných systémov,
- 07 – nedávne chybové kódy detekované počas aktuálnej alebo poslednej jazdy,
- 08 – kontrola palubného systému,
- 09 – všeobecné informácie o vozidle (napríklad VIN číslo),
- 0A – permanentné chybové kódy.

Nie všetky vozidlá však podporujú aj všetky príkazy. Výrobcovia si definujú aj vlastné príkazy (obdsol.com, 2020).

My sami si doma môžeme diagnostikovať vozidlo. Stačí si zohnať OBD-II skener, ktorý vysiela informácie prostredníctvom bezdrôtovej siete Bluetooth, spárovať ho s telefónom, stiahnúť si aplikáciu na to určenú a sledovať „živé“ informácie o vozidle. Takýto skener je dôležitou súčasťou aj našej diplomovej práce.




*Obrázok 6 Schéma OBD-II konektora a rozloženie PIN-ov podľa protokolov*

# 2 CIELE DIPLOMOVEJ PRÁCE

V diplomovej práci sa pokúsime vytvoriť funkčný V2V systém pomocou minipočítača Raspberry Pi, programovacieho jazyka Python a rozhrania OBD-II.

Hlavným cieľom diplomovej práce je:

- navrhnúť V2V systém po hardvérovej a softvérovej stránke,
- pokúsiť sa zrealizovať tento návrh krok za krokom – v prípade, ak narazíme na niečo, čo sa nedá zrealizovať podľa vytvoreného návrhu, tak pokúsiť sa nájsť náhradné riešenie a to následne zrealizovať,
- v prípade úspešnej realizácie návrhu otestovať vytvorený V2V systém v praxi a vyhodnotiť dosiahnuté výsledky, a naopak, pri neúspešnej realizácii návrhu identifikovať a vysvetliť príčinu, prečo sa nám ho nepodarilo zrealizovať.

# 3 NÁVRH, REALIZÁCIA A TESTOVANIE V2V SYSTÉMU

V rámci tejto kapitoly (3) sa pokúsime vytvoriť V2V systém a zistiť, či v praxi funguje alebo nie. Táto kapitola je rozdelná na tri podkapitoly. V prvej podkapitole (3.1) si spravíme návrh V2V systému – aký hardvér budeme potrebovať, aký operačný systém použijeme, aké softvérové balíčky budeme potrebovať, na aké logické celky bude rozdelený, aké budú mať jednotlivé logické celky úlohy, kde a ako budeme programovať. V druhej podkapitole (3.2) sa budeme snažiť krok za krokom návrh zrealizovať. Samozrejme, to čo vytvoríme, budeme priebežne testovať vo vozidle, avšak získané poznatky a odhalené problémy, ktoré zásadne nezabránia napredovaniu vývoja, rozoberieme až v nasledujúcej podkapitole (3.3). V tretej podkapitole (3.3), ak sa nám podarí vytvoriť funkčný V2V systém rozoberieme, ako ho vo vozidle spustiť, nastaviť a testovať. Rozoberieme tu aj poznatky a problémy, s ktorými sme sa stretli v priebehu vývoja a testovania. V jednotlivých oddieloch zároveň zhodnotíme, či systém funguje alebo nefunguje, kedy funguje a kedy nefunguje.

## 3.1 NÁVRH

V tejto podkapitole (3.1) popíšeme koncept, z čoho sa V2V systém bude skladať (hardvérová stránka) a ako by mohol fungovať (softvérová stránka). Najskôr popíšeme hardvérovú stránku. Základom bude minipočítač Raspberry Pi. My použijeme generáciu 3 model B (#1). Výhodou práve tohto minipočítača oproti jeho predošlým generáciám je už zabudovaný WiFi modul. Ideálne je ho uložiť do ochrannej krabičky, aby neprišlo k jeho poškodeniu. Budeme potrebovať zobrazovaciu jednotku, teda nejaký displej. My sme sa rozhodli pre 5palcový dotykový LCD displej značky Waveshare s rozlíšením 800x480 (#2). K nemu potrebujeme dva káble, ktorými sa k minipočítaču pripája, a sú to: USB-A/microUSB pre napájanie (#3) a HDMI pre zobrazovanie (#4). Budeme potrebovať dva microUSB napájacie adaptéry. Jeden do klasickej elektrickej zásuvky (#5) a jeden do zapaľovania vo vozidle s prúdom 2A (#6). Ako úložisko používa minipočítač Raspberry Pi pamäťovú microSD kartu (#7). Minimálne musí mať 8 GB. Aby sme ju mohli vložiť do slotu v počítači, tak budeme k nej potrebovať adaptér (#8). Prenos údajov z vozidla do minipočítača bude zabezpečovať OBD-II skener (#9) cez Bluetooth. My sme sa rozhodli pre „Vgate ELM327 Bluetooth OBD2 V2.1 Scanner.“ Jednotlivé komponenty môžete vidieť na obrázku 7. Okrem týchto komponentov

budeme ešte potrebovať veľký monitor s HDMI vstupom, klávesnicu a myš (ideálne 2 v 1), pripojenie do WAN siete cez LAN kábel a samozrejme vozidlo. Keďže V2V systém je o dvoch a viacerých vozidlách, tak všetky vyššie spomenuté komponenty potrebujeme dva a viackrát, okrem malého displeja a s ním súvisiacich káblov, microUSB napájacieho adaptéra do elektrickej zásuvky, veľkého monitora s HDMI vstupom, klávesnice, myši, LAN káblu.

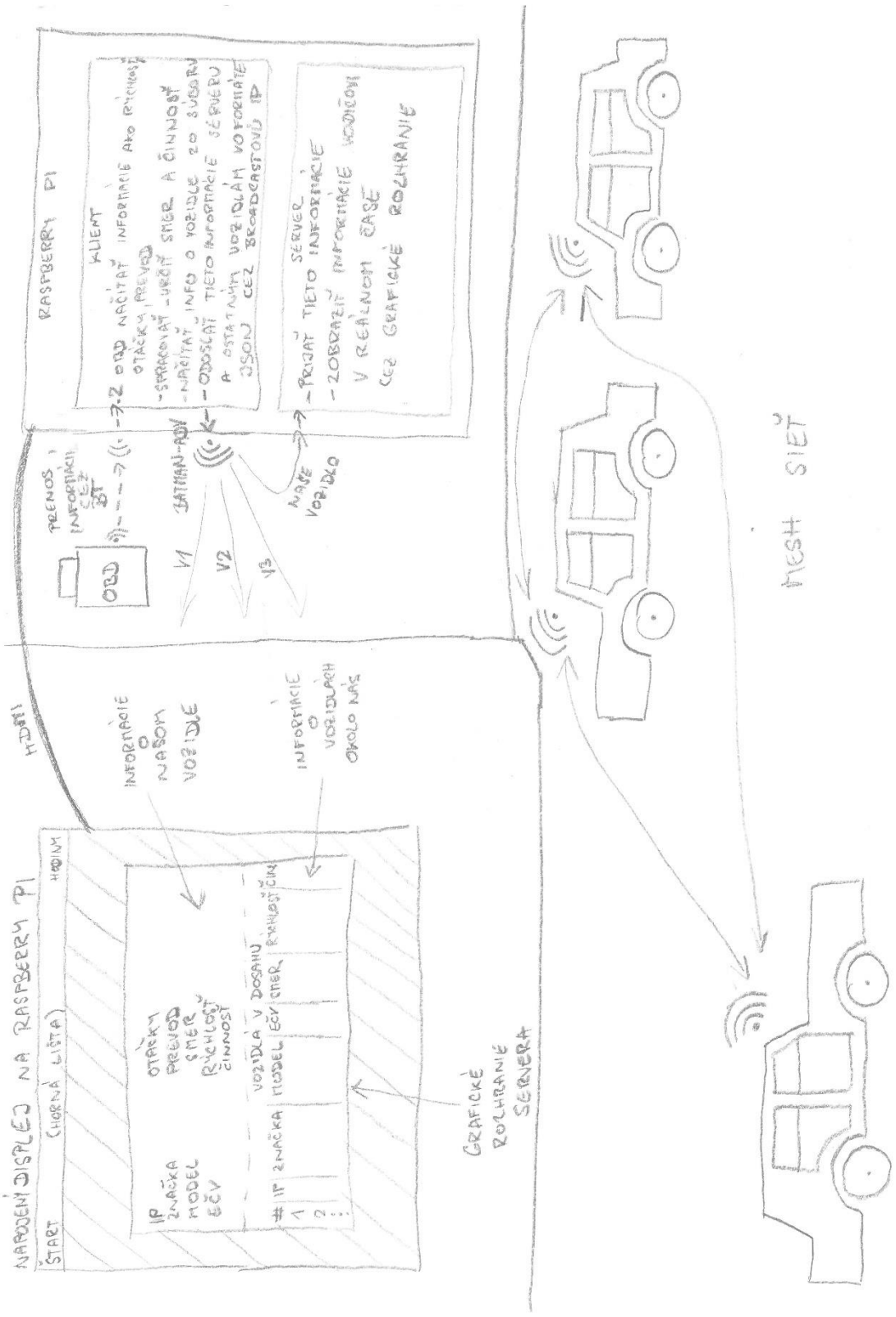

*Obrázok 7 Hardvérové komponenty nášho V2V systému*

Teraz rozoberieme, ako by sme chceli V2V systém riešiť z pohľadu softvérovej stránky. Ako operačný systém použijeme Raspbian, ktorý je šitý na mieru práve pre minipočítač Raspberry Pi. Minipočítač Raspberry Pi generácie 3 podporuje spojenie zariadení cez Bluetooth a operačný systém Raspbian má v sebe zabudovaného

softvérového Bluetooth manažéra, ktorý sa o toto vie postarať. Spojenie s OBD-II skenerom by sme chceli riešiť práve pomocou tohto manažéra. Pri predbežnom zisťovaní, aké informácie vieme získať cez rozhranie OBD-II sme zistili, že informácie ako značka vozidla, model vozidla, evidenčné číslo vozidla (EČV), nedokážeme cez neho získať. Tieto informácie však budeme chcieť zobraziť vodičovi na displeji. Preto ich pre každé vozidlo vyplníme manuálne. Vytvoríme si preto nejaký jednoduchý textový súbor, kde budú uložené tieto informácie. Ako sme spomínali v analýze súčasného stavu, tak V2V systém funguje v sieťovej topológii mesh. Pri hľadaní na internete sme narazili na softvérový balíček BATMAN-ADV, ktorý sa javí ako riešenie tohto problému. Programovanie budeme realizovať v programovacom jazyku Python verzie 3.7 a priamo na minipočítači. Operačný systém Raspbian ponúka na vývoj Python aplikácií integrované vývojové prostredie ThonnyIDE, čiže programovať budeme v ňom. Programovať budeme štýlom Dependency Injection. Získavanie informácií o vozidle cez rozhranie OBD-II by sme chceli realizovať pomocou Python balíčka `obd`. Zo skúseností s programovacím jazykom Python vieme, že ak chceme získať IP adresu zariadenia, tak musíme na to použiť Python balíček `netifaces`. Tieto dva teda určite budeme musieť doinštalovať, nakoľko nie sú súčasťou programovacieho jazyka Python. Grafické rozhranie budeme riešiť cez Python balíček `tkinter`. Sieťovú komunikáciu zase cez Python balíček `socket`. Tieto dva inštalovať nebudeme, pretože sú súčasťou programovacieho jazyka Python. Náš V2V systém bude rozdelený na dva logické celky, to jest aplikácia klienta a aplikácia servera. Aplikácia klienta bude mať úlohu sluhu. Jej hlavnou úlohou bude načítať informácie o našom vozidle z textového súboru (značka vozidla, model vozidla, evidenčné číslo vozidla – ĚCV), čítať informácie o vozidle cez rozhranie OBD-II (rýchlosť vozidla, počet otáčok motora za minútu, zaradený prevodový stupeň motora), spracovať tieto informácie (určiť z nich smer jazdy vozidla a činnosť vozidla), rozposlať broadcast-om všetky tieto informácie vozidlám okolo nás a nášmu vozidlu (to sú aplikácie servera), vo formáte JSON. Aplikácia klienta nebude zobrazovať žiadne informácie vodičovi. Naopak aplikácia servera bude mať úlohu pána. Tá bude pomocou grafického rozhrania zobrazovať aktuálne informácie vodičovi. Jednak bude zobrazovať informácie o našom vozidle, a jednak, v tabuľke „Vozidlá v dosahu," informácie o ostatných vozidlách v našom okolí. Z vyššie spomenutých informácií nám u „vozidiel v dosahu" však nebude zobrazovať počet otáčok motora za minútu a zaradený prevodový stupeň

motora, pretože z pohľadu nášho vozidla to pre nás nie sú dôležité informácie. Ďalší problém, ktorý nám tu vzniká, je spustenie V2V systému. Celý tento proces musí byť plne automatizovaný. To znamená, pri spustení minipočítača sa aplikácie klienta aj servera musia spustiť automaticky. V prípade zlyhania sa systém musí vedieť reštartovať bez intervencie používateľa, pretože minipočítač nemusí mať pripojené žiadne periférne zariadenia. Nakoniec, keďže by sme celý náš systém chceli skúsiť aspoň s tromi vozidlami, tak inštalácia nášho systému v operačnom systéme Raspbian musí byť tiež maximálne zjednodušená, respektíve automatizovaná. Preto by sme chceli vytvoriť inštalačný skript, ktorý pri jeho spustení spraví túto prácu za nás.

Návrh riešenia nášho V2V systému po hardvérovej aj softvérovej stránke môžete vidieť na obrázku 8.




*Obrázok 8 Návrh riešenia nášho V2V systému z pohľadu hardvérovej a softvérovej stránky*

## 3.2 REALIZÁCIA

V predchádzajúcej podkapitole (3.1) sme načrtli jednoduchý koncept, ako by to mohlo fungovať a ako by sme to chceli zrealizovať. V tejto podkapitole (3.2) sa pustíme do samotnej realizácie V2V systému krok za krokom. Ak narazíme na problém, že sa to nedá riešiť ako sme to navrhli, tak na to upozorníme a opíšeme nové riešenie. Jednotlivé súbory nášho systému sa budeme snažiť ukladať na jednom mieste, v adresári `/home/pi/Desktop/v2v`. Ten bude slúžiť ako koreňový adresár a zároveň ako lokálny Git repozitár, ktorý budeme priebežne synchronizovať s jeho vzdialeným Git repozitárom na URL adrese `https://github.com/david-bojnansky/diploma-v2v`.

### 3.2.1 Inštalácia operačného systému Raspbian

Existuje viacero známych distribúcií operačných systémov, ktoré sú prispôsobené pre minipočítač Raspberry Pi, ako napríklad Ubuntu MATE, Ubuntu Core, Ubuntu Server, Windows 10 IoT Core a mnoho ďalších. My sme sa však rozhodli pre oficiálny operačný systém Raspbian, vytvorený priamo od nadácie Raspberry Pi Foundation, ktorá za minipočítačmi stojí. Nejdeme tu do detailu popisovať postup inštalácie tohto operačného systému, pretože ten je možné nájsť v oficiálnom návode „Installing operating system images“ na URL adrese `https://www.raspberrypi.org/documentation/installation/installing-images/README.md`. Pri inštalácii sme sa aj my držali práve tohto návodu. Pri inštalácii máme na výber, či ju vykonáme cez inštalátor NOOBS alebo bez neho, čiže priamym stiahnutím obrazu (súbor `.img`) operačného systému Raspbian. My sme ju realizovali bez neho a tento obraz sme stiahli na URL adrese `https://www.raspberrypi.org/downloads/raspbian/`. Na výber máme viacero obrazov, my sme sa však rozhodli pre „Raspbian Buster with desktop and recommended software.“ Za poznamenie stojí, že súbor obrazu je zaobalený v `.zip` súbore, čiže sťahujeme `.zip` súbor a nie priamo `.img` súbor. Prvá časť inštalácie spočíva v zapísaní `.img` súboru operačného systému Raspbian na pamäťovú microSD kartu pomocou aplikácie BalenaEtcher (v prostredí Windows). Následne pamäťovú microSD kartu zasunieme do minipočítača, pripojíme periférne zariadenia a zapneme ho. Druhá časť inštalácie spočíva v preklikaní sa rôznymi nastaveniami samotnej

inštalácie operačného systému, ako napríklad akú jazykovú mutáciu používať, aké rozloženie klávesnice používať, aké časové pásmo používať a podobne. Je vhodné si zvoliť nejaký jedinečný názov pre minipočítač v sieti, napríklad `rpi-1`, `rpi-2` a podobne. Buď ho upraviť priamo pri inštalácii alebo cez príkazový riadok príkazom `sudo raspi-config`, prípadne `sudo nano /etc/hostname`.

### 3.2.2 Inštalácia softvérového balíčka BlueZ

Pri prvých pokusoch o spárovanie minipočítača Raspberry Pi s OBD-II skenerom sme dospeli k tomu, že zabudovaný softvérový Bluetooth manažér v operačnom systéme Raspbian nepostačuje na splnenie našich požiadaviek – poloautomatizácia párovania a automatizácia nadviazania spojenia pri spustení minipočítača. Preto sme pátrali po alternatíve na internete a dostali sme sa k softvérovému balíčku BlueZ. „BlueZ je oficiálny Bluetooth protokol pre linux. Jedná sa o projekt s otvoreným zdrojovým kódom distribuovaný pod licenciou GNU General Public Licence (GPL),“ uvádza sa na webstránke (bluez.org, 2020). V operačnom systéme Raspbian ho cez príkazový riadok nainštalujeme pomocou príkazu `sudo apt-get install bluez`. Inštaláciou tohto softvérového balíčka získame k dispozícii príkaz `bluetoothctl`, ktorým vieme spárovať OBD-II skener s minipočítačom, pričom tento proces vieme vďaka tomu aj poloautomatizovať, a ktorý rozoberáme v oddieli 3.2.4. Okrem neho poskytuje aj príkaz `rfcomm`, pomocou ktorého vieme nadviazať spojenie s OBD-II skenerom a tento proces vieme taktiež vďaka tomu automatizovať.

### 3.2.3 Informácie o našom vozidle (súbor `vehicleinfo.txt`)

Súbor `vehicleinfo.txt` v sebe nesie základné informácie o našom vozidle. Súbor má jednoduchú štruktúru – jedna hodnota na jednom riadku a nepoužíva validáciu ako napríklad JSON. Štruktúra je nasledovná: Bluetooth MAC adresa OBD-II skenera, Bluetooth párovací kód, značka vozidla, model vozidla a nakoniec evidenčné číslo vozidla (EČV). Tieto informácie nevieme získať cez rozhranie OBD-II, preto ich musíme manuálne určiť pre každé vozidlo. Tento súbor bude programom čítaný teda po riadkoch a bude čítať len prvých päť riadkov. To, čo už je na šiestom a ďalších riadkoch, program nebude nezaujímať. My sme si tam uložili Bluetooth MAC adresy

všetkých našich OBD-II skenerov. Príklad súboru `vehicleinfo.txt` môžete vidieť v zdrojovom kóde 1.

```
00:00:00:33:33:33
1234
Škoda
Octavia
NR751GR
```

*Zdrojový kód 1 Súbor vehicleinfo.txt*

### 3.2.4 Príkaz pre spárovanie OBD-II skenera s minipočítačom (súbor `obd2rpi/pairwithobd`)

Pri úplne prvom spustení minipočítača vo vozidle ešte nemáme žiadnym spôsobom prepojený OBD-II skener a minipočítač. V našom prípade riešime toto prepojenie cez bezdrôtovú sieť Bluetooth. Obe zariadenia teda musia podporovať tento typ siete. Prv, než môžu obe zariadenia medzi sebou komunikovať, musia sa najskôr spárovať. Toto spárovanie nám stačí spraviť len raz. Kedy a ako ho vykonáme si povieme až pri praktickom testovaní (v oddieli 3.3.1). Tu si len rozoberieme nami vytvorený príkaz spustiteľný v príkazovom riadku, ktorý toto párovanie realizuje v jednom kroku. Implementáciu tohto príkazu môžete vidieť v zdrojovom kóde 2. Samotnému súboru musíme v jeho vlastnostiach nastaviť oprávnenie na exekúciu v príkazovom riadku. Tento príkaz využíva aj softvérový balíček `expect`, ktorý nainštalujeme pomocou príkazu `sudo apt-get install expect`.

Expect je program, ktorý „sa rozpráva“ s inými interaktívnymi programami alebo skriptami, ktoré vyžadujú interakciu používateľa. Funguje spôsobom, že keď iný program alebo skript očakáva vstup, tak mu ho poskytne podľa definovaných pravidiel bez zásahu používateľa. Je to nástroj, ktorý slúži na automatizáciu (Ebrahim, 2017).

Ako prvé si zo súboru `vehicleinfo.txt` vytiahneme Bluetooth MAC adresu OBD-II skenera a párovací kód. Spustíme interaktívny príkaz `bluetoothctl` cez `spawn` a potom čakáme na znak mriežky. Všeobecne platí, že keď je príkaz `bluetoothctl` pripravený prijímať (ďalšie) príkazy, tak vypíše znak mriežky. Ak je teda pripravený, pošleme mu prvý príkaz, aby zapol Bluetooth. Ak je úspešne zapnutý,

pošleme mu príkaz aby zapol Bluetooth Agenta. Úloha Bluetooth Agenta je riadiť párovanie. Ak je úspešne zapnutý, registrujeme štandardného agenta, ktorý riadi párovanie pomocou párovacieho kódu. Ak existuje spojenie minipočítača s OBD-II skenerom, tak toto spojenie odstránime. Spustíme vyhľadávanie zariadení v okolí, počkáme 20 sekúnd a potom vyhľadávanie zastavíme. Ak sme našli OBD-II skener, tak s ním vytvoríme dôveryhodný vzťah a spustíme samotný proces párovania. Počas párovania si to vypýta párovací kód, ktorý sa automaticky zadá podľa toho, aký sme vytiahli zo súboru `vehicleinfo.txt`. Spojenie medzi OBD-II skenerom a minipočítačom by v tejto chvíli malo byť úspešne vytvorené. Následne už len ukončíme interakciu.

```
#!/usr/bin/expect -f

set address [exec head -1 /home/pi/Desktop/v2v/vehicleinfo.txt]
set pin [exec head -2 /home/pi/Desktop/v2v/vehicleinfo.txt | tail -1]
set prompt "#"

spawn bluetoothctl
expect $prompt

send -- "power on\n"
expect "succeeded"
expect $prompt

send -- "agent on\n"
expect "registered"
expect $prompt

send -- "default-agent\n"
expect "successful"
expect $prompt

send -- "remove $address\n"
expect -re "not available|removed"
expect $prompt

send -- "scan on\n"
expect "started"
expect $prompt

sleep 20

send -- "scan off\n"
```

```
expect "stopped"
expect $prompt

send -- "trust $address\n"
expect "succeeded"
expect $prompt

send -- "pair $address\n"
expect "PIN"
send -- "$pin\n"
expect "successful"
expect $prompt

send "quit\n"
expect "eof"
```

*Zdrojový kód 2 Súbor obd2rpi/pairwithobd*

### 3.2.5 Príkaz pre nadviazanie spojenia OBD-II skenera a minipočítača (súbor `obd2rpi/obd2rpi`)

Keď už máme úspešne spárované obe zariadenia (OBD-II skener a minipočítač), tak ďalší krok je nadviazať medzi nimi spojenie, aby si mohli vymieňať informácie. Toto spojenie nám treba nadviazať pri každom spustení minipočítača a pri každom zlyhaní tohto spojenia sa musí dokázať obnoviť. Celý tento proces bude musieť byť automatizovaný, pretože k minipočítaču nebudú musieť byť pripojené žiadne periférne zariadenia. Preto si vytvoríme príkaz spustiteľný v príkazovom riadku, ktorý toto spojenie nadviaže v jednom kroku. Implementáciu tohto príkazu môžete vidieť v zdrojovom kóde 3. Samotnému súboru musíme v jeho vlastnostiach nastaviť oprávnenie na exekúciu v príkazovom riadku. Tento príkaz využíva aj softvérový balíček BlueZ, presnejšie príkaz `rfcomm` z neho. Bluetooth MAC adresu OBD-II skenera si vyťahujeme zo súboru `vehicleinfo.txt`.

```
rfcomm connect hci0 "$(head -1 /home/pi/Desktop/v2v/vehicleinfo.txt)"
```

*Zdrojový kód 3 Súbor obd2rpi/obd2rpi*

### 3.2.6 Služba Bluetooth spojenia OBD-II skenera s minipočítačom (súbor `lib-systemd-system/v2v-obd2rpi.service`)

Keď spustíme „holý“ minipočítač (to jest bez displeja, myšky, klávesnice) vo vozidle, tak chceme, aby sa bez intervencie používateľa nadviazalo Bluetooth spojenie OBD-II skenera s minipočítačom. Na riešenie tohto problému je vhodné použiť systém služieb Systemd, ktorý je súčasťou operačného systému Raspbian.

„Systemd obsluhuje jednotky, sleduje ich stav, vie ich reštartovať pri zlyhaní, logovať chyby. Umožňuje obmedzovať zdroje ako dostupnú pamäť, využitie procesora a podobne. Ďalej sleduje procesy pomocou `cgroups` namiesto klasického PID, vďaka čomu nemôžu vyviaznuť ani pri takzvanom double-forking, a preto vždy vieme, ktoré procesy daná jednotka vlastní. Systemd tiež sprostredkováva komunikáciu medzi procesmi a umožňuje nastaviť spúšťanie jednotiek až vo chvíli, keď sú naozaj potrebné (takzvaný on-demand loading),“ uvádza autor (Knížek, 2016).

Služba `v2v-obd2rpi.service` zabezpečuje nadviazanie Bluetooth spojenia OBD-II skenera s minipočítačom Raspberry Pi. Implementáciu môžete vidieť v zdrojovom kóde 4. Je automaticky spustená pri boote systému hneď po tom, ako skupina služieb `multi-user.target` úspešne beží (sekcia `[Install]`). Služba `v2v-obd2rpi.service` je silne závislá na skupine služieb `multi-user.target`, to znamená, že ak skupina služieb `multi-user.target` nie je úspešne spustená, tak služba `v2v-obd2rpi.service` sa ani len nepokúsi spustiť (sekcia `[Unit]`). V sekcii `[Service]` definujeme správanie samotnej služby. V `PIDFile=` sme nastavili cestu k PID súboru, ktorý počas života služby v sebe uchováva číslo procesu, ku ktorému služba patrí. V `ExecStart=` uvádzame cestu ku skriptu, ktorý sa má vykonať pri spustení tejto služby, no ešte predtým toto spustenie oddialime o 10 sekúnd v nastavení `ExecStartPre=`. Štandardný výstup (stdin) a chybu (stderr) presmerujeme do logových súborov (`StandardOutput=` a `StandardError=`). V `KillSignal=` sme nastavili signál, ktorý sa má poslať procesu pri jeho ukončení (`SIGINT` je obdoba Ctrl+C v linuxe). Toto je dôležité z pohľadu príkazu `rfcomm`, ktorý používame pre nadväzovanie Bluetooth spojenia, a ktorý sa v príkazovom riadku ukončuje práve kombináciou kláves Ctrl+C. `Restart=` hovorí o tom, že služba sa vždy musí pokúsiť opätovne spustiť - reštartovať (skúša to do nekonečna). `RestartSec=` určuje, o koľko sekúnd sa má reštart služby

oddialiť. Keď už máme službu `v2v-obd2rpi.service` definovanú, tak musíme ju registrovať a aktivovať v systéme služieb Systemd cez príkazový riadok pomocou troch príkazov. Príkazom `ln -sf /home/pi/Desktop/v2v/lib-systemd-system/v2v-obd2rpi.service /lib/systemd/system/v2v-obd2rpi.service` vytvoríme v adresári `/lib/systemd/system` symbolický odkaz na súbor služby `v2v-obd2rpi.service`. Príkazom `systemctl daemon-reload` obnovíme démona služieb Systemd a príkazom `systemctl enable v2v-obd2rpi.service` následne aktivujeme službu.

```
[Unit]
Requires=multi-user.target
After=multi-user.target

[Service]
PIDFile=/run/v2v-obd2rpi.pid
ExecStartPre=/bin/sleep 10
ExecStart=/home/pi/Desktop/v2v/obd2rpi/obd2rpi
StandardOutput=append:/home/pi/Desktop/v2v/lib-systemd-system/log/v2v-obd2rpi.log
StandardError=append:/home/pi/Desktop/v2v/lib-systemd-system/log/v2v-obd2rpi.err
KillSignal=SIGINT
Restart=always
RestartSec=10s

[Install]
WantedBy=multi-user.target
```

*Zdrojový kód 4 Súbor lib-systemd-system/v2v-vehicleclient.service*

### 3.2.7 Inštalácia a konfigurácia softvérového balíčka BATMAN-ADV

„BATMAN-ADV pracuje iba na 2. vrstve ISO/OSI modelu. Používa a smeruje (alebo lepšie mostuje) ethernetové rámce. Emuluje virtuálny sieťový prepínač všetkých zúčastnených uzlov. Preto sa zdá, že všetky uzly sú lokálne prepojené, takže všetky vyššie prevádzkové protokoly nebudú ovplyvnené žiadnymi zmenami v sieti. Podporuje takmer akýkoľvek protokol, najmä však: IPv4, IPv6, DHCP, IPX," uvádza webstránka (kernel.org, 2020). Inštaláciu vykonáme cez príkaz `sudo apt-get install batctl`.

#### 3.2.7.1 Konfigurácia

Keď už máme softvérový balíček BATMAN-ADV nainštalovaný, tak nastavíme sieťové rozhranie, pod ktorým bude minipočítač bezdrôtovo komunikovať s okolitými vozidlami. Aby sme mali všetky konfiguračné súbory pokope, tak v koreňovom adresári nášho systému sme vytvorili súbor `etc-network/interfaces`, ktorého obsah je uvedený v zdrojovom kóde 5. Súbor `/etc/network/interfaces` zmeníme na symbolický odkaz odkazujúci práve na tento súbor pomocou príkazu `sudo ln -sf /home/pi/Desktop/v2v/etc-network/interfaces /etc/network/interfaces`. V súbore `etc-network/interfaces` sa nachádza iba jediný príkaz (`source-directory`), ktorý slúži na importovanie konfiguračných súborov z adresára `/etc/network/interfaces.d`. Máme dva takéto súbory. Jeden pre sieťové rozhranie `wlan0` (WiFi) a druhé pre sieťové rozhranie `bat0` (BATMAN-ADV).

```
# interfaces(5) file used by ifup(8) and ifdown(8)

# Please note that this file is written to be used with dhcpcd
# For static IP, consult /etc/dhcpcd.conf and 'man dhcpcd.conf'

# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d
```

*Zdrojový kód 5 Súbor etc-network/interfaces*

Sieťové rozhranie `wlan0` je definované v súbore `etc-network-interfaces.d/wlan0` podľa zdrojového kódu 6. Nastavenia pre toto sieťové rozhranie musia byť na všetkých minipočítačoch identické, to jest MTU veľkosť, kanál, SSID, mód a vymyslená MAC adresa prístupového bodu. Pre tento súbor vytvoríme symbolický odkaz v adresári `/etc/network/interfaces.d` pomocou príkazu `ln -sf /home/pi/Desktop/v2v/etc-network-interfaces.d/wlan0 /etc/network/interfaces.d/wlan0`.

```
auto wlan0
iface wlan0 inet manual
    mtu 1532
    wireless-channel 1
    wireless-essid VehicleNetwork
```

```
    wireless-mode ad-hoc
    wireless-ap 02:12:34:56:78:9A
```

*Zdrojový kód 6 Súbor etc-network-interfaces.d/wlan0*

Sieťové rozhranie `bat0` je definované v súbore `etc-network-interfaces.d/bat0` podľa zdrojového kódu 7. Tu platí to isté, nastavenia musia byť identické na všetkých minipočítačoch s výnimkou sieťovej IP adresy. Tú nastavujeme rozdielne od vozidla k vozidlu, pričom musí byť jedinečná, čo je zároveň vhodné, pretože každé vozidlo vieme jednoznačne identifikovať (v sieti). Na svete však existuje veľké množstvo vozidiel a súčasná IPv4 nie je vhodná na takéto použitie, hoci my ju tu používame. Riešením tohto problému je IPv6. Pre tento súbor vytvoríme symbolický odkaz v adresári `/etc/network/interfaces.d` pomocou príkazu `ln -sf /home/pi/Desktop/v2v/etc-network-interfaces.d/bat0 /etc/network/interfaces.d/bat0`.

```
auto bat0
iface bat0 inet static
    address 169.254.0.?
    netmask 255.255.255.0
    pre-up /usr/sbin/batctl if add wlan0
```

*Zdrojový kód 7 Súbor etc-network-interfaces.d/bat0*

Na záver ešte musíme určiť (fyzické) sieťové rozhranie (v našom prípade to je `wlan0`), ktoré má sieťové rozhranie `bat0` používať na komunikáciu, pretože `bat0` je virtuálne sieťové rozhranie. Následne obe sieťové rozhrania aktivujeme pomocou príkazu `ifconfig`. Celé to zakomponujeme do súboru, ktorý bude slúžiť ako príkaz, čiže treba mu navyše nastaviť vo vlastnostiach súboru aj oprávnenia pre exekúciu v príkazovom riadku. Implementáciu môžete vidieť v zdrojovom kóde 8.

```
#!/bin/bash

# Povieme BATMAN-ADV, ktoré sieťové rozhranie má používať
sudo batctl if add wlan0

# Zapneme sieťové rozhrania
sudo ifconfig wlan0 up
```

```
sudo ifconfig bat0 up
```

*Zdrojový kód 8 Súbor batman/batman*

Na záver musíme ešte do troch súborov doplniť tri príkazy. Vieme to spraviť jednoduchými príkazmi cez príkazový riadok. Príkazom `echo "batman-adv" >> /etc/modules` pridáme príkaz `batman-adv` na koniec súboru `/etc/modules`. Príkazom `echo "denyinterfaces wlan0" >> /etc/dhcpcd.conf` pridáme príkaz `denyinterfaces` na koniec súboru `/etc/dhcpcd.conf`. Príkazom `echo "/home/pi/Desktop/v2v/batman/batman" >> /home/pi/.bashrc` pridáme príkaz `batman` na koniec súboru `/home/pi/.bashrc`. Jednotlivé príkazy detailnejšie rozoberáme v oddieli 3.2.14.

### 3.2.8 Príprava vývojového prostredia

Samotné programovanie systému budeme realizovať na minipočítači v operačnom systéme Raspbian. Ako textový editor budeme používať ThonnyIDE, ktorý je súčasťou operačného systému od jeho inštalácie. Ak náhodou nie je, tak ho doinštalujeme jednoducho cez príkazový riadok príkazom `sudo apt-get install python3-thonny`.

### 3.2.9 Inštalácia programovacieho jazyka Python a jeho balíčkov

Náš systém budeme programovať v programovacom jazyku Python verzie 3.7, ktorý by mal byť súčasťou operačného systému Raspbian od jeho inštalácie. Overiť jeho prítomnosť vieme cez príkazový riadok príkazom `python3 --version`. Ak tento príkaz vypíše chybu typu príkaz neexistuje, tak ho nainštalovaný nemáme. Následne ho nainštalujeme príkazom `sudo apt-get install python3.7`. Zatiaľ s určitosťou vieme, že budeme potrebovať dva Python balíčky, ktoré nie sú súčasťou balíčka `python3.7`. Sú to balíčky `netifaces` a `obd`. Oba nainštalujeme pomocou príkazu `sudo pip3 install netifaces https://github.com/brendan-w/python-OBD/archive/master.zip`.

*Dodatočne pridávame, že v druhom prípade sme prepísali* `obd` *na URL adresu GitHub archívu, pretože Python balíček* `obd` *poskytoval (v dobe písania) staršiu verziu*

*s chybou, ktorá nám spôsobovala problémy a v tomto archíve už je táto chyba opravená.*

### 3.2.10 Aplikácia klienta (adresár `vehicleclient`)

Adresár `vehicleclient` v sebe nesie súbory pre aplikáciu klienta. V našom prípade plní aplikácia klienta funkciu sluhu a má štyri úlohy:

- čítať informácie o našom vozidle cez rozhranie OBD-II,
- spracovať tieto informácie,
- rozposielať spracované informácie cez WiFi do prostredia okolo nášho vozidla, respektíve ďalším vozidlám so spustenou aplikáciou servera (vrátane nášho vozidla),
- monitorovať „zdravie“ Bluetooth spojenia medzi OBD-II skenerom a minipočítačom.

Z toho vyplýva, že aplikácia klienta žiadne informácie nezobrazuje. Aplikácia klienta sa skladá zo súborov `mycommands.py`, `myvehicle.py`, `obdmonitor.py` a `vehicleclient.py` (hlavný súbor aplikácie klienta).

#### 3.2.10.1 Definovanie vlastných OBD-II príkazov (súbor `mycommands.py`)

Python balíček `obd` definuje najpoužívanejšie OBD-II príkazy, ktoré sú definované štandardom rozhrania OBD-II. Nájdeme tu napríklad príkaz pre získanie aktuálnej rýchlosti vozidla alebo počtu otáčok motora za minútu. Ak niektorý príkaz nedefinuje, tak pomocou triedy `OBDCommand` nám umožňuje si takýto príkaz definovať. To sme sa pokúsili využiť pri probléme ako získať aktuálne zaradený prevodový stupeň motora. Python balíček `obd` takýto príkaz nedefinuje, tak sme sa snažili nájsť viac informácií na Google. Podarilo sa nám nájsť, že takýto príkaz štandard rozhrania OBD-II definuje. Patrí do diagnostickej služby `01` a PID má `A4`. Slovne je opísaný ako „Transmission Actual Gear.“ Podľa dokumentácie Python balíčka `obd`, ktorá hovorí ako si vytvoriť vlastný príkaz, sme sa pokúsili tento príkaz implementovať. Nachádza sa práve v tomto súbore (`mycommands.py`). Implementáciu môžete vidieť v zdrojovom kóde 9. Tam, kde konvertujeme bajty na číslo, tak to delíme číslom 4,

pretože zdroj uvádza, že táto informácia má veľkosť 4 bajty. Telo funkcie `gearDecoder` je založené na príklade uvedenom v spomínanej dokumentácii. Samotný príkaz je definovaný na konci súboru. Ako prvý argument je uvedený názov príkazu, následne jeho popis, číselný kód v binárnej podobe, počet bajtov informácie, odkaz na funkciu dekódujúcu informáciu, ECU (Engine Control Unit) filter na odfiltrovanie prichádzajúcich správ a nakoniec logická hodnota pre rýchlu odpoveď – súvisí len s internou funkcionalitou Python balíčka `obd`.

```
from obd import OBDCommand
from obd.protocols import ECU
from obd.utils import bytes_to_int

def gearDecoder(messages):
    data = messages[0].data

    # Odseknúť službu a PID
    data = data[2:]

    # Konvertovať bajty na číslo
    gear = bytes_to_int(data) / 4

    return gear


# PID (A4, resp. 01A4) a počet bajtov (4) nájdený tu:
# https://en.wikipedia.org/wiki/OBD-II_PIDs
# Aj pri zaradenej rýchlosti však dostávame hodnotu 0.
# Skúšali sme aj iné (kvalitnejšie) OBD skenery,
# avšak pri nich sa aplikácia ani nespustila
GEAR = OBDCommand("GEAR", "Transmission Actual Gear",
                  b"01A4", 4, gearDecoder, ECU.ALL, False)
```

*Zdrojový kód 9 Súbor vehicleclient/mycommands.py*

#### 3.2.10.2 Informácie o našom vozidle (súbor `myvehicle.py`)

Ďalšou časťou aplikácie klienta je trieda `MyVehicle`, ktorá je nositeľom informácií o našom vozidle (jej implementáciu môžete vidieť v zdrojovom kóde 10). Medzi tieto informácie patrí: IP adresa sieťového rozhrania, značka vozidla, model vozidla, evidenčné číslo vozidla (EČV), počet otáčok motora za minútu, zaradený prevodový stupeň motora, smer jazdy vozidla, rýchlosť vozidla, konštantná rýchlosť

vozidla a činnosť vozidla. Všetky tieto informácie sú v triede `MyVehicle` reprezentované vlastnosťami triedy. Pri vytvorení novej inštancie tejto triedy majú uvedené vlastnosti nastavenú hodnotu `None`. Pred použitím musíme vytvorený objekt najskôr inicializovať. To zabezpečuje metóda `init`. Ako argumenty prijíma názov sieťového rozhrania (v našom prípade to je `bat0`) a názov súboru s informáciami o našom vozidle (v našom prípade to je `vehicleinfo.txt`). Metóda `init` najskôr zavolá internú metódu `_figureOutMyIp`, ktorá pomocou Python balíčka `netifaces` a názvu sieťového rozhrania zistí, akú IP adresu má priradené dané sieťové rozhranie. Tú potom uloží do vlastnosti `ip`. Hneď na to zavolá internú metódu `_loadInfoAboutMe`, ktorá prečíta súbor s informáciami o našom vozidle, vytiahne odtiaľ značku, model a evidenčné číslo vozidla a tie potom uloží do vlastností `brand`, `model` a `vrn`. Keď už máme objekt inicializovaný, môžeme bezpečne volať ďalšie metódy. Pokiaľ chceme aktualizovať zvyšné vlastnosti, robíme tak pomocou metódy `update`. Tá prijíma ako argumenty počet otáčok motora za minútu, zaradený prevodový stupeň motora a rýchlosť vozidla. Počet otáčok motora za minútu uloží do vlastnosti `rotates`, rýchlosť vozidla zase do vlastnosti `speed`. Pred uložením rýchlosti však najskôr aktualizujeme činnosť vozidla (vlastnosť `action`), pretože, aby sme mohli určiť činnosť vozidla, musíme porovnať predchádzajúcu a aktuálnu rýchlosť vozidla. Aktualizáciu činnosti vozidla zabezpečuje interná metóda `_updateAction`, ktorá prijíma ako argument rýchlosť vozidla. V zdrojovom kóde 10 môžete vidieť rôzne podmienky, podľa ktorých sa určuje činnosť vozidla (stojí na mieste, zrýchľuje, brzdí, takmer stojí na mieste, zastavuje, spomaľuje), no základ tvorí vždy porovnanie predchádzajúcej s aktuálnou rýchlosťou vozidla. Ak nie je splnená žiadna z podmienok, tak vozidlo sa nachádza v konštantom pohybe a vtedy sa navyše do vlastnosti `constantSpeed` uloží aktuálna rýchlosť vozidla, ktorá je základom pre overenie, či sa vozidlo stále pohybuje v konštantom pohybe s rozdielom +/–2 km/h. Pred aktualizáciou činnosti vozidla sa ešte aktualizujú vlastnosti `gear` (teda zaradený prevodový stupeň motora) a direction (smer jazdy vozidla). Rozhranie OBD-II poskytuje hodnotu zaradeného prevodového stupňa motora v číselnej podobe. Interná metóda `_updateGear` premieňa túto číselnú hodnotu na textovú, ktorú poznáme z každodenného života. Ak je hodnota rovná 0, tak máme zaradený „neutrál.“ Ak je hodnota menšia ako 0, tak máme zaradenú „spiatočku.“ V opačnom prípade máme

zaradený „dopredný“ prevodový stupeň. Smer jazdy vozidla sa určuje na základe zaradeného prevodového stupňa motora a zabezpečuje to interná metóda `_updateDirection`. Ak je zaradený „neutrál,“ tak smer jazdy je „nikde.“ Ak je zaradená „spiatočka,“ tak smer jazdy je „dozadu.“ Ak nie je zaradený ani jeden z týchto prevodových stupňov, tak je zaradený „dopredný“ prevodovoý stupeň a smer jazdy je teda „dopredu.“ Metóda `toJson` zoberie všetky aktuálne vlastnosti a ich hodnoty zaobalí do formátu JSON, ktorý využijeme ako formát pre správy posielané aplikácii servera. Prijíma logický argument, či správa bude poslaná nášmu serveru alebo do prostredia ostatným vozidlám. Zmyslom toho je, aby sme do prostredia okolo nás neposielali zbytočné, prípadne citlivé dáta. Napríklad v našom prípade je zbytočné, aby ostatné vozidlá dostali informáciu, aký počet otáčok má náš motor alebo aký prevodový stupeň má zaradený náš motor. Metóda `toEncodedJson` sa správa tak isto ako metóda `toJson`, len je doplnená o zakódovanie výstupu. Poslednou metódou je `echo`, ktorá vráti alternatívnu hodnotu v prípade, že hodnota vlastnosti je `None`.

```
import json
import netifaces

class MyVehicle:
    ip = None               # IP adresa
    brand = None            # Značka
    model = None            # Model
    vrn = None              # EČV
    rotates = None          # Otáčky (za minútu)
    gear = None             # Prevodový stupeň
    direction = None        # Smer jazdy (číselný kód)
    directionAsText = None # Smer jazdy
    speed = None            # Rýchlosť (km/h)
    constantSpeed = None    # Konštantná rýchlosť (pomocná premenná)
    action = None           # Činnosť (číselný kód)
    actionAsText = None     # Činnosť

    def init(self, netIfName, infoFilename, nfs = netifaces):
        self._figureOutMyIp(netIfName, nfs)
        self._loadInfoAboutMe(infoFilename)


    def _figureOutMyIp(self, netIfName, netifaces):
        bat0 = netifaces.ifaddresses(netIfName)
        self.ip = bat0[netifaces.AF_INET][0]["addr"]
```

```
def _loadInfoAboutMe(self, infoFilename):
    with open(infoFilename, "r") as fp:
        fp.readline()         # MAC adresa OBD skenera
        fp.readline()         # Bluetooth párovací kód
        brand = fp.readline() # Značka
        model = fp.readline() # Model
        vrn = fp.readline()   # EČV

        # Z konca odstrániť znak nového riadku
        self.brand = brand[:-1]
        self.model = model[:-1]
        self.vrn = vrn[:-1]


def update(self, rotates, gear, speed):
    self.rotates = rotates

    self._updateGear(gear)
    self._updateDirection()

    # Činnosť sa určuje na základe porovnania
    # aktuálnej (speed) a predchádzajúcej
    # rýchlosti (self.speed)
    self._updateAction(speed)

    self.speed = speed


def _updateGear(self, gear):
    if gear == None:
        self.gear = None
    elif gear == 0:
        self.gear = "N"
    elif gear < 0:
        self.gear = "R"
    else:
        self.gear = "D:" + str(gear)


def _updateDirection(self):
    if self.gear == None:
        self.direction = None
        self.directionAsText = None
    elif self.gear == "N":
        self.direction = 0
        self.directionAsText = "Nikde"
    elif self.gear == "R":
```

```
            self.direction = -1
            self.directionAsText = "Dozadu"
        else:
            self.direction = 1
            self.directionAsText = "Dopredu"


    def _updateAction(self, speed):
        if speed == None:
            self.action = None
            self.actionAsText = None
        elif speed == 0:
            self.action = 0
            self.actionAsText = "Stojí na mieste"
        # Pri zrýchľovaní rozlišujeme,
        # či je vozidlo v konštantnom pohybe
        elif self.action != 2 and speed > self.speed:
            self.action = 1
            self.actionAsText = "Zrýchľuje!"
        # Detto
        elif self.action == 2 and speed >= self.constantSpeed + 3:
            self.action = 1
            self.actionAsText = "Zrýchľuje!"
        elif speed <= self.speed - 10:
            self.action = -4
            self.actionAsText = "BRZDÍ!!!"
        elif speed <= 5 and speed < self.speed:
            self.action = -1
            self.actionAsText = "Takmer stojí na mieste"
        elif speed <= 15 and speed < self.speed:
            self.action = -2
            self.actionAsText = "Zastavuje!"
        elif speed <= self.speed - 3:
            self.action = -3
            self.actionAsText = "Spomaľuje!"
        else:
            self.constantSpeed = speed
            self.action = 2
            self.actionAsText = "V konštantnom pohybe"


    def toJson(self, forMyServer = False):
        data = {
            "aboutMe": forMyServer,
            "ip": self.ip,
            "brand": self.brand,
            "model": self.model,
            "vrn": self.vrn,
```

```
            "direction": self.direction,
            "directionAsText": self.directionAsText,
            "speed": self.speed,
            "action": self.action,
            "actionAsText": self.actionAsText,
        }

        if forMyServer:
            data.update({
                "rotates": self.rotates,
                "gear": self.gear,
            })

        return json.dumps(data)


    def toEncodedJson(self, forMyServer = False):
        return self.toJson(forMyServer).encode()


    def echo(self, name, alt = "Zisťujem..."):
        val = getattr(self, name)

        if val == None:
            return alt
        else:
            return val
```

*Zdrojový kód 10 Súbor vehicleclient/myvehicle.py*

#### 3.2.10.3 Monitorovanie stavu Bluetooth spojenia (súbor `obdmonitor.py`)

Pre nadviazanie Bluetooth spojenia minipočítača Raspberry Pi s OBD-II skenerom používame protokol RFCOMM, ktorý napodobňuje RS-232 sériový port. V softvérovom balíčku BlueZ je implementovaný ako príkaz `rfcomm`. Tento príkaz nám funguje bez problémov, avšak sme našli jeden nedostatok. V prípade, že vytiahneme OBD-II skener z konektora, tak operačný systém si stále myslí, že Bluetooth spojenie je aktívne. Keď aj OBD-II skener napojíme do konektora naspäť, tak spojenie už nefunguje. Preto sme sa snažili nájsť riešenie. Nadviazanie Bluetooth spojenia je realizované vrámci Systemd služby `v2v-obd2rpi.service` (rozoberali sme ju v oddieli 3.2.6). Preto sme vytvorili triedu `ObdMonitor`, ktorá monitoruje „zdravie“ Bluetooth spojenia medzi minipočítačom Raspberry Pi a OBD-II skenerom.

Implementáciu môžete vidieť v zdrojovom kóde 11. Pri vytvorení objektu tejto triedy musíme ako argument zadať názov služby (v našom prípade to je `v2v-obd2rpi.service`). Metóda `updateTime` slúži na aktualizáciu internej vlastnosti `_lastUpdateAt`. Táto vlastnosť reprezentuje čas, kedy naposledy sme z OBD-II skenera dostali nejakú informáciu. Tento čas sa v podstate aktualizuje každých pár milisekúnd. Pre overenie „zdravia“ Bluetooth spojenia slúži metóda `isAlive`. V prípade, že od času poslednej aktualizácie (`_lastUpdateAt`) prešlo viac ako 10 sekúnd, tak vtedy považujeme Bluetooth spojenie za stratené. V takom prípade reštartujeme službu `v2v-obd2rpi.service`.

```
import subprocess
import time

class ObdMonitor:
    _lastUpdateAt = None

    def __init__(self, serviceName,
                 t = time.time, execute = subprocess.call):
        self._serviceName = serviceName
        self._time = t
        self._exec = execute


    def updateTime(self, _ = None):
        self._lastUpdateAt = self._time()


    def isAlive(self):
        if (self._time() > self._lastUpdateAt + 10):
            self._exec([
                "/bin/systemctl",
                "restart",
                self._serviceName,
            ])

            return False
        else:
            return True
```

*Zdrojový kód 11 Súbor vehicleclient/obdmonitor.py*

#### 3.2.10.4 Spúšťací skript aplikácie klienta (súbor `vehicleclient.py`)

Súbor `vehicleclient.py` plní funkciu hlavného súboru aplikácie klienta. To znamená, že pri spustení aplikácie klienta sa ako prvý spracúva práve tento súbor. Definujeme v ňom triedu `VehicleClient`, ktorá zase v kontexte tried plní funkciu hlavnej triedy. Jej implementáciu môžete vidieť v zdrojovom kóde 12. Na začiatku triedy máme definované konštanty. Konštanty `ALL_IP` a `ALL_PORT` predstavujú broadcastovú IP adresu s portom, na ktorej „počúvajú“ všetky vozidlá v okolitom prostredí. Konštanta `ALL_ADDR` predstavuje takzvaný tuple zložený z IP adresy a portu. Zvyšné tri konštanty predstavujú názov sieťového rozhrania (`bat0`), názov súboru s informáciami o našom vozidle (`vehicleinfo.txt`) a názov služby, ktorá sa stará o nadviazanie Bluetooth spojenia OBD-II skenera s minipočítačom (`v2v-obd2rpi.service`). Metóda `__init__` je konštruktor, ktorý prijíma ako argumenty požadované závislosti (princíp Dependency Injection). Ako prvý argument, ak neberieme do úvahy parameter `self`, prijíma objekt nášho vozidla (`MyVehicle`), potom objekt UDP socket-u (`socket.socket`), objekt rozhrania OBD-II v asynchrónnej forme (`obd.Async`), objekt pre monitorovanie zdravia rozhrania OBD-II (`ObdMonitor`). Za konštruktorom nasleduje definícia internej metódy `_registerShutdownHandler`, ktorá v bežiacom procese aplikácie klienta registruje kus kódu, ktorý sa vykoná pri ukončení daného procesu so signálom `INT` alebo `TERM`. Kus kódu, ktorý sa vykoná pri ukončení daného procesu je uvedený v metóde `stop`. V zásade sa ukončí Bluetooth spojenie medzi OBD-II skenerom a minipočítačom, a uzavrie sa UDP socket. Ďalej nasledujú interné metódy `_initUdpSock` a `_initObd`. V `_initUdpSock` povieme UDP socket-u, že bude vysielať formou broadcastu. V `_initObd` zase registruje nami vytvorený OBD-II príkaz v objekte rozhrania OBD-II, ďalej registrujeme OBD-II príkazy, ktorých hodnoty majú byť sledované v reálnom čase a nakoniec spustíme objekt rozhrania OBD-II, aby začal prijímať informácie z OBD-II skenera. Za povšimnutie stojí aj to, že pri registrácii OBD-II príkazu `RPM` uvádzame ako druhý argument odkaz na metódu `updateTime` objektu `ObdMonitor`. Na čo slúži táto metóda sme vysvetlovali v pododdieli 3.2.10.3. Jedná sa o to, že pri každej novej hodnote `RPM` v reálnom čase sa zavolá aj metóda `updateTime`. Metóda `start` v podstate volá jednotlivé interné metódy. Inicializujeme tu aj objekt nášho vozidla (`MyVehicle`). Pred internou metódou `_run`

voláme manuálne metódu `updateTime` objektu `ObdMonitor`. Má to charakter incializácie pred použitím práve v internej metóde `_run`. Na koniec sme si nechali internú metódu `_run`. Tá spustí cyklus podmienený „zdravým“ Bluetooth spojením OBD-II skenera a minipočítača, ktorý je po každej iterácii prerušený na 1 sekundu. Na začiatku cyklu získame nami požadované hodnoty: počet otáčok motora za minútu, zaradený prevodový stupeň motora a rýchlosť vozidla. Následne aktualizujeme informácie o našom vozidle cez metódu `update`. Keď to máme, tak najskôr pošleme tieto informácie aplikácii servera nášho vozidla (pri volaní funkcie `toEncodedJson` je ako argument explicitne uvedená hodnota `True`) a potom do prostredia okolo nás (pri volaní funkcie `toEncodedJson` je ako argument implicitne uvedená hodnota `False`). Ako sme uviedli, tak naša trieda `VehicleClient` má aj metódu `start`. Samotné spustenie aplikácie klienta sa udeje práve po zavolaní tejto metódy. A to realizujeme práve pod definíciou tejto triedy.

```
import mycommands
import netifaces
import obd
import signal
import socket
import sys
import time

from myvehicle import MyVehicle
from obdmonitor import ObdMonitor

class VehicleClient:
    ALL_IP = "169.254.0.255" # Všetci v dosahu (broadcast) vrátane nás
    ALL_PORT = 20000
    ALL_ADDR = (ALL_IP, ALL_PORT)
    NETWORK_INTERFACE_NAME = "bat0"
    INFO_FILENAME = "/home/pi/Desktop/v2v/vehicleinfo.txt"
    OBD_SERVICE_NAME = "v2v-obd2rpi.service"

    def __init__(self, vehicle, udpSock, obd, obdMon, allAddr = ALL_ADDR):
        self._allAddr = allAddr
        self._vehicle = vehicle
        self._udpSock = udpSock
        self._obd = obd
        self._obdMon = obdMon
```

```
def _registerShutdownHandler(self):
    signal.signal(signal.SIGINT, self.stop)
    signal.signal(signal.SIGTERM, self.stop)


def _initUdpSock(self):
    self._udpSock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)


def _initObd(self):
    self._obd.supported_commands.add(mycommands.GEAR)
    self._obd.watch(obd.commands.RPM, self._obdMon.updateTime)
    self._obd.watch(mycommands.GEAR)
    self._obd.watch(obd.commands.SPEED)
    self._obd.start()


def _run(self, sleep):
    while self._obdMon.isAlive():
        rotates = self._obd.query(obd.commands.RPM).value
        gear = self._obd.query(mycommands.GEAR).value
        speed = self._obd.query(obd.commands.SPEED).value

        if rotates != None:
            rotates = rotates.magnitude

        if speed != None:
            speed = speed.magnitude

        self._vehicle.update(rotates, gear, speed)

        # Odošleme informácie o našom vozidle lokálnemu serveru
        self._udpSock.sendto(self._vehicle.toEncodedJson(True),
                             (self._vehicle.ip, self._allAddr[1]))

        # Odošleme informácie o našom vozidle do prostredia
        # okolo (vrátane) nás
        self._udpSock.sendto(self._vehicle.toEncodedJson(),
                              self._allAddr)

        sleep(1)


def start(self, netIfName = NETWORK_INTERFACE_NAME,
          infoFilename = INFO_FILENAME,
          nfs = netifaces, sleep = time.sleep):
    self._registerShutdownHandler()
    self._vehicle.init(netIfName, infoFilename, nfs)
```

```
            self._initUdpSock()
            self._initObd()
            self._obdMon.updateTime()
            self._run(sleep)


        def stop(self, sigNum = None, csf = None):
            try:
                self._obd.close()
            except Exception as e:
                print(e, file=sys.stderr)

            try:
                self._udpSock.close()
            except Exception as e:
                print(e, file=sys.stderr)


if __name__ == "__main__":
    vehicle = MyVehicle()
    udpSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    obdConn = obd.Async()
    obdMon = ObdMonitor(VehicleClient.OBD_SERVICE_NAME)

    client = VehicleClient(vehicle, udpSock, obdConn, obdMon)

    try:
        client.start()
    except KeyboardInterrupt:
        pass
    finally:
        client.stop()
```

*Zdrojový kód 12 Súbor vehicleclient/vehicleclient.py*

### 3.2.11 Aplikácia servera (adresár `vehicleserver`)

Adresár `vehicleserver` v sebe nesie súbory pre aplikáciu servera. V našom prípade plní aplikácia servera funkciu pána a má štyri úlohy:

- prijať predspracované informácie vyslané aplikáciou klienta od nášho a ostatných vozidiel okolo nás,
- spracovať prijaté informácie z pohľadu aplikácie servera,
- udržovať aktualizovaný zoznam vozidiel okolo nás (pridávať nové vozidlá v dosahu do zoznamu a odstraňovať staré vozidlá mimo dosahu zo zoznamu),

- zobrazovať informácie v reálnom čase a v grafickej podobe vodičovi vozidla.

Aplikácia servera sa skladá zo súborov `vehicle.py`, `foreignvehicles.py`, `objectmaker.py`, `gui.py` a `vehicleserver.py` (hlavný súbor aplikácie servera).

#### 3.2.11.1 Informácie o našom alebo cudzom vozidle (súbor `vehicle.py`)

Prvou časťou aplikácie servera je súbor `vehicle.py`, ktorý definuje triedu `Vehicle`. Implementáciu triedy `Vehicle` môžete vidieť v zdrojovom kóde 13. Táto trieda je nositeľom informácií o našom alebo cudzom vozidle, čiže z tohto pohľadu je univerzálna. Informácie, ktoré táto trieda nesie, sú uložené vo vlastnostiach triedy a sú to: sieťová IP adresa vozidla, značka vozidla, model vozidla, evidenčné číslo vozidla (EČV), počet otáčok motora za minútu, zaradený prevodový stupeň motora, smer jazdy vozidla, rýchlosť vozidla a činnosť vozidla. Okrem týchto vlastností je definovaná ešte interná vlastnosť `_lastUpdateAt`, ktorá reprezentuje čas poslednej aktualizácie informácií o vozidle. Všetky vlastnosti sú prednastavené na hodnotu `None`. Ako prvá je definovaná metóda `update`, ktorá aktualizuje informácie o vozidle podľa zadaných argumentov a navyše aktualizuje aj čas poslednej aktualizácie informácií (internej vlastnosti `_lastUpdateAt` nastaví čas v danom momente). Metóda `isReachable` overuje, či vozidlo je ešte v dosahu. Ak je aktuálny čas (argument `time`) menší alebo rovný ako čas poslednej aktualizácie informácií o vozidle (interná vlastnosť `_lastUpdateAt`) + 3 sekundy, tak dané vozidlo je stále v dosahu. Táto metóda sa používa len pri overovaní prítomnosti cudzích vozidiel (teda nie nášho, to by nemalo zmysel). Predposlednou metódou je metóda `echo`, ktorá vráti alternatívnu hodnotu v prípade, že hodnota vlastnosti je `None`. Poslednou metódou je metóda `fromJson`. Tá prijíma dáta typu JSON, ktoré boli vyslané aplikáciou klienta. Môžeme ju zavolať ako metódu triedy alebo existujúceho objektu. V prípade, že ju zavoláme ako metódu triedy, tak vytvorí nový objekt a podľa prijatých dát aktualizuje informácie pomocou metódy `update`. V opačnom prípade, teda ak ju zavoláme ako metódu existujúceho objektu, urobí vlastne to isté, len bez prvého kroku – vytvorenia nového objektu, a namiesto toho použije existujúci objekt. Za povšimnutie stojí ešte parameter `requiresAboutMe`. Ak je jeho hodnota `True`, tak informácie sa aktualizujú iba

v prípade, že dáta obsahujú informácie o našom vozidle. Tu platí jednoduché pravidlo. Ak objekt `Vehicle` predstavuje naše vozidlo, tak tento parameter sa explicitne nastavuje na hodnotu `True`. Ak predstavuje cudzie vozidlo, tak ho explicitne nastavovať nemusíme, pretože implicitne je nastavený na hodnotu `False`.

```
import json
import time

class Vehicle:
    ip = None
    brand = None
    model = None
    vrn = None
    rotates = None
    gear = None
    direction = None
    directionAsText = None
    speed = None
    action = None
    actionAsText = None
    _lastUpdateAt = None

    def update(self, ip, brand, model, vrn, rotates, gear,
               direction, directionAsText, speed, action,
               actionAsText, t = time.time):
        self.ip = ip
        self.brand = brand
        self.model = model
        self.vrn = vrn
        self.rotates = rotates
        self.gear = gear
        self.direction = direction
        self.directionAsText = directionAsText
        self.speed = speed
        self.action = action
        self.actionAsText = actionAsText
        self._lastUpdateAt = t()


    def isReachable(self, time):
        return time <= self._lastUpdateAt + 3


    def echo(self, name, alt = "Zisťujem..."):
        val = getattr(self, name)
```

```
        if val == None:
            return alt
        else:
            return val


    def fromJson(self, data, requiresAboutMe = False):
        data = json.loads(data)

        if requiresAboutMe and not data.get("aboutMe"):
            return None

        if self == None:
            self = Vehicle()

        self.update(data.get("ip"),
                    data.get("brand"),
                    data.get("model"),
                    data.get("vrn"),
                    data.get("rotates"),
                    data.get("gear"),
                    data.get("direction"),
                    data.get("directionAsText"),
                    data.get("speed"),
                    data.get("action"),
                    data.get("actionAsText"))

        return self
```

*Zdrojový kód 13 Súbor vehicleserver/vehicle.py*

#### 3.2.11.2 Zoznam cudzích vozidiel okolo nás (súbor `foreignvehicles.py`)

Vozidlá okolo nás, teda cudzie vozidlá, musíme udržovať v neuástle aktualizovanom zozname. Ak sa nejaké vozidlo dostane do nášho okolia, tak sa do tohto zoznamu musí pridať, a naopak, ak sa dostane mimo dosah, tak sa z tohto zoznamu musí odstrániť. Preto sme vytvorili triedu s názvom `ForeignVehicles`, ktorá plní túto úlohu. Implementáciu triedy `ForeignVehicles` môžete vidieť v zdrojovom kóde 14. Má viaceré metódy vrátane magických. Samotný zoznam vozidiel je uložený v internej vlastnosti `_vehicles`. Iterátor je zase uložený v internej vlastnosti `_it`. Iterovanie objektu v cykle zabezpečujú magické metódy `__iter__` a `__next__`. Magická metóda `__len__` vracia počet vozidiel v zozname pri zavolaní funkcie `len` s objektom ako argumentom. Magická metóda `__getitem__` umožňuje získať

vozidlo zo zoznamu indexovým spôsobom (`[i]`). Metóda `has` overuje, či zoznam obsahuje vozidlo s danou sieťovou IP adresou. Metóda `get` získa vozidlo podľa danej sieťovej IP adresy alebo keď v zozname také vozidlo nie je, tak vráti `None`. Metóda `add` pridá na koniec zoznamu vozidlo. Posledná metóda `removeAllUnreachable` odstráni zo zoznamu všetky vozidlá, ktoré sa stratili z dosahu.

```
class ForeignVehicles:
    _it = None

    def __init__(self):
        self._vehicles = []


    def __iter__(self):
        self._it = iter(self._vehicles)
        return self


    def __next__(self):
        return next(self._it)


    def __len__(self):
        return len(self._vehicles)


    def __getitem__(self, i):
        return self._vehicles[i]


    def has(self, ip):
        return self.get(ip) != None


    def get(self, ip):
        for v in self._vehicles:
            if v.ip == ip:
                return v

        return None


    def add(self, vehicle):
        self._vehicles.append(vehicle)
```

```
    def removeAllUnreachable(self, time):
        for i, v in enumerate(self._vehicles):
            if not v.isReachable(time):
                del self._vehicles[i]
```

*Zdrojový kód 14 Súbor vehicleserver/foreignvehicles.py*

#### 3.2.11.3 Továreň na tvorbu objektov (súbor `objectmaker.py`)

Trieda `ObjectMaker` predstavuje pomocnú triedu na tvorbu objektov. Nakoniec obsahuje len metódu, ktorá vytvorí objekt cuzdieho vozidla (trieda `Vehicle`). Implementáciu triedy `ObjectMaker` môžete vidieť v zdrojovom kóde 15.

```
from vehicle import Vehicle

class ObjectMaker:
    def foreignVehicle(self, json):
        return Vehicle.fromJson(None, json)
```

*Zdrojový kód 15 Súbor vehicleserver/objectmaker.py*

#### 3.2.11.4 Grafické rozhranie (súbor `gui.py`)

Aplikácia servera získané informácie o našom vozidle a cudzích vozidlách zobrazuje vodičovi v reálnom čase. Trieda `Gui` reprezentuje grafické rozhranie. Jedná sa o klasické okno, aké poznáme napríklad z Windows-u. Nejdeme do detailu opisovať ako táto trieda funguje, pretože to nie je z pohľadu tejto aplikácie kľúčové. Niečo málo si však povieme. Grafické rozhranie je založené na Python balíčku `tkinter`. Trieda `Gui` je odvodená od triedy `Frame`. Implementáciu triedy `Gui` môžete vidieť v zdrojovom kóde 16. Nový objekt vytvoríme zavolaním konštruktora s argumentami objektu nášho vozidla a zoznamu cudzích vozidiel. Interná metóda `_createMyVehicleView` vytvára pohľad pre informácie o našom vozidle (vrchná časť okna). Naopak, interná metóda `_createForeignVehiclesView` vytvára pohľad vo forme tabuľky pre informácie o cudzích vozidlách v našom okolí (spodná časť okna). Metódy `updateMyVehicle` a `updateForeignVehicles` aktualizujú tieto pohľady. V prvom prípade o našom vozidle a v druhom prípade o cudzích vozidlách v našom okolí. Metóda `invokeLater` je obdoba funkcie

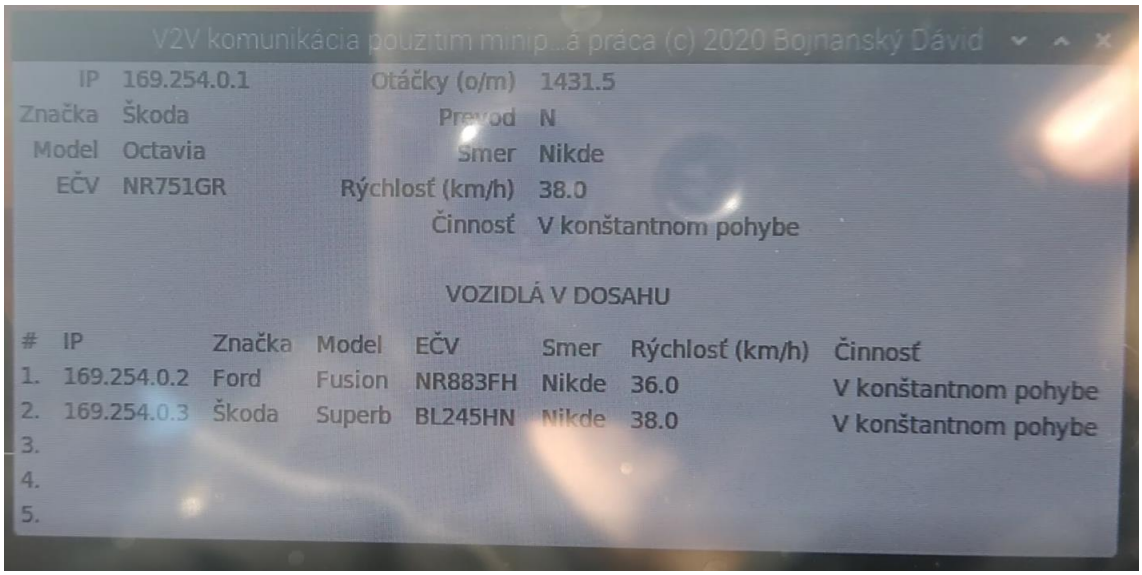
`SwingUtilities.invokeLater` z programovacieho jazyka Java. Pomocou tejto metódy vieme zabezpečiť, že požadovaný kód bude bežať vo vlákne grafického rozhrania. S grafickým rozhraním je možné manipulovať len z tohto vlákna. Skúšali sme to aj z hlavného vlákna, avšak k požadovaným zmenám (to jest k prepísaniu informácií) nedošlo. Nakoniec sú tu definované metódy `show` a `close`, kde v prvom prípade zobrazí okno a v druhom prípade ho zatvorí. Ukážku grafického rozhrania môžete vidieť na obrázku 9.

```
from tkinter import *

class Gui(Frame):
    _ipLabel = None
    _brandLabel = None
    _modelLabel = None
    _vrnLabel = None
    _rotatesLabel = None
    _gearLabel = None
    _directionLabel = None
    _speedLabel = None
    _actionLabel = None
    _afterId = None

    def __init__(self, master, vehicle, foreignVehicles):
        Frame.__init__(self, master)
        self._master = master
        self._vehicle = vehicle
        self._foreignVehicles = foreignVehicles
        self._foreignVehicleLabels = []


    def _createLabel(self, table, name, title, row, col):
        l = Label(table, text=title)
        l.grid(row=row, column=col * 2, sticky=E)

        l = Label(table, text="Zisťujem...")
        l.grid(row=row, column=(col * 2) + 1,
               sticky=W, padx=(10, 50))

        setattr(self, name + "Label", l)


    def _createMyVehicleView(self):
        t = Frame(master=self._master)
        t.grid(row=0, column=0, sticky=W)
```

```
        self._createLabel(t, "_ip", "IP", 0, 0)
        self._createLabel(t, "_brand", "Značka", 1, 0)
        self._createLabel(t, "_model", "Model", 2, 0)
        self._createLabel(t, "_vrn", "EČV", 3, 0)
        self._createLabel(t, "_rotates", "Otáčky (o/m)", 0, 1)
        self._createLabel(t, "_gear", "Prevod", 1, 1)
        self._createLabel(t, "_direction", "Smer", 2, 1)
        self._createLabel(t, "_speed", "Rýchlosť (km/h)", 3, 1)
        self._createLabel(t, "_action", "Činnosť", 4, 1)


    def _createForeignVehiclesView(self):
        table = Frame(master=self._master)
        table.grid(row=1, column=0, sticky=W)

        titles = ["#", "IP", "Značka", "Model", "EČV",
                  "Smer", "Rýchlosť (km/h)", "Činnosť"]

        l = Label(table, text="VOZIDLÁ V DOSAHU")
        l.grid(row=0, column=0,
               columnspan=len(titles),
               pady=(20, 10))

        for i, t in enumerate(titles):
            l = Label(table, text=t)
            l.grid(row=1, column=i, sticky=W, padx=(0, 10))

        for i in range(5):
            self._foreignVehicleLabels.append([])

            for j in range(len(titles)):
                l = Label(table)
                l.grid(row=i + 2, column=j,
                       sticky=W, padx=(0, 10))
                self._foreignVehicleLabels[i].append(l)


    def updateMyVehicle(self):
        self._ipLabel["text"] = self._vehicle.echo("ip")
        self._brandLabel["text"] = self._vehicle.echo("brand")
        self._modelLabel["text"] = self._vehicle.echo("model")
        self._vrnLabel["text"] = self._vehicle.echo("vrn")
        self._rotatesLabel["text"] = self._vehicle.echo("rotates")
        self._gearLabel["text"] = self._vehicle.echo("gear")
        self._directionLabel["text"] = self._vehicle.echo("directionAsText")
        self._speedLabel["text"] = self._vehicle.echo("speed")
        self._actionLabel["text"] = self._vehicle.echo("actionAsText")
```

```
def updateForeignVehicles(self):
    for i, row in enumerate(self._foreignVehicleLabels):
        row[0]["text"] = str(i + 1) + "."

        if i < len(self._foreignVehicles):
            v = self._foreignVehicles[i]

            row[1]["text"] = v.echo("ip")
            row[2]["text"] = v.echo("brand")
            row[3]["text"] = v.echo("model")
            row[4]["text"] = v.echo("vrn")
            row[5]["text"] = v.echo("directionAsText")
            row[6]["text"] = v.echo("speed")
            row[7]["text"] = v.echo("actionAsText")
        else:
            for j in range(7):
               row[j + 1]["text"] = ""


def invokeLater(self, runner):
    self._afterId = self.after(200, runner)


def show(self, invokeLaterRunner):
    self._master.title("V2V komunikácia ... 2020 Bojnanský Dávid")
    self._master.resizable(True, False)
    self._createMyVehicleView()
    self._createForeignVehiclesView()
    self.invokeLater(invokeLaterRunner)
    self._master.mainloop()


def close(self):
    if self._afterId != None:
        self.after_cancel(self._afterId)

    self._master.quit()
```

*Zdrojový kód 16 Súbor vehicleserver/gui.py*




*Obrázok 9 Grafické rozhranie aplikácie servera*

#### 3.2.11.5 Spúšťací skript aplikácie servera (súbor `vehicleserver.py`)

Súbor `vehicleserver.py` plní funkciu hlavného súboru aplikácie servera. To znamená, že pri spustení aplikácie servera sa ako prvý spracúva práve tento súbor. Definujeme v ňom triedu `VehicleServer`, ktorá zase v kontexte tried plní funkciu hlavnej triedy. Implementáciu môžete vidieť v zdrojovom kóde 17. Na začiatku triedy máme definované konštanty `IP` a `PORT`, ktoré predstavujú IP adresu sieťového rozhrania s portom, kde prijíma aplikácia servera informácie vyslané našim alebo cudzími vozidlami. Za povšimnutie stojí, že hodnota konštanty `IP` je nastavená na prázdny reťazec. To znamená, že aplikácia servera „počúva“ na všetkých priradených sieťových IP adresách (vrátane broadcast-ovej). Konštanta `ADDR` predstavuje takzvaný tuple zložený z IP adresy a portu. Zvyšné dve konštanty predstavujú názov sieťového rozhrania (`bat0`) a veľkosť buffer-u v bajtoch. Magická metóda `__init__` je konštruktor, ktorý prijíma ako argumenty požadované závislosti (princíp Dependency Injection). Ako prvý argument, ak neberieme do úvahy parameter `self`, prijíma objekt továrne na tvorbu objektov (`ObjectMaker`), potom objekt nášho vozidla (`Vehicle`), objekt UDP socket-u (`socket.socket`), objekt zoznamu cudzích vozidiel (`ForeignVehicles`), objekt grafického rozhrania (`Gui`). Za konštruktorom nasleduje definícia internej metódy `_registerShutdownHandler`, ktorá v bežiacom procese aplikácie servera registruje kus kódu, ktorý sa vykoná pri ukončení

daného procesu so signálom `INT` alebo `TERM`. Kus kódu, ktorý sa vykoná pri ukončení daného procesu je uvedený v metóde `stop`. V zásade sa zatvorí grafické rozhranie a uzavrie sa UDP socket. Ďalej nasledujú interné metódy `_initUdpSock` a `_figureOutMyIp`. V `_initUdpSock` povieme UDP socket-u, že informácie budeme prijímať z broadcast-u, a že nechceme blokovať vlákno, ak žiadna informácia nie je k dispozícii. Vo `_figureOutMyIp` zase zisťujeme, aká IP adresa je priradená k sieťovému rozhraniu podľa mena (`bat0`). Používame na to Python balíček `netifaces`. Predposlednou (internou) metódou je `_udpSockReceiver`. Z UDP socket-u prečíta prijaté JSON dáta. Následne podľa IP adresy identifikujeme, či sa prijaté dáta týkajú nášho alebo cudzieho vozidla. Ak sa týkajú nášho vozidla a sú kompletné (teda obsahujú aj informácie ako počet otáčok motora za minútu a zaradený prevodový stupeň motora), tak aktualizujeme informácie o našom vozidle pomocou metódy `fromJson`, kde za povšimnutie stojí druhý argument `True`. Následne aktualizujeme aj pohľad o našom vozidle. Ak sa jedná o cudzie vozidlo, tak sa ho podľa IP adresy pokúsime nájsť v zozname. Ak neexistuje, tak ho vytvoríme aj s danými informáciami a pridáme ho do zoznamu. Ak existuje, tak ho aktualizujeme pomocou metódy `fromJson`. Keďže sme nastavili UDP socket tak, aby neblokoval vlákno, tak musíme ošetriť aj chybu ku ktorej dochádza v prípade, že sa snažíme zo stream-u čítať aj keď v ňom nič nie je. Vtedy je vyvolaná chyba `BlockingIOError`. V takom prípade ju jednoducho vedome odignorujeme. Nakoniec prejdeme celý zoznam, či je aktuálny. Ak sa v ňom nachádzajú vozidlá, ktoré sú mimo dosah, tak sa z neho odstránia. Potom aktualizujeme pohľad cudzích vozidiel. Úplne na záver naplánujeme ďalšie spustenie tejto metódy o pár milisekúnd neskôr. Ako je zrejmé, celá táto činnosť sa odohráva vo vlákne grafického rozhrania, čo v praxi nie je zrovna najlepší nápad, pretože to môže znekvalitniť „zážitok“ používateľa z používania aplikácie napríklad spomalenou reakciou grafického rozhrania. Ideálne je tieto dve veci navzájom oddeliť. Pre účel našej aplikácie to však postačuje. Posledná metóda `start` v podstate volá jednotlivé interné metódy a nakoniec zobrazí grafické rozhranie v podobe okna. Samotné spustenie aplikácie servera sa udeje práve po zavolaní tejto metódy. A to realizujeme práve pod definíciou tejto triedy.

```
import netifaces
import signal
```

```
import socket
import sys
import time

from foreignvehicles import ForeignVehicles
from gui import Gui
from objectmaker import ObjectMaker
from tkinter import Tk
from vehicle import Vehicle

class VehicleServer:
    # Uvedená IP ("") reprezentuje priradenú IP-čku z rozsahu
    # 169.254.0.1-254 (unicast) a 169.254.0.255 (broadcast).
    # To znamená, že server počúva na dvoch IP-čkách
    IP = ""
    PORT = 20000
    ADDR = (IP, PORT)
    NETWORK_INTERFACE_NAME = "bat0"
    BUFFER_SIZE = 1024

    _ip = None

    def __init__(self, new, vehicle, udpSock,
                 foreignVehicles, gui, addr = ADDR):
        self._addr = addr
        self._new = new
        self._vehicle = vehicle
        self._udpSock = udpSock
        self._foreignVehicles = foreignVehicles
        self._gui = gui


    def _registerShutdownHandler(self):
        signal.signal(signal.SIGINT, self.stop)
        signal.signal(signal.SIGTERM, self.stop)


    def _figureOutMyIp(self, netIfName, netifaces):
        bat0 = netifaces.ifaddresses(netIfName)
        self._ip = bat0[netifaces.AF_INET][0]["addr"]


    def _initUdpSock(self):
        self._udpSock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
        self._udpSock.setblocking(False)
        self._udpSock.bind(self._addr)
```

```
    def _udpSockReceiver(self):
        try:
            bufferSize = VehicleServer.BUFFER_SIZE
            (json, (ip, port)) = self._udpSock.recvfrom(bufferSize)

            if ip == self._ip or ip == "127.0.0.1":
                if self._vehicle.fromJson(json, True) != None:
                    self._gui.updateMyVehicle()
                # Inak ignoruj nekompletné dáta o mne
            else:
                foreignVehicle = self._foreignVehicles.get(ip)

                if foreignVehicle == None:
                    foreignVehicle = self._new.foreignVehicle(json)
                    self._foreignVehicles.add(foreignVehicle)
                else:
                    foreignVehicle.fromJson(json)
        except BlockingIOError:
            pass

        self._foreignVehicles.removeAllUnreachable(time.time())
        self._gui.updateForeignVehicles()
        self._gui.invokeLater(self._udpSockReceiver)


    def start(self, netIfName = NETWORK_INTERFACE_NAME, nfs = netifaces):

        self._registerShutdownHandler()
        self._figureOutMyIp(netIfName, nfs)
        self._initUdpSock()
        self._gui.show(self._udpSockReceiver)


    def stop(self, sigNum = None, csf = None):
        try:
            self._gui.close()
        except Exception as e:
            print(e, file=sys.stderr)

        try:
            self._udpSock.close()
        except Exception as e:
            print(e, file=sys.stderr)


if __name__ == "__main__":
    new = ObjectMaker()
    vehicle = Vehicle()
```

```
udpSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
tk = Tk()
foreignVehicles = ForeignVehicles()
gui = Gui(master=tk, vehicle=vehicle,
          foreignVehicles=foreignVehicles)

server = VehicleServer(new, vehicle, udpSock,
                       foreignVehicles, gui)

try:
    server.start()
except KeyboardInterrupt:
    pass
finally:
   server.stop()
```

*Zdrojový kód 17 vehicleserver/vehicleserver.py*

### 3.2.12 Služba aplikácie klienta (súbor `lib-systemd-system/v2v-vehicleclient.service`)

Ako sme už uviedli v oddieli 3.2.6, pri spustení minipočítača bez periférnych zariadení chceme, aby minipočítač pri spustení nadviazal Bluetooth spojenie s OBD-II skenerom automaticky. V tomto štádiu vývoja sa to už netýka len nadviazania Bluetooth spojenia, ale to isté musíme zabezpečiť aj pre aplikáciu klienta a servera. Opäť využijeme systém služieb Systemd. Služba `v2v-vehicleclient.service` zabezpečuje spustenie aplikácie klienta, ktorá je definovaná podobným spôsobom ako služba `v2v-obd2rpi.service`. Implementáciu môžete vidieť v zdrojovom kóde 18. Má nastavenú cestu k PID súboru, oddiaľuje štart o 10 sekúnd (oproti absolútnemu štartu systému je to 20 sekúnd), presmerováva štandardný výstup (stdout) a chybu (stderr) do logových súborov, reštart služby vykonáva nekonečne veľakrát a medzi jednotlivými pokusmi o reštart má aj 10 sekundovú pauzu. Hlavným rozdielom oproti `v2v-obd2rpi.service` je to, že táto služba je silne závislá na službe `v2v-obd2rpi.service` namiesto skupinovej služby `multi-user.target`. Pokiaľ služba `v2v-obd2rpi.service` úspešne nebeží (teda spojenie medzi OBD-II skenerom a minipočítačom nebolo úspešne nadviazané), tak táto služba sa ani len nepokúsi o spustenie. Používa sa tu ešte silnejší stupeň závislosti `BindTo=`. Plní tú istú funkciu ako `Requires=`, avšak navyše hovorí o tom, že ak služba `v2v-`

`obd2rpi.service` zlyhá, tak automaticky sa ukončí aj táto služba. Ako sme už spomínali predtým, tak hlavným súborom aplikácie klienta je pre nás súbor `vehicleclient/vehicleclient.py`. Spustenie aplikácie klienta sa realizuje v nastavení `ExecStart=`. Na spustenie používame interpreter programovacieho jazyka `python3`. Keď už máme službu `v2v-vehicleclient.service` definovanú, tak musíme ju registrovať a aktivovať v systéme služieb Systemd cez príkazový riadok pomocou troch príkazov. Príkazom `ln -sf /home/pi/Desktop/v2v/lib-systemd-system/v2v-vehicleclient.service /lib/systemd/system/v2v-vehicleclient.service` vytvoríme v adresári `/lib/systemd/system` symbolický odkaz na súbor služby `v2v-vehicleclient.service`. Príkazom `systemctl daemon-reload` obnovíme démona služieb Systemd a príkazom `systemctl enable v2v-vehicleclient.service` následne aktivujeme službu.

```
[Unit]
BindsTo=v2v-obd2rpi.service
After=v2v-obd2rpi.service

[Service]
PIDFile=/run/v2v-vehicleclient.pid
ExecStartPre=/bin/sleep 10
ExecStart=/usr/bin/python3 /home/pi/Desktop/v2v/vehicleclient/vehicleclient.py
StandardOutput=append:/home/pi/Desktop/v2v/lib-systemd-system/log/v2v-vehicleclient.log
StandardError=append:/home/pi/Desktop/v2v/lib-systemd-system/log/v2v-vehicleclient.err
Restart=always
RestartSec=10s

[Install]
WantedBy=v2v-obd2rpi.service
```

*Zdrojový kód 18 Súbor lib-systemd-system/v2v-vehicleclient.py*

### 3.2.13 Služba aplikácie servera (súbor `lib-systemd-system/v2v-vehicleserver.service`)

Poslednou službou je služba zabezpečujúca spustenie aplikácie servera. Implementáciu môžete vidieť v zdrojovom kóde 19. Používa identickú definíciu ako

služba `v2v-vehicleclient.service` vrátane silnej závilosti `BindTo=`. Hlavný súbor aplikácie servera tu však predstavuje súbor `vehicleserver/vehicleserver.py`. Rozdielom je aj to, že táto služba je silne závislá na službe `v2v-vehicleclient.service` a skupinovej službe `graphical.target`. Opäť, ak tieto služby nebežia, tak sa nespustí ani táto. A naopak, ak jedna z nich zlyhá, tak sa ukončí aj táto. Závislosť na skupinovú službu `graphical.target` tu používame preto, že hlavnou úlohou aplikácie servera je zobrazovať informácie vodičovi. Ak však zobrazovacia jednotka nie je k dispozícii, tak nemá význam túto službu ani len spúšťať. V definícii tejto služby sú navyše nastavenia `Environment=`. Slúžia práve na nastavenie grafického prostredia. Bez týchto nastavení, by sa nám samotná aplikácia nezobrazila na zobrazovacej jednotke. Keď už máme službu `v2v-vehicleserver.service` definovanú, tak musíme ju registrovať a aktivovať v systéme služieb Systemd cez príkazový riadok pomocou troch príkazov. Príkazom `ln -sf /home/pi/Desktop/v2v/lib-systemd-system/v2v-vehicleserver.service /lib/systemd/system/v2v-vehicleserver.service` vytvoríme v adresári `/lib/systemd/system` symbolický odkaz na súbor služby `v2v-vehicleserver.service`. Príkazom `systemctl daemon-reload` obnovíme démona služieb Systemd a príkazom `systemctl enable v2v-vehicleserver.service` aktivujeme danú službu.

```
[Unit]
BindTo=v2v-vehicleclient.service graphical.target
After=v2v-vehicleclient.service graphical.target

[Service]
Environment="DISPLAY=:0"
Environment="XAUTHORITY=/home/pi/.Xauthority"
PIDFile=/run/v2v-vehicleserver.pid
ExecStartPre=/bin/sleep 10
ExecStart=/usr/bin/python3 /home/pi/Desktop/v2v/vehicleserver/vehicleserver.py
StandardOutput=append:/home/pi/Desktop/v2v/lib-systemd-system/log/v2v-vehicleserver.log
StandardError=append:/home/pi/Desktop/v2v/lib-systemd-system/log/v2v-vehicleserver.err
Restart=always
RestartSec=10s

[Install]
WantedBy=v2v-vehicleclient.service graphical.target
```

*Zdrojový kód 19 Súbor lib-systemd-system/v2v-vehicleserver.service*

### 3.2.14 Automatizácia inštalácie pomocou skriptu (súbor `install`)

Problém, s akým sme sa od začiatku vývoja stretli, bolo skoordinovanie programovania s testovaním vo vozidle. Keď sme niečo naprogramovali v programovacom jazyku Python, otestovať sa to dalo len vo vozidle. K dispozícii sme síce mali malú zobrazovaciu jednotku, ale písať niečo zmysluplné na nej neprichádzalo do úvahy. Na prípadné malé opravy kódu alebo pozeranie logov to však stačilo. Okrem toho nám vo vozidle chýbal internet a samotné bezdrôtové sieťové rozhranie už bolo nakonfigurované na použitie s BATMAN-ADV. Internet bol potrebný hlavne kvôli inštalovaniu rôznych balíčkov, či už softvérových do operačného systému alebo priamo do programovacieho jazyka Python. Internet sme v minipočítačoch zabezpečovali cez LAN kábel. Celý proces programovania vyzeral asi nasledovne:

- vnútri v dome zapnúť minipočítač a pripojiť potrebné periférne zariadenia (myš, klávesnica, LAN kábel, HDMI kábel k veľkému monitoru),
- naprogramovať niečo zmysluplné, nainštalovať chýbajúce softvérové alebo Python balíčky a vypnúť minipočítač,
- presunúť sa do vozidla, naštartovať vozidlo, zapnúť minipočítač a pripojiť potrebné perfirérne zariadenia (myš, klávesnica, HDMI kábel k malému displeju),
- otestovať naprogramovanú funkcionalitu, pričom človek zistí, že to nefunguje, hľadať riešenie (či už na internete alebo v hlave), skúsiť opraviť kód priamo vo vozidle ak to je možné, skúsiť to znova,
- vypnúť motor vozidla, presunúť sa naspäť do domu.

Celý proces programovania sa opakoval dookola a niekoľko dní za sebou. Takéto presúvania medzi vozidlom a domom, zapínanie a vypínanie minipočítača, štartovanie a vypínanie motora vozidla stálo veľa času a bolo to dosť otravné. Viackrát sme museli dobre premyslieť čo chceme spraviť a ako to chceme spraviť, aby sme túto činnosť minimalizovali. Potom sa k tomu pridalo druhé vozidlo. Nebolo nič nezvyčajné, že nám na ulici stáli dve naštartované vozidlá s jedným vodičom, no niekedy aj bez dozoru. Keď už do toho prišlo aj to druhé vozidlo, tak na obidvoch vozidlách musel byť spustený ten istý kód (vzájomne kompatibilný). Po každej zmene zrkadliť pamäťovú microSD kartu nebolo riešenie. Viaceré nastavenia by sa aj tak museli meniť a bolo by

to aj časovo náročné. Na udržiavanie aktuálnosti oboch minipočítačov bol potrebný centrálny repozitár. Na tom nám poslúžil GitHub. Keď už bol kód funkčný a úspešne otestovaný na dvoch vozidlách, tak sme to chceli skúsiť aj na troch. V prípade, že to chceme skúsiť aj na viac ako troch, tak sme potrebovali celý proces inštalácie automatizovať. Inštaláciu operačného systému Raspbian, jeho konfiguráciu a stiahnutie repozitára z GitHub-u síce musíme spraviť manuálne, ale všetko ostatné už vieme spraviť cez inštalačný skript. Jeho implementáciu môžete vidieť v zdrojovom kóde 20. Jedná sa o klasický Bash skript. Súboru treba nastaviť aj oprávnenie na exekúciu v príkazovom riadku. Najskôr zmeníme aktuálny pracovný adresár na ten, kde máme celý projekt uložený. Nemôžme ho uložiť nikde inde, iba na plochu používateľa `pi` do adresára `v2v`. Celý projekt uvažuje iba s touto lokáciou. Nainštalujeme balíčky potrebné pre projekt z pohľadu operačného systému a potom aj z pohľadu programovacieho jazyka Python. Nastavenie sieťového rozhrania spočíva v premenovaní súboru `bat0.example` na `bat0`, v oprave IP adresy, zmenení vlastníka na používateľa `pi` a vytvorení symbolických odkazov. Ďalej musíme registrovať služby. Tiež pre ne vytvoríme symbolické odkazy, vytvoríme adresár kam sa budú ukladať logy z jednotlivých služieb a zmeníme vlastníka na používateľa `pi`. Potom už len aktivujeme služby. Ďalším krokom je premenovať súbor `vehicleinfo.txt.example` na `vehicleinfo.txt` a zmeniť vlastníka na používateľa `pi`. V ňom musíme zmeniť už spomínané informácie z oddielu 3.2.3. Posledný krok je aktivácia BATMAN-ADV. To spočíva v doplnení troch rôznych príkazov do troch rôznych súborov. Registrujeme BATMAN-ADV ako modul, sieťovému rozhraniu `wlan0` zakážeme DHCP a príkaz `batman` (rozoberaný v pododdieli 3.2.7.1) vykonáme pri spustení interaktívneho shell-u. Toto posledné riešenie asi nie je úplne ideálne, ale funguje. Lepšie by bolo tento príkaz vykonať po inicializácii sieťového modulu. Na záver sa ešte vypíše, čo je potrebné spraviť. To sme si ale priebežne napísali.

```
#!/bin/bash

cd /home/pi/Desktop/v2v

# Nainštalovať OS závislosti
apt-get update
apt-get install -y batctl bluez expect python3.7
```

```
# Nainštalovať Python závislosti
pip3 install netifaces https://github.com/brendan-w/python-OBD/archive/master.zip

# Sieťové rozhranie
cp etc-network-interfaces.d/bat0.example etc-network-interfaces.d/bat0
chown pi:pi etc-network-interfaces.d/bat0
ln -sf "$(pwd)/etc-network/interfaces"         /etc/network/interfaces
ln -sf "$(pwd)/etc-network-interfaces.d/bat0"  /etc/network/interfaces.d/bat0
ln -sf "$(pwd)/etc-network-interfaces.d/wlan0" /etc/network/interfaces.d/wlan0

# Služby
ln -sf "$(pwd)/lib-systemd-system/v2v-obd2rpi.service"       /lib/systemd/system/v2v-obd2rpi.service
ln -sf "$(pwd)/lib-systemd-system/v2v-vehicleclient.service" /lib/systemd/system/v2v-vehicleclient.service
ln -sf "$(pwd)/lib-systemd-system/v2v-vehicleserver.service" /lib/systemd/system/v2v-vehicleserver.service
mkdir -p lib-systemd-system/log
chown pi:pi lib-systemd-system/log
systemctl daemon-reload
systemctl enable v2v-obd2rpi.service
systemctl enable v2v-vehicleclient.service
systemctl enable v2v-vehicleserver.service

# vehicleinfo.txt
cp vehicleinfo.txt.example vehicleinfo.txt
chown pi:pi vehicleinfo.txt

# Aktivovať BATMAN-ADV
echo "" >> /etc/modules
echo "batman-adv" >> /etc/modules
echo "" >> /etc/modules
echo "" >> /etc/dhcpcd.conf
echo "denyinterfaces wlan0" >> /etc/dhcpcd.conf
echo "" >> /etc/dhcpcd.conf
echo "" >> /home/pi/.bashrc
echo "$(pwd)/batman/batman" >> /home/pi/.bashrc
echo "" >> /home/pi/.bashrc

# Čo ešte treba urobiť
echo ""
echo ""
echo ""
echo "Úlohy:"
echo "------"
echo "1. V súbore ./etc-network-interfaces.d/bat0 oprav IP adresu"
echo "2. V súbore ./vehicleinfo.txt oprav údaje o aute"
```

*Zdrojový kód 20 Súbor install*

### 3.2.15 Kompletný zdrojový kód

Kompletný zdrojový kód máme uložený aj v našom GitHub repozitáre na URL adrese `https://github.com/david-bojnansky/diploma-v2v`.

## 3.3 PRAKTICKÉ TESTOVANIE A ZHODNOTENIE VÝSLEDKOV

V2V systém sa nám podarilo úspešne doprogramovať a ostáva ho už len spustiť a otestovať v praxi. Dosiahnuté výsledky a nadobudnuté poznatky sa následne pokúsime stručne zhodnotiť.

### 3.3.1 Spustenie systému vo vozidle

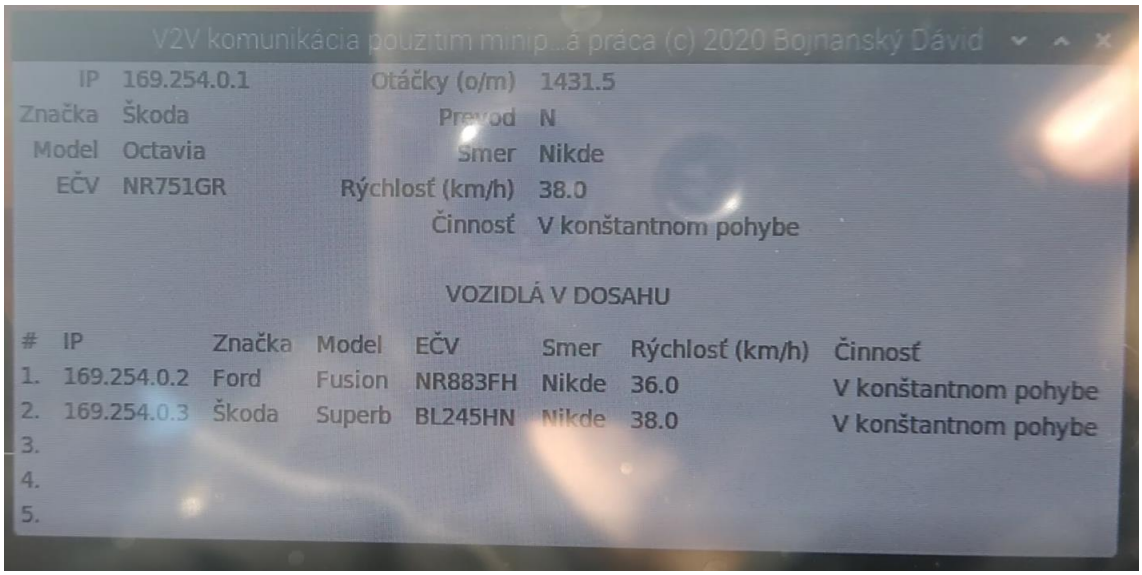
Keď sa už nachádzame vo vozidle a máme všetko pripravené, tak si musíme položiť otázku, či už OBD-II skener máme spárovaný s minipočítačom Raspberry Pi alebo nie (určite ho nemáme pri absolútne prvom spustení minipočítača vo vozidle). Ak nie, tak k minipočítaču musíme pripojiť aj periférne zariadenia ako displej, klávesnicu a myš. Do OBD-II konektora zasunieme OBD-II skener a naštartujeme vozidlo. Minipočítač zapojíme do microUSB napájacieho adaptéra zasunutého v zapaľovaní vo vozidle. Keď sa načíta operačný systém, tak otvoríme príkazový riadok a spustíme príkaz `/home/pi/Desktop/v2v/obd2rpi/pairwithobd` (za prepokladu, že MAC adresu skenera a párovací PIN kód máme v súbore `vehicleinfo.txt` správne zadané). Párovací proces by sa mal zahájiť a úspešne ukončiť po približne 30 sekundách. Následne reštartujeme minipočítač. Do minúty od načítania operačného systému by sa malo zobraziť grafické rozhranie s priebežne aktualizujúcimi sa informáciami o našom vozidle. Ak sa tieto informácie priebežne aktualizujú, tak všetko je v poriadku a minipočítač môžeme vypnúť. Potom môžeme odpojiť všetky periférne zariadenia. Minipočítač si položíme na palubovú dosku. Odteraz už môžeme spúšťať minipočítač bez periférnych zariadení a do minúty by sa mal V2V systém spustiť a začať vysielať informácie o našom vozidle do okolia. Tento postup zopakujeme pre aspoň jedno ďalšie vozidlo. Pokiaľ chceme v niektorom vozidle sledovať informácie o našom vozidle a o vozidlách okolo nás v reálnom čase, tak k minipočítaču pripojíme displej. Po spustení minipočítača by malo nabehnúť grafické rozhranie s priebežne aktualizujúcimi sa informáciami o našom ako aj o ostatných vozidlách okolo nás.

### 3.3.2 Všeobecná funkčnosť systému

To čo sme si naplánovali v návrhu (podkapitola 3.1), tak to sa nám aj podarilo zrealizovať. Výsledok je možné vidieť na obrázku 10. Informácie, ktoré sú zobrazené na obrázku, sú zachytené z testovacej jazdy s dvomi ďalšími vozidlami. V hornej časti okna sa zobrazujú informácie o našom vozidle. V jeho ľavej časti sú IP adresa sieťového rozhrania, značka vozidla, model vozidla a evidenčné číslo vozidla. Ako sme už spomínali, tak tieto informácie nevieme získať z rozhrania OBD-II, a preto sú fixne definované v súbore. V strednej časti sa nachádza počet otáčok motora za minútu, zaradený prevodový stupeň motora, smer jazdy vozidla, rýchlosť vozidla a činnosť vozidla. Z týchto informácií fungujú len tri, a to počet otáčok motora za minútu, rýchlosť vozidla a činnosť vozidla. Prvé dve sú získané priamo z rozhrania OBD-II a činnosť vozidla je menená na základe rýchlosti vozidla. Naopak, nefunguje zaradený prevodový stupeň motora a smer jazdy vozidla. Smer jazdy vozidla sme chceli založiť na zaradenom prevodovom stupni motora. Mali sa zobrazovať hodnoty ako „nikde, dopredu alebo dozadu." Python balíček, ktorý sme použili na komunikáciu s rozhraním OBD-II, nemal definovaný príkaz pre získanie zaradeného prevodového stupňa motora. Preto sme vytvorili podľa oficiálnej dokumentácie tohto balíčka vlastný OBD-II príkaz, ktorý mal takúto informáciu z rozhrania OBD-II získať. Žiaľ, neustále nám vracia hodnotu 0. Snažili sme sa dopátrať k možným riešeniam, ale nepodarilo sa nám to. Skúšali sme aj kvalitnejšie OBD-II skenery, avšak bez výraznejšieho úspechu. Je pravda, že sme mali k dispozícií len vozidlá s manuálnou prevodovkou. Pri automatickej by bol výsledok možno iný. V spodnej časti okna sa zobrazuje tabuľka s informáciami o vozidlách okolo nás, ktoré sú v dosahu. Ak sa vozidlo stratí z dosahu, tak sa vozidlo z tabuľky vymaže. Informácie, ktoré o týchto vozidlách zobrazujeme, sú IP adresa sieťového rozhrania, značka vozidla, model vozidla, evidenčné číslo vozidla, smer jazdy vozidla, rýchlosť vozidla a činnosť vozidla. So smerom jazdy vozidla tu máme ten istý problém. Je dôležité poznamenať, že žiadne informácie o cudzích vozidlách nespracúvame v našom vozidle, ale rovno ich tak prijímame. To znamená, že napríklad aktuálnu činnosť cuzdieho vozidla nevypočítavame, ale rovno ju tak príjmame. Pri testovaní sme odskúšali aj to, či v prípade, ak vytiahneme OBD-II skener z konektora (čím systém znefunkčníme) a potom ho zapojíme naspäť, sa systém sám reštartuje a spustí. Dospeli sme k záveru, že toto nám funguje. Platí to aj v prípade zavretia okna aplikácie servera. Týmto sme sa snažili docieliť zjednodušený kiosk mód.

Celkovo môžeme teda konštatovať, že náš systém v praxi funguje. Priebeh testovania opísaných vlastností sme zachytili aj na niekoľko video záznamov:

- Vozidlá idúce za sebou I. (`https://youtu.be/kq7yeCqjYpg`)
    - V tomto videu spomíname, že vozidlo idúce priamo pred nami sa často stráca z nášho dosahu na rozdiel od toho pred ním. Po testovaní sme však zistili, že v týchto vozidlách boli nechtiac zamenené minipočítače. Z toho vyplýva, že vozidlo priamo pred nami sa menej často strácalo z dosahu ako to pred ním.
- Vozidlá idúce za sebou II. (`https://youtu.be/ZFmHJJL8_uo`)
- Automatický reštart systému (`https://youtu.be/rrhvNowX0fs`)

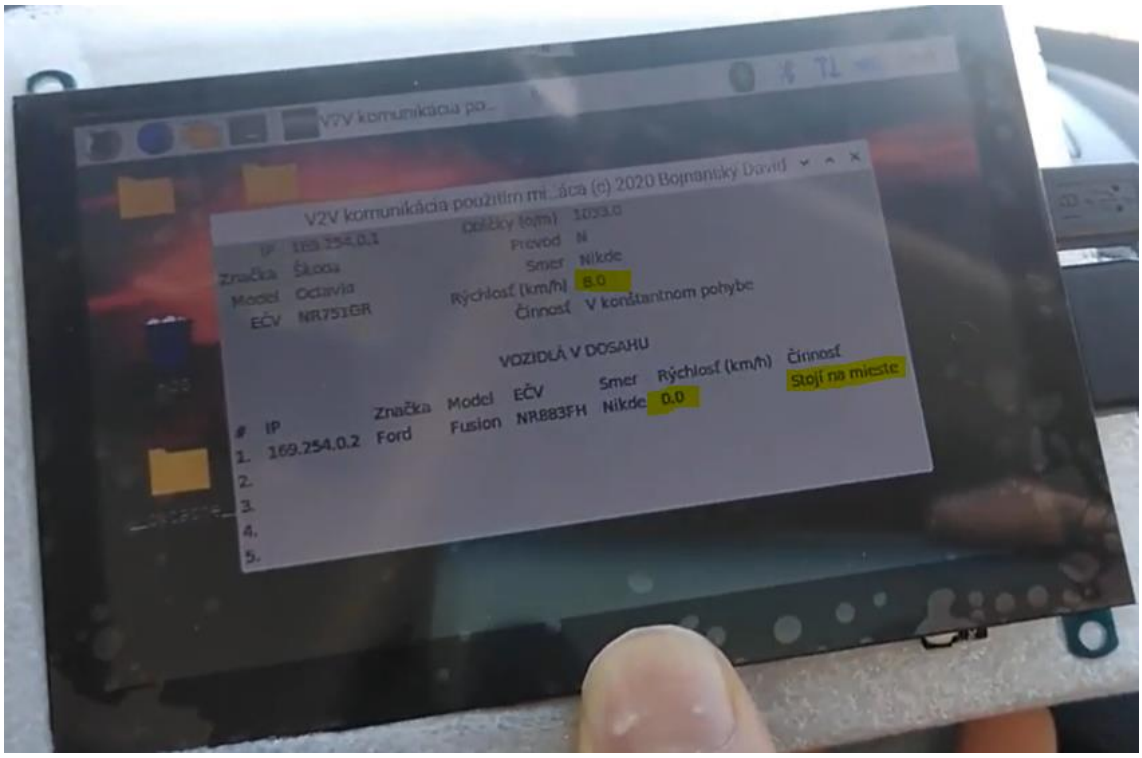



*Obrázok 10 Zobrazované informácie na displeji minipočítača a ich funkčnosť*

### 3.3.3 Dosah signálu

Pri testovacej jazde, kde sme šli tri vozidlá za sebou s odstupom približne 10 metrov, sa stávalo, že vozidlá pred nami sa nám na chvíľu stratili z dosahu. V niektorých prípadoch však spojenie fungovalo aj na viac ako 10 metrov. Všetky tri vozidlá mali minipočítač položený na palubnej doske. Myslíme si však, že ak by sme minipočítače umiestnili napríklad na strechu vozidiel, tak by sme dostali stabilnejší signál a aj väčší dosah. Nemali sme však dostatočne dlhé napájacie káble. Spravili sme však aj test, kde jedno vozidlo stálo na mieste a druhé sa pomaly pohybovalo smerom

od neho. Pri tomto teste sme zistili, že signál bol stabilný približne do 50 metrov. Pri viac ako 50 metrov sa spojenie už nedokázalo nadviazať. Jedenkrát sa stalo, že spojenie sa na chvíľu prerušilo a opäť sa nadviazalo. Stalo sa to približne po 20 metroch. Toto testovanie máme zdokumentované vo videu „Dosah signálu“ (`https://youtu.be/NyofHGUne5c`) a ukážku môžete vidieť na obrázku 11.

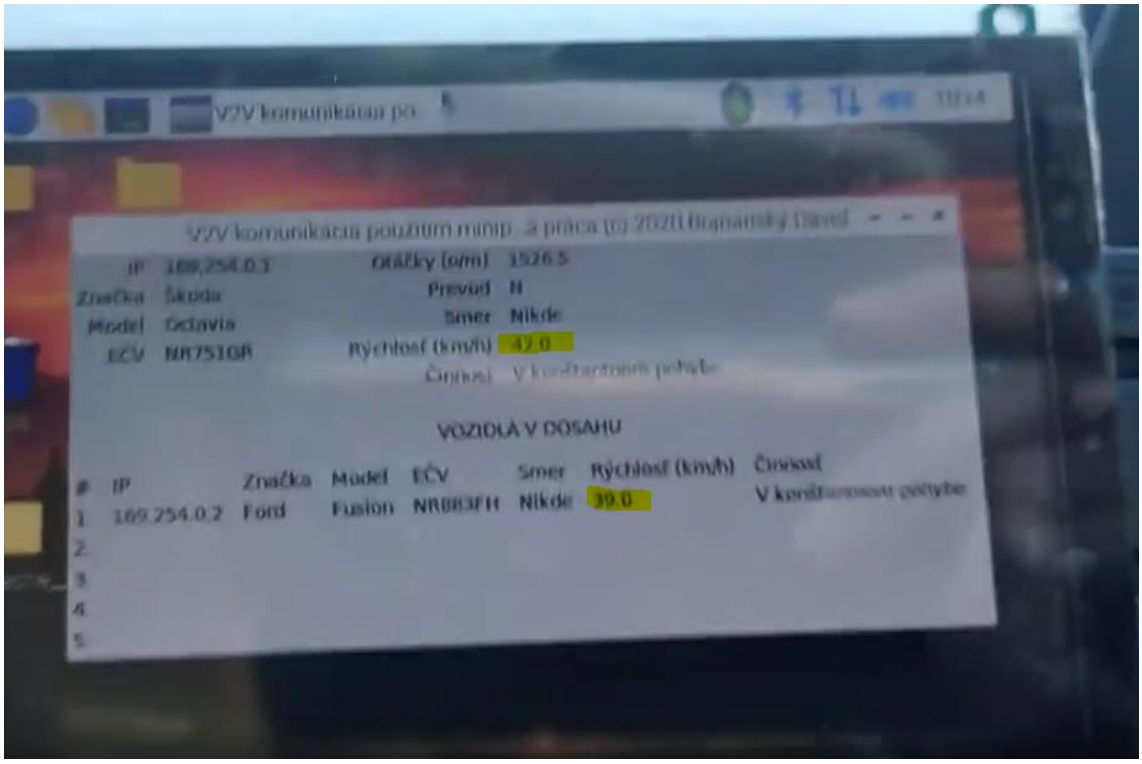



*Obrázok 11 Testovanie dosahu signálu*

### 3.3.4 Míňajúce sa vozidlá

Otestovali sme aj prípad, keď dve vozidlá idú oproti sebe, minú sa a pokračujú ďalej v smere svojej jazdy. Týmto testom sme chceli hlavne zistiť, či tieto vozidlá dokážu medzi sebou nadviazať spojenie a vymeniť si informácie. Testovali sme to pri rýchlosti 40 km/h oboch vozidiel. V bode, keď prechádzali okolo seba, bola teda celková rýchlosť 80 km/h. Výsledkom tohto testu je, že vozidlá dokázali medzi sebou nadviazať spojenie a vymeniť si informácie. Tento test máme zdokumentovaný aj vo forme videa „Vozidlá, ktoré sa míňajú“ (`https://youtu.be/ac9dYk9JT6I`) a ukážky môžete vidieť na obrázkoch 12 a 13.

*Obrázok 12 Testovanie nadviazania spojenia míňajúcich sa vozidiel*




*Obrázok 13 Celková rýchlosť míňajúcich sa vozidiel je 80 km/h*

# ZÁVER

Myšlienka V2V systému je skutočne skvelá. Avšak jeho realizácia je oveľa zložitejšia než si väčšina ľudí uvedomuje. Zahŕňa mnoho technologických aspektov, mnoho vládnych regulácií, ale aj menej obľúbenú spoluprácu výrobcov vozidiel. Ako sme v prvej kapitole uviedli, tak hlavná myšlienka V2V systému bola rozšírená o komunikáciu so všetkým okolo nás (V2X), kde patrí dopravná infraštruktúra, chodci, internetová sieť a podobne.

Hlavným cieľom našej diplomovej práce bolo vytvoriť funkčný V2V systém, otestovať ho v praxi a vyhodnotiť dosiahnuté výsledky. Dovolíme si tvrdiť, že to sa nám podarilo. Podarilo sa nám na začiatku navrhnúť V2V systém po hardvérovej aj softvérovej stránke a krok za krokom sme tento návrh následne zrealizovali. Hoci sme narazili na viaceré problémy, tie ktoré bolo z pohľadu celkovej funkčnosti V2V systému dôležité vyriešiť, tak tie sa nám aj vyriešiť nakoniec podarilo. Vytvorený V2V systém sme následne otestovali a získané výsledky sme vyhodnotili. Zjednodušene môžeme konštatovať, že systém nám funguje okrem jednej veci. Tá, ale nie je kritická pre funkčnosť systému ako takého. Konkrétne sa nám cez rozhranie OBD-II nepodarilo získať zaradený prevodový stupeň motora, a preto sme nemohli jednoznačne určiť smer jazdy. Nie sme si úplne istý, ale táto informácia je dostupná zrejme iba vo vozidlách s automatickou prevodovkou a my sme mali k dispozícii len s manuálnou.

Na záver ešte pridávame tipy ako by sa dal náš systém vylepšiť pre zlepšenie „zážitku“ vodiča z jazdy:

- GPS – pomocou neho by sme vedeli určiť smer jazdy vozidla, zistiť ako ďaleko je od nás iné vozidlo a podobne,
- pípací zvuk signalizujúci brzdenie vozidla pred nami (vozidlo pred nami by sa určovalo na základe porovnania našich a jeho GPS súradníc),
- automatizácia počiatočného spárovania s OBD-II skenerom pri spustení minipočítača (nemyslíme automatizáciu nadviazania spojenia s OBD-II skenerom, pretože to máme spravené).

# ZOZNAM BIBLIOGRAFICKÝCH ODKAZOV


KARPAT, M. 2007. Komunikácia medzi autami - vyskúšali sme technológiu budúcnosti. [online]. [cit. 2019-11-30]. Dostupné na internete: <https://auto.sme.sk/c/3469935/komunikacia-medzi-autami-vyskusali-sme-technologiu-buducnosti.html>.

THEPIHUT.COM. 2016. Raspberry Pi 3 Model B. [online]. [cit. 2019-11-30]. Dostupné na internete: <https://thepihut.com/products/raspberry-pi-3-model-b>.

THEPIHUT.COM. 2019. Raspberry Pi 4 Model B. [online]. [cit. 2019-11-30]. Dostupné na internete: <https://thepihut.com/products/raspberry-pi-4-model-b>.

NHTSA.GOV. 2019. About NHTSA. [online]. [cit. 2019-12-01]. Dostupné na internete: <https://www.nhtsa.gov/about-nhtsa>.

NHTSA.GOV. 2019. Vehicle-to-Vehicle Communication. [online]. [cit. 2019-12-01]. Dostupné na internete: <https://www.nhtsa.gov/technology-innovation/vehicle-vehicle-communication>.

HALL-GEISLER, K. 2017. All new cars could have V2V tech by 2023. [online]. [cit. 2019-12-01]. Dostupné na internete: <https://techcrunch.com/2017/02/02/all-new-cars-could-have-v2v-tech-by-2023/>.

STANDARDS.IEEE.ORG. 2010. IEEE 802.11p-2010 - IEEE Standard for Information technology-- Local and metropolitan area networks-- Specific requirements-- Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments. [online]. [cit. 2019-12-01]. Dostupné na internete: <https://standards.ieee.org/standard/802_11p-2010.html>.

DORMEHL, L a EDELSTEIN, S. 2019. Sit back, relax, and enjoy a ride through the history of self-driving cars. [online]. [cit. 2019-12-06]. Dostupné na internete: <https://www.digitaltrends.com/cars/history-of-self-driving-cars-milestones/>.

ARENA, F. a PAU, G. 2019. An Overview of Vehicular Communications. [online]. [cit. 2019-12-06]. Dostupné na internete: <https://www.mdpi.com/1999-5903/11/2/27/pdf>.

MISHRA, S. a iní. 2015. System on Chip Interfaces for Low Power Design. 406 s. ISBN 978-0128017906.

OBDII.COM. 2011. OBD-II Background. [online]. [cit. 2019-12-06]. Dostupné na internete: <http://www.obdii.com/background.html>.

TECHOPEDIA.COM. 2019. What is ZigBee?. [online]. [cit. 2019-12-07]. Dostupné na internete: <https://www.techopedia.com/definition/4390/zigbee>.

ELECTRONICS-NOTES.COM. 2019. What is ZigBee Technology. [online]. [cit. 2019-12-07]. Dostupné na internete: <https://www.electronics-notes.com/articles/connectivity/zigbee/what-is-zigbee-technology-tutorial.php>.

TECHOPEDIA.COM. 2019. What is Bluetooth?. [online]. [cit. 2019-12-07]. Dostupné na internete: <https://www.techopedia.com/definition/26198/bluetooth>.

10NAJS.SK. 2018. Čo je to Bluetooth? Aký má dosah? Bluetooth 5 vs Bluetooth 4. [online]. [cit. 2019-12-07]. Dostupné na internete: <https://10najs.sk/co-je-to-bluetooth/>.

RASPBERRYPI.ORG. 2019. What is Raspberry Pi?. [online]. [cit. 2019-12-07]. Dostupné na internete: <https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/>.

JIANG, D. a DELGROSSI, L. 2008. IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments. [online]. [cit. 2019-12-09]. Dostupné na internete: <https://www.researchgate.net/publication/224314431_IEEE_80211p_Towards_an_International_Standard_for_Wireless_Access_in_Vehicular_Environments>.

BEAL, V. 2019. 802.11 IEEE wireless LAN standards. [online]. [cit. 2019-12-09]. Dostupné na internete: <https://www.webopedia.com/TERM/8/802_11.html>.

BLANK, E. 2016. Z Wave Vs ZigBee: Which Is Better For Your Smart Home?. [online]. [cit. 2019-12-26]. Dostupné na internete: <https://thesmartcave.com/z-wave-vs-zigbee-home-automation/>.

ABDELGADER, A. a LENAN, W. 2014. The Physical Layer of the IEEE 802.11p WAVE Communication Standard: The Specifications and Challenges. [online]. [cit. 2019-12-26]. Dostupné na internete: <https://www.researchgate.net/publication/279474688_The_Physical_Layer_of_the_IEEE_80211_p_WAVE_Communication_Standard_The_Specifications_and_Challenges>.

VAMOSI, R. 2017. New V2V communication could give hackers a free ride. [online]. [cit. 2019-12-26]. Dostupné na internete: <https://www.synopsys.com/blogs/software-security/v2v-communication-hackers/>.

GRAHAM, K. 2017. Vehicle-to-vehicle communication tech going nowhere. [online]. [cit. 2019-12-28]. Dostupné na internete: <http://www.digitaljournal.com/tech-and-science/technology/vehicle-to-vehicle-communication-tech-going-nowhere/article/503010>.

KOON, J. 2019. Will Vehicle-to-Vehicle Communication Ever Take Off?. [online]. [cit. 2019-12-28]. Dostupné na internete: <https://new.engineering.com/story/will-vehicle-to-vehicle-communication-ever-take-off>.

JAIN, G. 2018. Talking Cars: A Survey of Protocols for Connected Vehicle Communication. [online]. [cit. 2019-12-28]. Dostupné na internete: <https://wiprodigital.com/2018/06/20/talking-cars-a-survey-of-protocols-for-connected-vehicle-communication/>.

OBDSOL.COM. 2020. OBD MODES. [online]. [cit. 2020-02-06]. Dostupné na internete: <https://www.obdsol.com/knowledgebase/obd-software-development/obd-services-modes/>.

KNÍŽEK, J. 2016. Nebojte se systemd: co to je a co umí?. [online]. [cit. 2020-02-19]. Dostupné na internete: <https://www.root.cz/clanky/nebojte-se-systemd-co-to-je-a-co-umi/>.

EBRAHIM, M. 2017. Expect Command And How To Automate Shell Scripts Like Magic. [online]. [cit. 2020-02-20]. Dostupné na internete: <https://likegeeks.com/expect-command/>.

BLUEZ.ORG. 2020. Common questions. [online]. [cit. 2020-02-20]. Dostupné na internete: <http://www.bluez.org/faq/common/>.

KERNEL.ORG. 2020. batman-adv – The Linux Kernel documentation. [online]. [cit. 2020-02-28]. Dostupné na internete: <https://www.kernel.org/doc/html/v4.15/networking/batman-adv.html>.

# ZOZNAM PRÍLOH

Príloha A – CD nosič

- Príloha A.1 – Diplomová práca vo formáte PDF
- Príloha A.2 – Zdrojový kód