

Vývoj samoříditelné platformy

Self-driving Platform Development

Filip Peterek

Bakalářská práce

Vedoucí práce: Ing. Jan Gaura, Ph.D.

Ostrava, 2021

Abstrakt

Tato práce se zabývá vývojem softwaru pro samořiditelnou platformu vznikající na VŠB-TUO. Po dokončení by software měl být schopen na základě vstupů ze softwaru pro analýzu obrazu, GPS, LiDARu a řídící jednotky vozidla bezpečně navigovat autonomní platformu univerzitním kampusem.

Jádro řídícího softwaru je rozděleno do dvou komponent. První komponenta zajišťuje komunikaci s řídící jednotkou vozidla pomocí sběrnice CAN [3], na starosti má ovládání rychlosti vozidla a natáčení kol. Druhá komponenta má na starosti plánování cesty na základě výše vyjmenovaných vstupů. První komponentě poté zadá pouze požadovanou rychlosť vozidla a natáčení kol, samotné posílání CAN zpráv však nemusí řešit. Komponenty mezi sebou komunikují pomocí protokolu TCP [23]. K testování ovládacího softwaru lze využít simulátor, který dokáže nasimulovat chování první komponenty. Pro řízení vozidla jsou v hojně míře využívány PID regulátory [1].

V době odevzdání této práce ještě není software plně dokončen, jelikož se jedná o výzkumný projekt, který svým rozsahem přesahuje bakalářskou práci, a vývoj platformy bude dále pokračovat.

Klíčová slova

plánování cesty, client-server komunikace, sběrnice CAN, PID regulátor, GPS

Abstract

This thesis is focused on the development of software for a self-driving platform, which is being developed by the Technical University of Ostrava. Upon completion, the software should be capable of safely navigating an autonomous vehicle through the university campus, taking inputs from image analysis software, GPS, LiDAR, and the vehicle's own controller unit.

The core of the path planning software is split into two components. The first component handles communication with the vehicle's controller unit via a CAN bus [3]. The second component then does all the path planning, taking into account the aforementioned inputs. The second component then asks the first component to maintain a certain speed or steer the vehicle, however, it doesn't have to handle CAN messages, including driving inputs or control checksums, by itself. The two components communicate using the Transmission Control Protocol [23]. The path planning software can be tested using a simulator, capable of mocking the CAN handling software, along with the entire vehicle. The simulator honors the TCP API, however, the vehicle physics are simplified. PID controllers are used in multiple parts of the software [1].

At the time of publishing this paper, the software is still unfinished, as the self-driving platform is an academic research project, which outspans this bachelor's thesis in scope, and the development of the platform is set to continue.

Keywords

autonomous driving, path planning, client-server communication, CAN bus, PID controller, GPS

Poděkování

Chtěl bych poděkovat vedoucímu mé práce, Ing. Janu Gaurovi, PhD., nejen za pomoc s tvorbou práce a návrhem softwaru, ale především za možnost se projevit a pracovat na skutečném a zajímavém projektu. Dále chci poděkovat svým kolegům z práce, kteří mě naučili spoustu věcí, na které v akademickém prostředí není čas nebo příležitost.

Obsah

Seznam použitých symbolů a zkratek	7
Seznam obrázků	8
Seznam tabulek	9
1 Úvod	10
2 Architektura řídícího softwaru	11
3 Server a CAN komunikace	12
3.1 Sběrnice CAN	12
3.2 TCP server a jeho metody	13
3.3 Regulace PI regulátorem	15
3.4 Python implementace	16
4 Simulátor vozidla	18
5 Klient a ovládací software	19
5.1 Definice cílů cesty vozidla	20
5.2 Vizualizér	20
5.3 REST API	21
5.4 Python implementace	22
6 Testované způsoby řízení vozidla	25
6.1 Řízení vozidla pomocí GPS	25
6.2 Řízení vozidla pomocí detekce grafických kódů	35
6.3 Řízení vozidla kombinací vstupů z GPS a kamery a za využití mapových podkladů .	37
7 Práce s mapovými podklady	38
7.1 Stahování mapových podkladů a jejich zpracování	38

7.2	Vyhledání nejbližší silnice na mapě	39
7.3	Plánování cesty za využití mapových podkladů	40
8	Webová aplikace	42
8.1	Implementace frontendu webové aplikace	42
8.2	Implementace backendu webové aplikace	43
9	Budoucí vývoj vozidla	45
9.1	Vstup z LiDARu	45
9.2	Chytřejší práce s mapovými podklady	45
9.3	Metriky, chytřejší logování, monitorování	46
10	Závěr	47
Zdroje		48
Přílohy		52
A	Výsledky měření pomocí GPS	53

Seznam použitých zkratек a symbolů

CAN	– Controller Area Network
PID	– Proporcionální-integrační-derivační
TCP	– Transmission Control Protocol
UDP	– User Datagram Protocol
GPS	– Global Positioning System
API	– Application Programming Interface
XML	– Extensible Markup Language
JSON	– JavaScript Object Notation
YAML	– YAML Ain’t Markup Language
HTML	– HyperText Markup Language
TX	– Transmitter
RX	– Receiver
RPC	– Remote Procedure Call
REST	– Representational State Transfer
OSM	– OpenStreetMap
URL	– Uniform Resource Locator
BFS	– Breadth-First Search
ELK	– Elasticsearch-Logstash-Kibana
DSL	– Domain Specific Language

Seznam obrázků

6.1	Rovná cesta středem ulice zaznamenaná pomocí GPS (Zdroj grafických podkladů: Seznam.cz, a.s.)	29
6.2	Krátká jízda osobním automobilem zaznamenaná pomocí GPS (Zdroj grafických podkladů: Seznam.cz, a.s.)	30
7.1	Nejkratší cesta nalezená za využití algoritmu BFS (Zdroj mapových podkladů: Seznam.cz, a.s., zdroj grafiky vozidla viz [43])	41
A.1	Krátká chůze zaznamenaná pomocí GPS (Zdroj grafických podkladů: Seznam.cz, a.s.)	54
A.2	Delší chůze, vždy po chodníku nebo silnici, zaznamenaná pomocí GPS (Zdroj grafických podkladů: Seznam.cz, a.s.)	55
A.3	Rovná chůze středem cesty zaznamenaná pomocí GPS (Zdroj grafických podkladů: Seznam.cz, a.s.)	56
A.4	Krátká jízda osobním automobilem zaznamenaná pomocí GPS (Zdroj grafických podkladů: Seznam.cz, a.s.)	57
A.5	Krátká chůze zaznamenaná pomocí GPS (Zdroj grafických podkladů: Seznam.cz, a.s.)	58

Seznam tabulek

3.1 Ukázka současné verze vlastní CAN specifikace	13
---	----

Kapitola 1

Úvod

Tato práce se zabývá vývojem řídícího softwaru autonomního vozidla vyvíjeného na Vysoké škole báňské. Software bude sloužit k navigaci vozidla areálem univerzity, plánování cesty a zajištění bezpečnosti provozu. Pro tento účel řídící kód využívá vstupy z GPS, kamerového systému, LiDARu a ovládací jednotky vozidla. Analýza obrazu a dat naměřených LiDARem není předmětem této práce, výsledek této analýzy je získáván z API komponenty určené právě k tomuto účelu, také vyvíjené jako součást projektu autonomní platformy.

Software, který je vyvíjen jako součást této práce, je tvořen převážně v Pythonu a je rozdělen do více komponent. Kromě řídícího softwaru byl vyvinut také jednoduchý simulátor umožňující testování řízení, skript umožňující automatickou obnovu mapových podkladů, a webová aplikace sloužící ke vzdálenému ovládání vozidla. Při tvorbě řídících komponent byla využita řada open source knihoven a veřejných zdrojů, včetně map OpenStreetMap [39] nebo Mapy.cz [37]. Zdrojový kód, který je součástí této práce, je rovněž otevřený a veřejně dostupný na Githubu [44, 45, 46, 47, 48, 49].

Kapitola 2

Architektura řídícího softwaru

Jak již bylo zmíněno, řídící software je rozdělen do dvou komponent, komunikujících mezi sebou pomocí protokolu TCP na bázi architektury klient-server. Jako server slouží program zajišťující přenos dat na rozhraní CAN. Server takto abstrahuje nízkoúrovňové a bezpečnostní záležitosti, mezi které patří výpočet kontrolních součtu, zasílání CAN zpráv, TX/RX synchronizace, žádání o možnost ovládat vozidlo, regulace rychlosti pomocí PI regulátoru nebo také zpracování výstupu z GPS modulu. Klient tak může implementovat pouze samotné plánování cesty, v dotazech na server stačí žádat pouze o dodržování rychlosti, úhlu natočení kol, a případně také o aplikaci nouzové brzdy. Server samozřejmě dokáže poskytovat zpětnou vazbu, tedy informace o aktuální rychlosti, pozici, natočení kol nebo zdraví systému (tzv. *healthcheck*). Server je v praxi většinou spouštěn na zařízení Raspberry Pi, ačkoliv může být spouštěn na libovolné jednotce připojené k rozhraní CAN samoředitelné platformy, za předpokladu, že je na dané jednotce nainstalován interpreter jazyka Python.

Toto dělení nejen zvyšuje úroveň abstrakce a tím i čitelnost kódu, ale zároveň umožňuje spustit program plánující cestu vozidla na vzdáleném stroji. Takto můžeme například program spouštět na výkonnějším stroj a vyhnout se omezením Raspberry Pi, virtualizovat ovládací software [12] nebo implementovat jiný typ klienta.

V současné době existují dva různí klienti. První klient se snaží vozidlo řídit na základě vstupů ze senzorů umístěných na vozidle a z komponenty implementující analýzu obrazu. Druhý klient umožňuje řídící pokyny zadávat manuálně do jednoduchého textového prostředí. Tento klient je určen pouze k ověření funkčnosti komunikace pomocí sběrnice CAN.

Případná virtualizace ovládacího softwaru [12] také umožní zautomatizovat monitorování řídícího programu [9, 10]. Ačkoliv k žádným pokusům o automatickou orchestraci a monitorování zatím nedošlo, bylo by možné například spouštět řídící software v Kubernetes [12], nastavit kontinuální nasazování pomocí CI/CD pipeline [11] a logy či metriky kontrolovat strojově pomocí nástrojů jako ELK stack [10] nebo Prometheus [9].

Kapitola 3

Server a CAN komunikace

TCP server slouží k umožnění vzdáleného ovládání vozidla pomocí libovolného klienta. Software serveru je spouštěn na zařízení Raspberry Pi připojeném k rozhraní CAN autonomní platformy. CAN komunikace je implementována za využití knihovny *python-can* a software je schopen vozidlu předávat pokyny a zároveň vozidlo regulovat a monitorovat. Sám nedokáže plánovat cestu nebo reagovat na vnější události ovlivňující vozidlo, dokáže však zpracovat interní chyby. Software tedy slouží čistě a pouze ke zpracování interních záležitostí vozidla.

Architektura softwaru a důvod dělení na server a klient jsou podrobněji popsány v kapitole 2 – Architektura řídícího softwaru.

3.1 Sběrnice CAN

Controller Area Network [3], zkráceně CAN, je standard poskytující určitý základ pro komunikaci na sběrnici. Standard definuje fyzické požadavky na sběrnici a formát zpráv. Součástí zprávy je osm bytů vyhrazených pro data. Sběrnice CAN je, mimo jiné, hojně využívána právě v automobilovém průmyslu.

3.1.1 Vlastní specifikace

Standard sice přiřazuje každé zprávě pole o velikosti osm bytů vyhrazené pro data, již však ne-definuje, jaká data mají zprávy obsahovat, v jakém formátu mají být data posílána, nebo jak se s nimi má pracovat [3]. Obsah zpráv tak musel být vydefinován vlastní specifikací, která prošla hned několika iteracemi, a jejíž finální podoba je výsledkem úzké spolupráce konstruktérů fyzické platformy a tvůrců řídícího softwaru. Specifikace byla navrhována tak, aby umožnila dostatečně jemné řízení vozidla, poskytovala řídícímu softwaru všechna potřebná jízdní data a zahrnovala množství prostředků pro kontrolu integrity zprávy, rychlé zachycení chyb či reportování problémů. Na chyby nebo problémy je samozřejmě důležité reagovat, v krajních případech až nouzovým zastavením

Tabulka 3.1: Ukázka současné verze vlastní CAN specifikace

Message Name	ID	Cycle (ms)	Byte	Length (bits)	Min Value	Max Value
System Status	201	20	1	1	0	1
Control Request	201	20	2	1	0	1
Steering Request	201	20	3	8	0	255
Drive Request	201	20	4	8	0	255
Drive Gain	201	20	5	2	0	4
Steering Gain	201	20	6	2	0	4
Message Count	201	20	7	8	0	255
Data Checksum	201	20	8	8	0	255

vozidla. Software pro ovládání platformy byl již několikrát přepisován na základě upravené specifikace, nejaktuálnější verze softwaru však implementuje nejnovější verzi specifikace. Ukázku části specifikace si lze prohlédnout v tabulce 3.1.

Specifikace, kromě formátu a obsahu dat, určuje také způsob kódování rychlosti vozidla nebo natočení předních kol. Tyto hodnoty jsou kódovány a uloženy v jednom bytu. Při práci s CAN sběrnicí tak je nutné data správně přepočítat a zakódovat, případně dekódovat a převést do formátu IEEE 754 [15].

3.2 TCP server a jeho metody

Za účelem zvýšení úrovně abstrakce a zjednodušení ovládání vozidla, ale také aby bylo umožněno vzdálené ovládání platformy a zjednodušila se výměna způsobů ovládání, je nízkoúrovňové řízení CAN zprávami abstrahováno a obaleno jednoduchým TCP serverem. Při vývoji serveru se přirozeně nabídla možnost implementace REST API [13] a využití protokolu HTTP [16] nebo využití jednoho z rozšířených RPC prokolů, jako jsou XMLRPC [14] nebo gRPC [27]. Ačkoliv validní, tyto způsoby implementace se jevily jako neefektivní – rozhraní poskytované serverem je velmi jednoduché, plaintextový protokol HTTP nebo variace protokolů RPC tak působily až zbytečně komplikovaně. Pro komunikaci byl nakonec použit vlastní binární protokol a jeho implementace přímo nad TCP sockety. Výhodou je rychlosť a jednoduchost implementace, úspornost a výkonnost řešení v reálném provozu nebo také fakt, že toto řešení nevyžaduje žádné externí závislosti nad rámec standardní knihovny jazyka Python. Nevýhody současného řešení by se projevily pouze při výrazném nárůstu komplexity poskytovaného rozhraní. Tento scénář by mohl vést k znepřehlednění současného řešení. Nutnosti zvyšování komplexity rozhraní ovšem v současné době nic nenasvědčuje.

Aktuální verze serveru implementuje celkem pět metod. Na vstupu přijímá celkem tři byty. První byte je vyhrazen pro identifikaci metody serveru. Následující dva byty dotazu obsahují data. Klient musí vždy poslat minimálně tři byty. Na vstupu jsou, nezávisle na metodě, pokaždé očekávány tři byty. Jestliže klient pošle v dotazu více než dva datové byty, budou nadbytečná data ignorována.

Jako identifikátor metody je využito celé nezáporné číslo. V případě, že první byte zprávy obsahuje identifikátor nepříslušící žádné metodě, je dotaz ignorován. Obsah datových bytů je určen danou metodou serveru. Pokud pro určitou funkcionality není třeba předávat serveru žádná data, může být obsah datových bytů libovolný, data budou načtena, avšak ignorována.

3.2.1 Metoda *drive*

Metoda *drive* slouží k předání požadované rychlosti vozidla a natočení kol. Obsah prvního datového bytu je interpretován jako rychlosť, druhý datový byte reprezentuje natočení předních kol. Oba datové byty jsou interpretovány jako celá znaménková čísla.

Fyzická změna rychlosti nebo natočení kol samozřejmě nemůže nastat okamžitě, rychlosť změny je omezena fyzikálními vlastnostmi konstrukce i PI regulátorem [1], proto nelze vrátit klientu potvrzení o provedené změně. Odpověď metody je ekvivalentní k odpovědi metody *healthcheck*. *Healthcheck* v odpovědi slouží jako potvrzení přijetí dotazu a schopnosti dotazu vyhovět. V případě, že klient potřebuje potvrzení o dosažení požadované rychlosti nebo úhlu natočení kol, je možné po zavolení metody *drive* opakováně volat metodu *info* a kontrolovat, zda vozidlo žádaných jízdních parametrů skutečně dosáhlo.

3.2.2 Metoda *healthcheck*

Metoda *healthcheck* slouží k ověření *živosti* systému. *Živost* je binární stav, systém je buď *živý*, nebo *mrtvý*. Server je považován za *živý*, jestliže je schopen zpracovávat TCP spojení a zároveň má plnou kontrolu nad vozidlem. V opačném případě je server považován za *mrtvý*. Metoda nepřijímá žádný vstup, datové byty jsou ignorovány.

Healthcheck ve své odpovědi posílá pouze jeden byte. Pokud je server živý a má kontrolu nad vozidlem, obsahuje odpověď nenulovou hodnotu. V případě, že server vozidlo ovládat nedokáže, se na výstupu metody *healthcheck* objeví nulová hodnota. Absence odpovědi pak samozřejmě znamená, že software serveru není spuštěný.

3.2.3 Metoda *info*

Info metoda TCP serveru se využívá k získání aktuálních jízdních dat vozidla. Na vstupu nepřijímá žádné parametry, všechny datové byty jsou tedy ignorovány. Na výstupu server vrátí tři byty. První dva byty obsahují znaménková celá čísla. Obsahem prvního bytu je aktuální rychlosť vozidla. Druhý byte obsahuje úhel natočení kol. Poslední byte poté obsahuje logickou hodnotu značící stav nouzové brzdy. Jestliže je hodnota třetího bytu nulová, nouzová brzda není aktivována. Nenulová hodnota značí, že došlo k aktivaci nouzového brzdění, a bude třeba jej deaktivovat, pokud chceme pokračovat v jízdě.

3.2.4 Metoda *ebrake*

Ebrake, nebo-li '*Emergency Brake*', česky 'nouzová brzda', slouží k ovládání nouzového brždění. První byte vstupu je serverem interpretován jako logická hodnota značící požadovaný stav nouzové brzdy. Nenulovou hodnotu server považuje za signál k aktivaci nouzového brždění. Při přijetí nulové hodnoty na vstupu poté dochází k deaktivaci nouzové brzdy. Druhý byte vstupu je ignorován.

Výstup metody *ebrake* je opět ekvivalentní výstupu metody *healthcheck*. Nenulová hodnota na výstupu značí, že server požadavek přijal a je schopen jej zpracovat. Nula reprezentuje stav, kdy server nemá kontrolu nad vozidlem a není schopen nouzovou brzdu aktivovat. Pokud však ke ztrátě kontroly došlo důsledkem interní chyby, vozidlo začne nouzově brzdit samo.

3.2.5 Metoda *position*

Position využijeme, chceme-li zjistit geografickou polohu vozidla. Tato metoda, která neakceptuje žádný vstup, ve svém 17 bytovém výstupu vraci tři hodnoty.

První byte obsahuje logickou hodnotu značící zda je geografická poloha věrohodná. Nulový byte značí polohu nevěrohodnou, nenulový byte věrohodnou. Poloha je považována za nevěrohodnou např. pokud nedojde ke správné inicializaci vozidla, GPS modul nedokáže zaměřit pozici, apod.

V následujících osmi bytech je zakódována zeměpisná šířka (latitude), v posledních osmi bytech pak zeměpisná délka (longitude). Ačkoliv se jedná o desetinná čísla, pro zjednodušení jsou zakódována jako čísla celá. Skutečná hodnota zeměpisné šířky a délky je vynásobena číslem 10^{10} a převedena na celé číslo. Výsledné číslo je poté odesláno ve formátu little endian. Při zpracování odpovědi serveru si tedy klient musí dát pozor na endianitu a správně dekódovat přijatou hodnotu.

3.3 Regulace PI regulátorem

Elektrický motor vozidla samozřejmě koncept rychlosti nezná. Elektrický motor je schopen vydávat určitý točivý moment, o který je možné si zažádat. Mezi točivým momentem a rychlostí vozidla ovšem neexistuje lineární závislost. Rychlosť vozidla může ovlivnit více faktorů. Např. při vysokém konstatním točivém momentu na vodorovné nebo svažující se vozovce by vozidlo mohlo neustále zrychlovat, minimálně dokud by nedosáhlo rychlosti nebezpečné nejen pro samotnou platformu, ale především pro své okolí. Naopak při stoupání by vozidlo mohlo na rychlosť ztrácat.

O vydání určitého točivého momentu je možné požádat za pomoci CAN sběrnice. Řídící software vyšle CAN zprávu s požadovanou rychlosťí. Kontrolní jednotka vozidla zprávu zpracuje, zkонтroluje, a předá požadavek elektromotoru, který upraví svůj výstup. Při ovládání vozidla ovšem není určování momentu vhodné. Podstatně jednodušší by bylo žádat o dodržování konkrétní rychlosťi, ne momentu. Proto je točivý moment třeba regulovat. A právě zde se jako řešení nabízí využití PI regulátoru [1].

Regulátor je zabudován do zdrojového kódu serveru a je implementován softwarově. TCP server na vstupu metody *drive* dostane požadavek na udržování rychlosti. Server metodu zpracuje a předá regulátoru požadovanou hodnotu. Druhým vstupem regulátoru je aktuální rychlosť, kterou lze nalézt v CAN zprávách kontrolní jednotky platformy. Na svém výstupu regulátor vrací požadovaný moment, který řídící program předá vozidlu pomocí sběrnice CAN.

Proporcionální složka regulátoru slouží ke změně točivého momentu na základě odchylky naměřené rychlosti od rychlosti požadované. Integrační složka regulátoru slouží ke sčítání předchozích odchylek a k udržování určité hladiny momentu, aby např. při dosažení požadované rychlosti, kdy odchylka bude rovna nule, nedošlo k vypnutí motoru. Derivační složka není v této aplikaci potřeba, proto je využit pouze PI regulátor.

Regulováno je i natočení kol. Zde je regulátor využit zejména k zabránění velkých skokových změn v požadavcích na natočení kol. Není žádoucí, aby požadovaný úhel například rychle osciloval mezi hodnotami -20 a 20 stupňů. PI regulátor zde slouží právě k zajištění pozvolné změny požadavku. Požadovaný úhel natočení kol je na sběrnici CAN zadáván ve stupních, aplikace regulátoru je zde velmi jednoduchá.

3.4 Python implementace

Pro implementaci kódu byl zvolen jazyk Python 3.6. Zdrojový kód serveru má pouze dvě externí závislosti, a to knihovny *python-can* a [33] *gpsd-py3* [29]. Dále je samozřejmě extenzivně využívána standardní knihovna jazyka Python.

3.4.1 Konfigurace

Ke konfiguraci komponenty jsou využívány systémové environmentální proměnné, a to zejména kvůli jednoduchosti řešení a konzistence s ostatními komponentami. Jelikož je program spouštěn pouze na Raspberry Pi fyzicky připojeném ke CAN sběrnici vozidla, k virtualizaci pomocí orchestračních nástrojů, jako je třeba Kubernetes, pravděpodobně nikdy nedojde. Argument o jednoduchosti konfigurace za využití proměnných prostředí při virtualizaci tak u této komponenty neplatí.

3.4.2 TCP server

K vytvoření TCP serveru byl využit modul *socketserver*, který náleží ke standardním modulům jazyka Python. Server funguje na synchronní bázi a zpracování dotazů probíhá pouze na jednom vlákně. K serveru je v jednom momentě připojen vždy jen jeden klient, jenž posílá pouze relativně malé množství dotazů. Asynchronní nebo vícevláknová implementace tak není potřeba, pouze by zapříčinila znepřehlednění kódu, a to s velmi nízkými až dokonce žádnými benefity. Server naslouchá na konfigurovatelném portu. Instanci rozhraní vozidla nebo port lze serveru předat parametrem

při inicializaci. Pokud tak uživatel neučinní, rozhraní vozidla je vytvořeno automaticky a číslo portu je vyčteno z enviromentální proměnné *SERVER_PORT*.

3.4.3 Rozhraní vozidla

Rozhraní vozidla, v kódu třída *Car*, představuje abstrakci nad sběrnicí CAN, jež vznikla, aby samotný TCP server nemusel pracovat přímo s CAN zprávami. Instance třídy *Car* umožňují TCP serveru nastavovat nebo získávat hodnotu úhlu natočení předních kol, rychlosť, stav záchranné brzdu, případně získat GPS souřadnice vozidla, aniž by TCP server musel počítat s momentem, CAN hodnotami, nebo pracovat s hardwarovými moduly.

Za účelem práce s CAN sběrnicí byly implementovány třídy *Transmitter* a *Receiver*, které slouží k přijímání a vysílání zpráv. Interně třídy využívají prostředky poskytované modulem *python-can*, zvenčí představují především abstrakci nad nízkoúrovňovými CAN záležitostmi. *Transmitter* na sběrnici vysílá zprávy *DriveMessage* a *ControlMessage*. *Receiver* přijímá zprávu *CarData*. Data po přijetí zpracuje a předá za pomocí callbacku třídě *Car*. Ke konverzi mezi CAN hodnotami a formátem IEEE 754 [15] slouží pomocné třídy *Driving* a *Steering*. GPS souřadnice jsou z fyzického GPS modulu připojeného k Raspberry Pi získávány za využití modulu *gpsd-py3*.

CAN zprávy jsou posílány periodicky. K zajištění periodicity je využita knihovna *python-can*, která zvládne periodické zasílání zpráv zajistit. Zasílaná data lze libovolně měnit za využití rozhraní abstraktní třídy *ModifiableCyclicTaskABC* poskytované modulem *python-can*. Knihovna pravidelně opakované vysílání zpráv na separátním vlákně zajistí sama. V kódu tak stačí vytvořit periodickou úlohu a pouze dle potřeby upravovat požadovaná jízdní data či kontrolní součty.

Kapitola 4

Simulátor vozidla

Simulátor vozidla, taktéž implementovaný za využití jazyka Python, má za cíl nahradit a nasimulovat server běžící na Raspberry Pi připojeném k autonomní platformě. Simulátor implementuje naprostě stejné API jako skutečný server. Fyzické vozidlo je však nahrazeno pouhou virtuální approximací. Simulace samozřejmě určitým způsobem počítá s fyzikálními reáliemi vozidla – natočení přední soupravy určitou dobu trvá, akcelerace není lineární, vozidlo samovolně zpomaluje, pokud motor neposkytuje dostatečný výkon. Přesto je simulace zjednodušená a počítá pouze s elementárními fyzikálními zákony. Simulátor vznikl s cílem umožnit jednoduché a rychlé testování jednotlivých částí řídího softwaru. Realistická a přesná simulace by byla pro tento účel zbytečně složitá a silně překračovala rozsah a cíl projektu.

Kód simulátoru také nemá žádné závislosti. Využívá pouze standardní knihovnu jazyka Python verze 3.6. Ke konfiguraci lze využít environmentální proměnné operačního systému. V konfiguraci lze specifikovat port, na kterém má TCP server naslouchat, zapnout ladící funkce, nebo také nastavit výchozí pozici vozidla.

Program po spuštění vytvoří TCP server a inicializuje vozidlo ve výchozí pozici. Na základě vstupu TCP serveru je poté vozidlo ovládáno. Geografická poloha simulované platformy je automaticky aktualizována s pohybem virtuálního vozidla, aby výstup metody *drive* TCP serveru odpovídal reálnému výstupu v praktickém zapojení. Klientu je tak poskytnut ekvivalent skutečného vozidla. Simulátor je následně využíván například při testování práce s mapovými podklady, jako je vyhledávání a plánování cesty nebo hledání nejbližší využitelné vozovky nebo při vývoji a testování webové aplikace sloužící k monitorování a ovládání vozidla, tedy když plně realistická simulace není nutná a jednoduchá approximace více než stačí.

Kapitola 5

Klient a ovládací software

Účelem klienta je plánovat cestu a rozhodovat o chování a pohybu vozidla na základě vysokoúrovňových informací a pomocí vysokoúrovňových příkazů. Komponenta funguje jako klient ve vztahu nejen k serveru zajišťujícímu CAN komunikaci, ale také softwaru zajišťujícímu analýzu okolního prostředí za pomocí soustavy kamer a LiDARů. Důvodem volby této architektury bylo zjednodušení softwaru pro plánování cesty. Klient nemusí řešit frekvenci CAN zpráv, analýzu obrazu, ani implementovat asynchronní server nebo synchronizaci vláken – nemůže se například stát, že by se v polovině výpočtu změnil vstup z kamery. Klient si všechny informace vyžádá až v momentě, kdy je doopravdy vyžaduje, provede rozhodnutí, a dokud nedosáhne cíle, rozhodovací cyklus periodicky opakuje.

Tato softwarová komponenta tedy zastává primárně roli klienta, zejména v relaci k vozidlu, to ovšem neznamená, že v roli klienta musí působit ve všech ohledech a všech aplikacích. Například pokud by došlo k implementaci monitorování pomocí technologie Prometheus [9], musel by software vystavit své metriky pomocí HTTP serveru. Dále komponenta implementuje jednoduché REST API, které umožňuje vzdálené ovládání za využití protokolu HTTP. API je podrobněji popsáno v sekci 5.3 – REST API.

Komunikace s Raspberry Pi na vozidle probíhá pomocí vlastního binárního protokolu popsaného v části 3.2 – TCP server a jeho metody. Podobně je vlastní protokol využit také při komunikaci s analyzátem okolního prostředí. V obou případech byl pro přenos dat zvolen protokol TCP, a to především z toho důvodu, že TCP dokáže zajistit doručení všech paketů do cílové destinace [23]. Protokol UDP [24] by byl vhodný například pokud by data z kamery byla streamována, nebo pokud by mezi softwarem pro plánování cesty a CAN komunikátorem existoval konstantní proud dat, tedy opět stream, v tomto případě však stream příkazů, ne grafických vstupů. K tomu ovšem nedochází, řídící software si data vyžaduje teprve v momentě, kdy je doopravdy potřebuje. Navíc je při řízení vozidla extrémně důležité, aby nedocházelo ke ztrátě paketů a vozidlo dostalo všechny odeslané pokyny a následně jejich přijetí potvrdilo.

5.1 Definice cílů cesty vozidla

Aby mělo plánování cesty autonomní neosazené platformy nějaký smysl, musí být znám cíl cesty vozidla. V praxi může tento cíl představovat například 3D tiskárna sloužící jako zdroj materiálu, nebo destinace, do které má být naložený materiál dovezen. K logické definici těchto cílů slouží právě waypointy. Waypointy, tedy geografické body na mapě, lze zadat nejen pomocí grafického vizualizéru (viz podkapitola 5.2 – Vizualizér), ale také pomocí webové aplikace, která by v praxi měla být preferována. Webové aplikaci se podrobněji věnuje kapitola 8 – Webová aplikace.

Waypointů je možné zadat teoreticky neomezené množství, prakticky je samozřejmě množství omezeno pamětí, kterou může plánovací software využít. Po zadání waypointu plánovací software nalezne nejkratší možnou cestu k danému místu na mapě a pokusí se vozidlo navigovat až k požadovanému cíli. Tento postup opakuje postupně pro každý waypoint, a to v takovém pořadí, v jakém uživatel waypointy zadal. Při dosažení posledního waypointu se vozidlo zastaví a čeká na další pokyny. Pokud se uživatel při vytváření waypointu pokusí waypoint vytvořit mimo silnice definované mapovými podklady, je waypoint automaticky umístěn na nejbližší silnici. Práce s mapovými podklady je podrobněji rozebrána v kapitole 7 – Práce s mapovými podklady.

Jelikož je náročné zastavit přesně v určitém geografickém bodě, už jen kvůli nepřesnosti GPS, počítáme s určitou tolerancí v okruhu okolo uživatelem zadaného bodu. Navigace čistě pomocí technologie GPS byla vyzkoušena a je popsána v sekci 6.1 – Řízení vozidla pomocí GPS. Díky relativně vysoké nepřesnosti GPS se tento styl navigace projevil jako nepřesný a náročný na využití. Pro přesnou navigaci vozidla a zastavení v přesném bodě by ovšem bylo možné využít vstupu z kamery, a po dosažení geografického cíle autonomní platformu navigovat pomocí vizuálních prvků, například tedy grafického kódu nebo vodorovného značení definujícího parkovací místo, vozidlo dovést do přesného bodu. Tato funkcionality by následně umožnila například automatické strojové vykládání převáženého nákladu. V současné době, především z časových důvodů, tato funkcionality není vyvíjena, ačkoliv však není vyloučena.

5.2 Vizualizér

Klient obsahuje jednoduchý vizualizér, pomocí kterého lze na mapě zobrazit pozici vozidla, historii pohybu, vykreslit softwaru známé cesty, aktivní waypointy, a zvýraznit cestu, po níž se vozidlo aktuálně pohybuje. Dále vizualizér umožňuje vytváření waypointů levým tlačítkem myši. Vizualizér slouží především k vývojovým a testovacím účelům. V reálném provozu vozidla by měla být využívána především webová aplikace.

5.3 REST API

REST API [13] slouží ke vzdálenému získávání informací a zadávání pokynů plánovacímu softwaru. API v současné době implementuje čtyři endpointy. Předávaná data jsou obsažena v těle dotazu klienta nebo odpovědi serveru a naformátována ve formátu JSON [19], případně jsou přímo součástí URI [16], pokud je tento přístup pro daný případ vhodnější.

5.3.1 Endpoint */waypoints*

Endpoint */waypoints* slouží k práci s waypointsy. Podporuje tři metody standardu HTTP [16], konkrétně GET, POST a DELETE. Metoda GET slouží k získání seznamu všech aktuálních waypointsů. Server po přijetí požadavku naformátuje seznam waypointů a vrátí jej jako JSON pole ve své odpovědi. Metoda POST se využívá k vytváření waypointů. Server v těle požadavku očekává jeden JSON objekt se dvěma atributy, *latitude* a *longitude*. Po přijetí dotazu je JSON rozparsován a na základě obsahu objektu je vozidlu přidán nový waypoint. Metoda DELETE je přirozeně využita k mazání waypointů. Jedním dotazem je možné smazat jeden waypoint. Metoda DELETE očekává jeden parametr, id, který značí unikátní identifikátor waypointu, který má být smazán. V případě metody DELETE je parametr očekáván již v URI, obsah těla nebude API u této metody v potaz.

5.3.2 Endpoint */position*

/position slouží k získání geografické pozice autonomního vozidla. Implementuje metodu GET a nepřijímá žádné parametry. V těle odpovědi vrací jeden JSON objekt s atributy *latitude* a *longitude*, představující aktuální umístění vozidla.

5.3.3 Endpoint */heading*

GET dotaz na endpoint */heading* umožňuje získat aktuální natočení vozidla. Natočení je uvedeno ve stupních a odpovídá natočení v kartézském souřadnicovém systému, není tak relativní k pólům planety Země. Tato funkce slouží především k vykreslování vozidla na mapě, která je samozřejmě projekcí glóbu do kartézského systému. Odpověď zde také představuje JSON objekt s jedním atributem pojmenovaným *heading*. Jiné metody standardu HTTP endpoint */heading* nepodporuje.

5.3.4 Endpoint */info*

Endpoint */info* také využívá pouze metodu GET a veškerý vstup ignoruje. Ve své odpovědi vrací pozici vozidla, jeho natočení, ale také seznam waypointů. Odpověď serveru představuje objekt se třemi atributy. Atribut *position* představuje objekt geografického bodu. Atribut *heading* je pouze primitivní desetinné číslo. Pod atributem *waypoints* se poté skrývá pole waypointů. Endpoint *info* tedy slouží jako ekvivalent k GET dotazu na všechny tři předchozí endpointy. Hlavní výhodou

využití endpointu `/info` je samozřejmě nižší množství TCP spojení oproti volání alespoň dvou ze tří předchozích endpointů najednou.

5.3.5 Výhody a nevýhody REST API

Nespornou výhodou architektonického stylu REST [13], zejména pak v porovnání s binárními RPC protokoly [27], vlastními protokoly, apod., je především jednoduchost jeho využití. Využití formátu JSON [19] umožňuje jednodušší ladění a vyhledávání chyb, právě protože je formát lidsky čitelný. K využití a zavolání REST API stačí jakýkoliv HTTP [16] klient, proto je jednoduché implementovat klienta v jakémkoliv seriálním jazyce. Není třeba doufat, že pro daný jazyk bude existovat implementace Protocol Buffers [30] kompilátoru a gRPC [27] klienta. Dokonce je možné REST API volat i z webového prohlížeče nebo třeba z terminálu za využití programu *cURL* [38]. Za nevýhodu lze považovat nižší výkon a vyšší nároky na přenesený objem dat v porovnání s binárními protokoly, ovšem s nízkým množstvím požadavků v současné aplikaci, v kombinaci s rychlostí moderních procesorů, není rychlosť zpracování dotazu faktorem.

5.4 Python implementace

Software je, konzistentně s ostatními komponentami, vyvíjen v programovacím jazyce Python verze 3.6. Kromě standardních modulů jazyka Python jsou zde využity knihovny *Pygame* [32], *Geopy* [28], *Osmium* [31] a *Flask* [17].

5.4.1 Konfigurace

Program lze konfigurovat pomocí enviromentálních proměnných operačního systému. V případě řídícího softwaru už lze diskutovat také o možné virtualizaci. V takovém případě je samozřejmě tento způsob konfigurace vhodný zejména kvůli své jednoduchosti v kombinaci s technologiemi jako jsou Docker nebo Kubernetes [12], které využití tohoto způsobu konfiguraci samy podporují. V některých případech ovšem může být tento způsob konfigurace až příliš jednoduchý, a to pokud potřebujeme vyjádřit například objekty se zanořenými atributy, seznamy, apod. V takovém případě by určitě bylo vhodnější využít například konfigurační mapy Kubernetes a pro konfiguraci využít třeba formát YAML [18] nebo JSON [19].

5.4.2 TCP klient

Implementace TCP klienta využívá modul *socket* standardní knihovny jazyka Python. Funkce pro komunikaci pomocí protokolu TCP se nachází v souboru *client.py*. Využití je velmi jednoduché. Při překladu souboru interpreterem se načte konfigurace z enviromentálních proměnných, proto stačí pouze importovat daný soubor, nebo jen požadované funkce, a ty poté volat s případnými argumenty. Není třeba klienta konfigurovat v kódu, vytvářet instance tříd, apod.

Klient samozřejmě využívá vlastní protokol popsáný v sekci 3.2 – TCP server a jeho metody. K volání funkcí serveru implementuje pět metod, *drive*, *healthcheck*, *info*, *position* a *ebrake*, které volají stejnojmenné funkce serveru. Dále také implementuje funkci *camera_info*, jež se nedovolává na řídící server, ale na server provádějící analýzu vstupu z kamery. V tomto případě ovšem server nevrací informace o okolním prostředí, tato funkce slouží pouze k získání pozice detekovaného grafického kódu. Funkce *camera_info* je tedy využívána především při následování kódů, ne při navigaci pomocí geografických waypointů.

5.4.3 REST API

REST API je implementováno jako velmi jednoduchá Flask [17] aplikace. Narození od klienta musí být API inicializováno zavolením funkce *init*, jíž je v argumentu předán ukazatel na instanci třídy *CarController*, která zajišťuje řízení vozidla. Předání tohoto objektu je potřeba proto, aby API mohlo interagovat s řídícím kódem. API se spouští na separátním vlákně, třída *CarController* proto musí být vláknově bezpečná (anglicky thread-safe). V opačném případě by mohlo dojít např. ke konkurenčnímu přístupu k atributům.

5.4.4 Řídící kód

K ovládání vozidla slouží především třídy *CarController* a *PathPlanner*. Třída *CarController* slouží k vysokoúrovňovému ovládání vozidla. Pomocí této třídy lze například nastavit waypointy nebo zapnout sledování grafických kódů. Dále také agreguje a spravuje instance pomocných tříd, jako jsou *PositionFetcher* a *PositionTracker*, sloužící k získávání a sledování pozice vozidla, *Visualizer*, třídu umožňující grafickou vizualizaci, nebo instanci třídy *Map* obsahující mapové podklady univerzitního kampusu.

PathPlanner poté rozhoduje o pohybu vozidla. Na základě poskytnutých informací, jako jsou například mapové podklady, vstup z GPS nebo detekce obrazu, apod., rozhoduje jak o okamžitých akcích, tak o dlouhodobém pohybu a dráze vozidla ve snaze dovést vozidlo do cíle. Za účelem plánování cesty bylo vyzkoušeno hned několik způsobů. Zkoušeným způsobům, poznatkům ze zkoušení postupů, nalezeným problémům a výsledkům výzkumu se podrobněji věnuje kapitola 6 – Testované způsoby řízení vozidla.

5.4.5 Využité knihovny

Knihovna *pygame* [32] byla zvolena pro implementaci vizualizéru zejména díky jednoduchému využití a vykreslování grafických podkladů, mezi které patří nejen základní geometrické útvary, jako jsou třeba úsečka či kruh, ale také obrazové podklady a grafické sprity.

Modul *geopy* [28] je využíván pro své možnosti práce s geografickými souřadnicemi. Důležitou, a v kódě často využívanou, funkci knihovny *geopy* je funkce *geopy.distance.distance*, jež poskytuje funkcionality pro výpočet fyzické vzdálenosti mezi dvěma geografickými body. Získaná vzdálenost

v metrech se poté využívá například při hledání nejkratší cesty na mapě, nebo také při převodu geografických souřadnic na souřadnice kartézské, které následně umožňují zjednodušení výpočtů za cenu nižší, avšak stále dostačující, přesnosti, využití goniometrických souřadnic pro výpočet úhlu, apod.

Mikroframework *Flask* [17] je framework umožňující tvorbu HTTP serveru a zpracování HTTP dotazů. Jedná se o jeden z nejznámějších a nejrozšířenějších Python frameworků. Jako mikroframework je velmi jednoduchý na využití, ale také relativně drobný a minimalistický. Není proto sám o sobě vhodný na tvorbu webových aplikací, za tímto účelem je efektivnější využít technologií stojících na *Flasku* a rozšiřujících jej o větší množství funkcionality, pro tvorbu REST API je ovšem *Flask* velmi dobrá volba. Nevýhodou *Flasku* je chybějící podpora asynchronního zpracování dotazů, proto se dnes často na úkor *Flasku* využívá třeba *Starlette* [35] nebo z něj vycházející *FastAPI* [34], které díky své asynchronní povaze dokážou zpracovat podstatně vyšší množství dotazů. V současné aplikaci však na API chodí maximálně jednotky dotazů za sekundu a *Flask* stačí více než dostatečně.

Kapitola 6

Testované způsoby řízení vozidla

Při vývoji softwaru byly zkoušeny hned tři způsoby řízení vozidla, a to řízení čistě pomocí GPS, řízení pouze pomocí grafických kódů a nakonec kombinace řízení pomocí vstupů z GPS, kamery a případně také LiDARu. První dva způsoby se projevily jako v praxi téměř nevyužitelné, alespoň samy o sobě. I přesto výzkum těchto technik řízení přinesl spoustu poznatků, objevů, zkušeností, ale také technologického progresu a kódu, proto výzkum nelze považovat za neúspěšný. Ačkoliv jsou totiž techniky řízení nepraktické samy o sobě, lze je zkombinovat s jinými technikami, a tím vytvořit spolehlivý software schopný bezpečně řídit vozidlo. K tomuto účelu lze samozřejmě také využít zdrojových kódů napsaných při provádění výzkumu.

6.1 Řízení vozidla pomocí GPS

První pokusy o řízení proběhly za pomocí technologie GPS. Při testování se nepočítalo ani s mapovými podklady, autonomní platforma měla jezdit pouze po omezené ploše, aby se vyzkoušely možnosti tohoto způsobu řízení. K vozidlu byl připojen GPS modul, pomocí kterého server spuštěný na zařízení Raspberry Pi určoval aktuální geolokaci vozidla. Řídící kód byl naprogramován tak, aby vozidlo vedl přímo k nejbližšímu waypointu. Nastavení waypointů tuto skutečnost reflektovalo – waypointy byly rozmištěny tak, aby vozidlu nestála v cestě žádná překážka, budova, obrubník, apod. Navigace ve 2D prostoru bez žádných překážek není nic složitého, na první pohled tedy úspěchu nic bránit nemělo. Přesto se ovšem objevilo několik problémů, které bylo třeba řešit.

6.1.1 Kalibrace a zaměření GPS

První problémy s GPS se projevily již při spuštění serveru a inicializaci GPS modulu. Jedním z těchto problémů bylo pomalé úvodní zaměření vozidla ve venkovních prostorech. Tento efekt je sice nežádaný, ovšem není neřešitelný. Řídící kód byl upraven tak, aby tuto skutečnost bral v potaz a před rozjezdem alespoň deset sekund čekal, aby měl GPS modul možnost pozici správně určit, a aby se omezila oscilace naměřených hodnot, které jsou při zaměřování velmi nepřesné. Navíc není

předpokládáno, že by v případném reálném provozu docházelo k častému vypínání vozidla a s ním i GPS modulu. Také lze projevy problému omezit například nákupem dražšího hardwaru, nebo neustálým napájením GPS, a to i v momentě, kdy je vozidlo vypnuté. V současném stavu ovšem krátké čekání po spuštění vozidla nevadí, proto tato řešení nejsou nyní využívána v praxi.

Druhý problém, nezaměřitelnost vozidla ve vnitřních prostorech je problém podstatně větší. Platforma stavěná za účelem rozvozu materiálu v industriálních prostorech pohybu uvnitř musí být schopna, v opačném případě bude její využití velmi omezené. A jelikož má vozidlo být co nejjednodušší na využití a nemělo by vyžadovat stavbu dodatečné infrastruktury umožňující zaměření ve vnitřních prostorech, bylo už v tomto momentě rozhodnuto o tom, že bude třeba implementovat také řízení pomocí systému kamer, LiDARů, nebo kombinace obojího.

6.1.2 Nepřesnost GPS

GPS není schopna dosáhnout milimetrové přesnosti. K tomuto tématu existuje množství publikací [8]. Přesto nebyla možnost řízení vozidla pouze za využití vstupu z GPS zavržena bez dostatečného testování daného způsobu řízení. Určitý čas byl proto věnován právě zkoumání velikosti a závažnosti odchylky a možnostem jejího odfiltrování.

6.1.2.1 Problémy při prvním testování

Ovládací software vozidla byl při implementaci nejprve testován za využití simulátoru, který pozici vozidla počítá za využití rychlosti vozidla a jeho přesně vypočítané pozice. Simulátor aktualizoval pozici vozidla velmi často a v softwaru nebyla implementována žádná umělá odchylka ani zpoždění. Při testování tak simulované vozidlo jezdilo korektně, vždy včas zatáčelo a správně hledalo přímou cestu k cíli. Problém ovšem nastal při testování softwaru v praxi na fyzickém vozidle s reálným GPS modulem.

První testovací trasy byly pomocí waypointů definovány tak, aby vozidlo nemuselo prudce zatáčet, aby nenařazilo na žádné překážky a aby nebyla ohrožena bezpečnost osob, majetku, ale také samotného vozidla. Vozidlo po spuštění ovládacího softwaru úspěšně zahájilo pohyb. Po bezpečném a rovném rozjezdu bylo povoleno zatáčení (viz oddíl 6.1.4 – Směr natočení vozidla), v tomto momentě však nastaly problémy. Autonomní platforma začala nepředvídatelně zatáčet a směřovat špatným směrem. Často při první příležitosti změnila trajektorii pohybu a snažila se najet do nejbližší překážky. Toto chování bylo nečekané, neboť na simulátoru fungoval software bezproblémově, a také nežádoucí, jelikož waypointy byly definovány tak, aby vozidlo jelo rovně a zatáčelo pouze minimálně.

Nepřesnost GPS se samozřejmě nabízela jako hlavní viník této nepříjemné situace, proto byla přesnost GPS modulu přezkoušena hned na místě. Software byl upraven tak, aby naměřené GPS hodnoty logoval na standardní výstup programu. Naměřené hodnoty ihned potvrzily pravdivost doménky – nepřesnost systému GPS byla na potřeby autonomního řízení s využitím GPS jako jedi-

ného vstupu příliš vysoká. Modul často umisťoval vozidlo jinde, než kde se ve skutečnosti nacházelo. Špatná lokalizace vozidla následně zapříčinila špatné určení směru pohybu autonomní platformy (viz oddíl 6.1.4 – Směr natočení vozidla) a software začal vozidlo směrovat špatným směrem.

Zde se samozřejmě objevil velký problém z hlediska řízení vozidla. Možnost ovládání autonomní platformy za využití pouze vstupu z GPS však stále nebyla zavrhnuta. Software proto byl znovu upraven. Tentokrát došlo k úpravě způsobu určování aktuální pozice vozidla. Pozice se samozřejmě stále získávala z GPS modulu, nově ovšem software přestal pracovat s nezpracovanými daty poskytovanými modulem. Místo využití pouze těch nejnovějších souřadnic software nyní využíval vážený průměr posledních deseti naměřených hodnot. Vážený průměr byl zvolen za účelem minimalizace zpoždění, které by při rychlém pohybu přineslo využití obyčejného neváženého aritmetického průměru.

Při výpočtu váženého průměru byla nejnovější naměřené souřadnice přiřazena váha o hodnotě 1. Váha následujících hodnot poté odpovídala 0,9 násobku předchozí váhy. Při vykreslení pozice získané pomocí váženého průměru nejnovějších deseti hodnot vypadala grafická vizualizace podstatně lépe vedle vizualizace nezpracovaných čistých dat poskytnutých GPS modulem. Při vyzkoušení na fyzickém vozidle se však nepřesnost GPS, zvlášt v kombinaci s pomalou odezvou (viz sekce 6.1.3 – Pomalá odezva GPS), stále jevila jako veliký problém. Následující výzkum se tedy věnoval právě nepřesnosti GPS a snaze zjistit, jak nepřesná GPS doopravdy je, jaké jsou reálné dopady nepřesnosti při řízení a do jaké míry pomáhá v předchozím odstavci zmíněný vážený průměr nejnovějších deseti naměřených hodnot.

6.1.2.2 Testování přesnosti GPS za pomocí mobilní aplikace

Za účelem testování GPS vznikla velmi jednoduchá aplikace pro mobilní operační systém Android. Aplikace byla naprogramována v programovacím jazyce Kotlin – oficiálně doporučeným jazykem pro tvorbu aplikací pro zmiňovaný operační systém společnosti Google [36]. Samotná aplikace nebude v této práci kvůli své jednoduchosti podrobně rozebírána. Aplikace obsahuje pouze jedno velké funkční tlačítko, kterým lze zapnout logování GPS souřadnic. Souřadnice jsou logovány jednou za sekundu do obyčejného textového souboru uloženého v paměti telefonu. Každý soubor je označen časem vytvoření – tedy časem, kdy uživatel spustil logování geolokace. Soubor musí být po dokončení logování extraiován z paměti telefonu manuálně. Aplikace byla testována pouze na zařízení Samsung Galaxy Note 9 s nejnovější verzí softwaru, jež byla v době tvorby aplikace pro daný hardware dostupná. Výsledný program není publikován v žádném veřejném obchodě a slouží skutečně pouze jako jednorázová utilita pro krátké a rychlé testování. Zdrojové kódy jsou ovšem volně dostupné na Githubu [49] a také jsou přiloženy k této práci.

Dokončení mobilní aplikace umožnilo následné testování přesnosti GPS v praxi. Měření GPS souřadnic pomocí mobilního telefonu bylo prováděno na území obce Kobeřice a přineslo výsledky v určitých ohledech překvapivé, v jiných zase předpokládatelné. Naměřené hodnoty byly za úče-

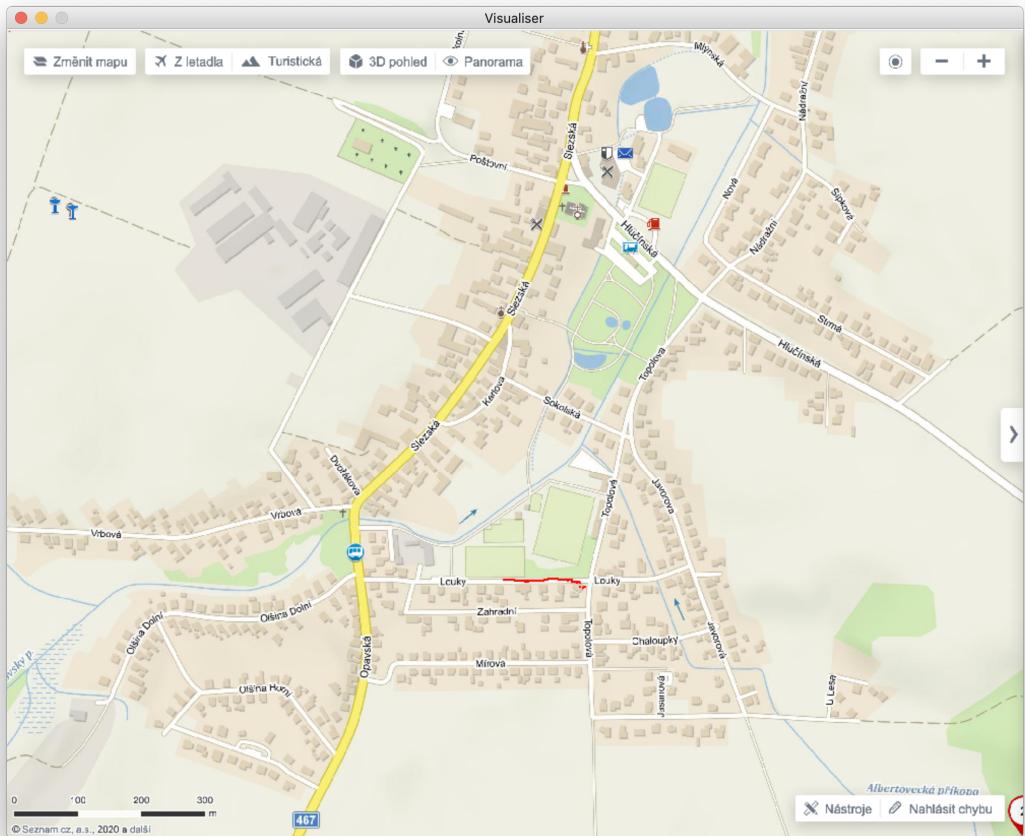
lem vyhodnocení měření vykresleny do grafických mapových podkladů. Naměřené hodnoty jsou v obrázcích vyznačeny červenou barvou.

Ze zajímavých výsledků lze zmínit například cestu středem rovné ulice. Měření bylo v tomto případě provedeno rovným pohybem přesným středem silnice. Měření začalo i skončilo ve středu silnice. Naměřené hodnoty jsou graficky znázorněny na obrázku 6.1. GPS modul v počátku měření naměřil hodnotu v domě na pokraji silnice, tedy mimo vozovku, na které se zařízení ve skutečnosti nacházelo. Po chvíli se měření stabilizovalo. Zajímavé ovšem je, že na sebe naměřené hodnoty navazují. Předpoklad, že budou naměřené hodnoty náhodně rozmístěny okolo středu vozovky byl tímto vyvrácen. Také tím byla vyvrácena doměnka, že aplikace váženého průměru na naměřené hodnoty zlepší přesnost. Ve skutečnosti by přesnost zůstala přinejlepším podobná, neboť z obrázku můžeme vyčít, že pokud se naměřené hodnoty odchylují od středu silnice, zůstávají vychýlené po delší dobu. Pokud by na tyto hodnoty byl aplikován vážený průměr, průměrná hodnota by zůstala pozadu za reálnou hodnotou, odchylka od středu by však odfiltrována nebyla. Zároveň lze však vidět, že se odchylka po určité době změní a naměřené souřadnice se odchýlí na opačnou stranu. Odchylka tedy není konstatní a nelze ji jednoduše odečít.

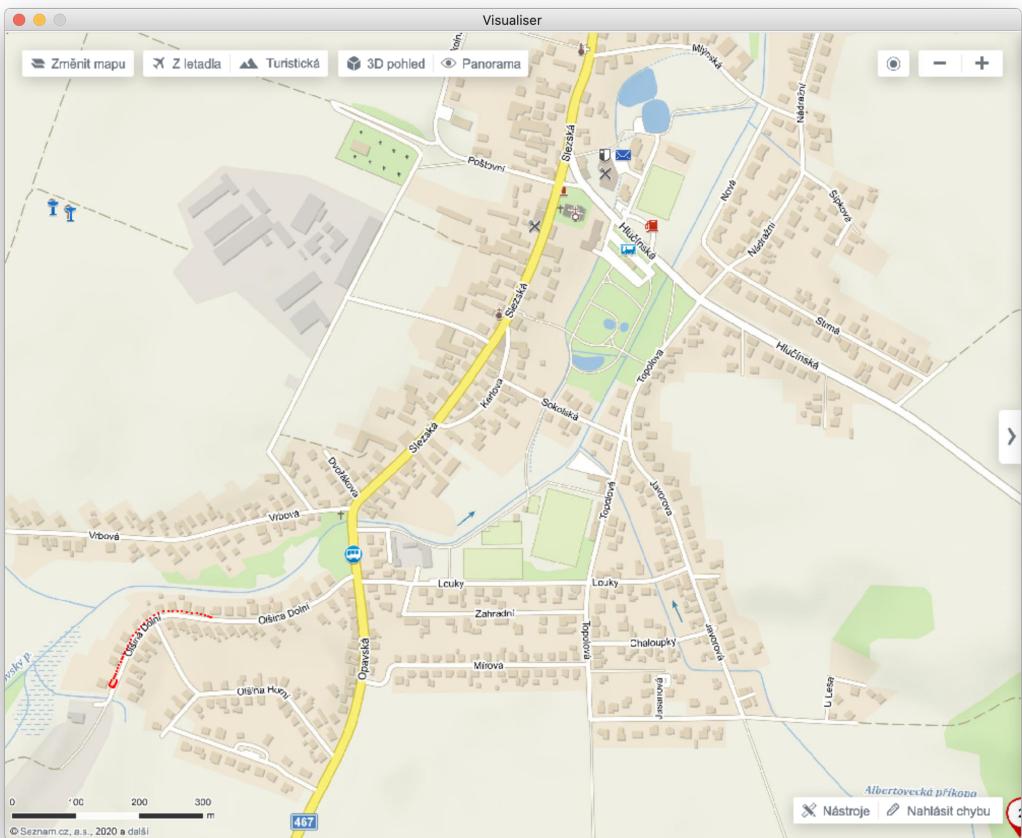
Zároveň se zde nabízí teorie, že operační systém Android na výstup GPS modulu sám aplikuje algoritmy pro odfiltrování nepřesností, a uživateli API vrací již zarovnané hodnoty. V takovém případě by pokusy o očištění těchto hodnot od odchylky neměly smysl. K tomuto tématu se nepodařilo dohledat dodatečné informace, ovšem pokud by tato teorie byla pravdivá, stačila by data poskytovaná API systému Android k potvrzení příliš vysoké nepřesnosti GPS souřadnic i po aplikaci filtrů.

Jako další zajímavé měření lze považovat měření za jízdy osobním automobilem. Vozidlo dosahovalo rychlosti 30 km/h, tedy maximální povolené rychlosti na silnici, na níž bylo měření prováděno. Měření bylo samozřejmě provedeno dvěma osobami, z nich jedna řídila vozidlo, zatímco druhá operovala mobilní telefon s aplikací. Nedošlo tak k žádnému ohrožení bezpečnosti provozu. Výsledek měření lze vidět na obrázku 6.2.

Na obrázku 6.2 je vidět, že při vyšší rychlosti naměřené hodnoty již nejsou spojité. Stále se však v průběhu měření objevuje určitá odchylka, tentokrát ovšem skoro konstantní. Lze odečít, že směr a rychlosť vozidla není velký problém vyhodnotit při vyšších rychlostech, kdy je projev odchylky relativně ke skutečné změně nízký. Ovládací software však vzniká za účelem řízení autonomní platformy menších rozměrů kampusem univerzity. V takovémto případě je rychlosť 30 km/h nereálná. Pro vozidlo a jeho náklad nemusí být vhodná a pro bezpečnost provozu v areálu je příliš vysoká. Dále si lze povšimnout, že ačkoliv je odchylka přijatelně nízká pro určení směru pohybu, přesně určit pozici vozidla není možné. GPS modul velmi často umístil osobní automobil do některého z domů stojících podél silnice. I přes takto vysokou odchylku lze jednoduše určit, po které silnici se vozidlo momentálně pohybuje. Za účelem přesné navigace pouze pomocí GPS, což byl původní cíl výzkumu, je ovšem třeba přinejhorším centimetrové přesnosti, jež GPS momentálně dosáhnout nedokáže.



Obrázek 6.1: Rovná cesta středem ulice zaznamenaná pomocí GPS (Zdroj grafických podkladů: Seznam.cz, a.s.)



Obrázek 6.2: Krátká jízda osobním automobilem zaznamenaná pomocí GPS (Zdroj grafických podkladů: Seznam.cz, a.s.)

Výsledky všech měření v plné velikosti jsou obsaženy v příloze publikace.

6.1.2.3 Návrhy řešení nepřesnosti GPS

Za účelem vytvoření funkční implementace ovládacího softwaru využívajícího vstup pouze z GPS vzniklo množství teorií zabývajících se způsoby filtrování odchylky. Předmětem prvního teoretického řešení bylo vzít nejbližší bod na nejbližší silnici vzhledem k naměřené pozici, a předpokládat, že se autonomní platforma po dané silnici pohybuje. Tato teorie však byla rychle vyvrácena. Pokud by se vozidlo pohybovalo po trajektorii neparalelní se středem silnice, velmi rychle by opustilo silnici, ačkoliv by upravené hodnoty vykazovaly rovný pohyb po naefinované cestě.

Druhou ze zavrhnutých teorií byla idea naměřenou hodnotu validovat výpočtem. Pokud vozidlo popojízdí určitou rychlosť s určitým úhlem natočení kol, není reálné, aby náhle rychle změnilo svůj směr a začalo se pohybovat na opačnou stranu. Tato úvaha na první pohled působí logickým dojmem, má ovšem jeden veliký problém. Abychom mohli validovat novou pozici na základě pozice předchozí a dopočítané očekávané změny pozice, museli bychom si být jistí přesnosti pozice předchozí. Díky nepřesnosti GPS touto jistotou ovšem nedisponujeme, validace tak může být problematická a nepřesná.

Mezi návrhy řešení se zařadilo také řešení hrubou silou – využití většího množství GPS modulů a práce s průměrem naměřených hodnot. Tento návrh však také nebyl bezchybný. Pokud by nebyly moduly synchronizovány a neaktualizovaly by své naměřené hodnoty všechny najednou v krátkém časovém rozmezí, nemusel by průměr být vypočítáván, zejména díky využití starých hodnot. Dále toto řešení samozřejmě vyžaduje připojení většího množství GPS modulů k jednomu zařízení Raspberry Pi. Nakonec však nedošlo k testování tohoto řešení, neboť řízení čistě pomocí vstupů z GPS bylo ještě před testováním zavrhnuto.

6.1.3 Pomalá odezva GPS

Nepopiratelným problémem při snaze řídit vozidlo pouze za pomocí GPS byla samozřejmě také pomalá odezva GPS modulu. GPS modul aktualizoval souřadnice frekvencí pouze 1 Hz. Jedna aktualizace za sekundu není žádný problém při tradičním využití GPS, tedy například pokud vozidlo řídí člověk, zároveň však nepředstavuje velký problém ani při řízení pomocí vstupu z kamery. Pokud však má být vozidlo řízeno pouze vstupem z GPS, je frekvence jednoho hertze příliš nízká. Jestliže vezmeme v potaz, že se vozidlo může pohybovat rychlostí několika metrů za vteřinu, zvláště poté v kombinaci s nepřesnou GPS, která může naměřit lokaci s odchylkou více než jeden metr, mohlo by v praxi jednoduše dojít k situaci, kdy vozidlo například projede křižovatkou, na které mělo odbočit, nebo křižovatku detekuje příliš pozdě, pokusí se zatočit, ale narazí například na obrubník, který díky absenci vstupu z kamery nebo LiDARu v uvažovaném způsobu ovládání nedokáže detektovat.

Problém s rychlosťí odezvy GPS je samozřejmě řešitelný investicí do GPS modulu s vyšší obnovovací frekvencí. V tomto momentě ovšem bylo jasné, že řízení vozidla pouze pomocí GPS je

v nejlepším případě velmi nepraktické, v nejhorším případě velmi nebezpečné pro vozidlo i okolí. Tento problém tedy zůstal neřešen a bylo rozhodnuto, že bude vyzkoušen jiný způsob řízení.

6.1.4 Směr natočení vozidla

Pro úspěšnou navigaci v prostoru je samozřejmě také nutné vědět, kterým směrem je vozidlo natočeno a kterým směrem se pohybuje. Na platformě ovšem nebyl umístěn kompas. Proto bylo třeba směr natočení vozidla určit jiným způsobem. Řídící kód musel být znovu upraven, tentokrát takovým způsobem, aby směr, kterým se vozidlo pohybuje, byl určován za využití změny v naměřených GPS souřadnicích.

Zde ovšem vyvstal nový problém vycházející z nedokonalosti a nepřesnosti GPS. GPS nedokáže zaměřit pozici se 100% přesností, vždy dojde k naměření s určitou odchylkou, která musí být při práci s naměřenými souřadnicemi započítána. Tato odchylka může vozidlo na mapě umístit jinam, než kde ve skutečnosti je. Při vysokých rychlostech, kdy dochází k velkým změnám souřadnic, je odchylka relativně k absolutní změně zanedbatelná a odchylka mezi reálným a vypočítaným směrem pohybu dosahuje pouze zanedbatelných hodnot. Vozidlo, jež je subjektem této práce, ovšem dosahuje pouze relativně nízkých rychlostí, převážně kvůli bezpečnosti provozu. V nízkých rychlostech, ve kterých absolutní změna pozice dosahuje nízkých hodnot, je odchylka způsobena nepřesností GPS relativně o hodně vyšší, již není zanedbatelná, a odchylka mezi reálným a vypočítaným směrem pohybu je schopná dosahovat hodnoty až 180° . Hodnoty s tak vysokými odchylkami jsou samozřejmě v praxi nevyužitelné. Problematice nepřesnosti GPS se podrobněji věnuje sekce 6.1.2 – Nepřesnost GPS.

Dalším problémem, který s sebou tento přístup určení směru přinesl, byla nemožnost určit směr natočení vozidla, když se vozidlo nehýbalo. Tento problém měl nežádoucí projevy zejména po startu vozidla, kdy nebylo možné určit, kterým směrem musí platforma jet, aby dosáhla cíle, a zda potřebuje zatáčet. Problém byl řešen obyčejným rozjetím autonomní platformy rovně dopředu. Z naměřeného pohybu byl dopočten směr, a teprve poté došlo k umožnění zatáčení v řídícím softwaru. V průběhu jízdy pak lze tento problém odstranit pamatováním si posledního naměřeného směru před případným dočasným zastavením vozidla.

Samotný výpočet směru pohybu vozidla je velmi jednoduchý. Za pomocí naměřených GPS souřadnic zjistíme, jaký pohyb vozidlo vykonalo. Z předchozí a nově naměřené hodnoty vyčteme rozdíl v zeměpisné šířce i délce. Pomocí knihovny *geopy* převedeme rozdíl uvedený v zeměpisné šířce a délce na metry a převedeme tak globální světové souřadnice na souřadnice kartézské. To nám umožní využít goniometrické funkce a zjednodušit výpočty, viz sekce 6.1.5 – Práce s 2D prostorem. Dále za využití Pythagorovy věty vypočteme celkovou vzdálenost, kterou vozidlo překonalo. Tímto získáme pravoúhlý trojúhelník, jehož přepona je dráha pohybu vozidla, odvěsny poté reprezentují rozdíly v zeměpisné šířce a v zeměpisné délce. Následně stačí rozdíl v zeměpisné šířce uvedený v metrech vydělit celkovou vzdáleností. Touto operací získáme kosinus úhlu. Na získaný kosinus aplikujeme funkci arkus kosinus, tedy funkci kosinu inverzní, čímž získáme úhel v rozsahu prvního

a druhého kvadrantu. Jelikož však úhel může patřit do třetího nebo čtvrtého kvadrantu kruhu, je třeba zkontrolovat, jestli se vozidlo pohybovalo směrem na západ. Jestliže je nová lokalita vozidla západně od původní pozice, autonomní platforma provedla pohyb vlevo v kartézských souřadnicích. V takovém případě pouze stačí odečíst vypočítaný úhel od úhlu 360° . Tak získáme směr, kterým se v kartézských souřadnicích vozidlo pohybuje a zároveň kterým směrem je momentálně natočeno. Výpočet požadovaného směru pohybu je samozřejmě analogický, pouze se při výpočtu jako vstupy využijí aktuální pozice vozidla a souřadnice cílového waypointu.

Na základě vypočtených směrů je poté vypočítáno správné natočení kol požadované k nasměrování vozidla směrem k cíli. Výpočet požadovaného natočení kol není složitý, jakmile známe směr, kterým se vozidlo momentálně pohybuje, a směr, kterým by se vozidlo mělo pohybovat, aby dosáhlo svého cíle. Prostým odečtením těchto směrů od sebe získáme odchylku mezi požadovaným a aktuálním směrem pohybu. Natočení kol následně může mít vůči rozdílu směrů lineární závislost, případně lze nastavit krokové změny při určitých hodnotách odchylky. Vyzkoušeny byly obě možnosti, v současnosti je využívána varianta s využitím krokových změn při předdefinovaných velikostech odchylky.

6.1.5 Práce s 2D prostorem

Jak již bylo předestvěno dříve, v této práci se často převádí geografické souřadnice na souřadnice kartézské. Díky elipsoidní nátuře tvaru planety Země totiž nemusí jeden stupeň zeměpisné šířky odpovídat jednomu stupni zeměpisné délky relativně k vzdálenosti vyjádřené v jednotkách soustavy SI. Navíc mezi vzdáleností vyjadřovanou jedním stupněm zeměpisné šířky a jedním stupněm zeměpisné délky neexistuje univerzální poměr, neboť se poměr mezi těmito hodnotami mění na základě aktuální zeměpisné šířky.

Samozřejmě univerzitní kampus, pro který je autonomní platforma stavěna, není rozlohou nijak veliký, proto se nabízela možnost pracovat s geografickými souřadnicemi jako by byly ekvivalentní souřadnicím kartézským, a předpokládat, že chyba způsobená tímto zjednodušením bude zanedbatelná. Problémy s touto domenkou ovšem byly odhaleny velmi rychle. Velikost chyby často přesahovala rozumné hodnoty. Projev chyby bylo možné zachytit nejen při testování softwaru, ale také při pouhém umístění vlastních značek v libovolné veřejné mapové službě (při práci na projektu byly využívány služby OpenStreetMap [39] a Mapy.cz [37]). Samozřejmě bude umístění značek opticky zkreslené, protože dochází ke kreslení na 2D projekci sférické planety. Toto zkreslení vytváří optickou chybu, jež nebude mít na samotné výpočtyliv, i tak lze ale velmi dobře vidět nepřesnost námi využitého zjednodušení. Při umístění bodů na mapu tak, aby hodnoty geografických souřadnic tvorily čtverec v souřadnicích kartézských, tedy dva sousedící body mají vždy jednu souřadnici stejnou, a rozdíl lišící se geografické souřadnice byl pro všechny čtyři strany stejný, je již na první pohled lze vidět, že body tvoří lichoběžník, ne čtverec, ačkoliv je lichoběžník také zkreslený právě projekcí glóbu do 2D prostoru.

Po odhalení nepřesnosti tohoto zjednodušení bylo tedy rozhodnuto, že bude vyzkoušen jiný způsob zjednodušení výpočtu. Ačkoliv by totiž korektní práce s geografickými souřadnicemi elipsoidní planety byla nejpřesnější, pro relativně malý univerzitní kampus, který lze navíc v řídícím softwaru považovat za 2D plochu, je implementace tohoto řešení zbytečně časově náročná s ohledem na relativně nízký benefit v daném měřítku. Proto momentální verze softwaru převádí geografické souřadnice do lokálního kartézského systému. Tento způsob se prokázal jako dostatečně přesný v měřítku kampusu Vysoké školy báňské, a ačkoliv samozřejmě není dokonalý a přináší s sebou také určitou chybu, díky využití lokálního kartézského systému a stále relativně nevelikému měřítku je chyba přijatelně nízká.

Převod do lokálního souřadnicového systému je řešen způsobem předestřeným dříve v sekci 6.1.4 – Směr natočení vozidla. Dva geografické body se považují za úhlopříčku obdélníku. Délky stran obdélníku vypočteme aplikací funkce *geopy.distance.distance*. Dva body obdélníku již samozřejmě známe, zbylé dva body získáme tak že vezmeme zeměpisné šířky známých bodů, a zkombinujeme je se zeměpisnými délkami opačného bodu.

Vypočítané délky stran nám tak definují vektor, jenž odpovídá úhlopříčce obdélníku. Následně využijeme faktu, že obdélník lze definovat také kombinací bodu a vektoru. Jeden vrchol obdélníku umístíme do bodu (0, 0). Když k tomuto bodu poté přičteme vypočtený vektor, získáme opačný vrchol. Tyto vrcholy nám nyní reprezentují geografické souřadnice v lokálním kartézském systému.

Délku úhlopříčky následně můžeme spočítat buď aplikací Pythagorovy věty, nebo funkcí *geopy.distance.distance*. Využití zmíněné funkce z knihovny *geopy* bude pravděpodobně přesnější, Pythagorovu větu ovšem naopak můžeme aplikovat i v případě, kdy neznáme původní geografické body, které byly převedeny do kartézského souřadnicového systému. Takový případ může nastat třeba po zavolání funkce, jež jako své argumenty přijímá pouze kartézské souřadnice. Zdrojový kód funkce, ale ani funkce nebo metody touto funkcí volané již k původním geografickým souřadnicím nemají jak přistoupit.

Nespornou výhodou kartézského souřadnicového systému je jednoduchost jeho využití. V kartézském souřadnicovém systému lze také bez jakýchkoliv problémů aplikovat funkce sinus a kosinus, které jsou kritické při práci s úhly. Dále na počtech s kartézskými souřadnicemi často není nic složitého. Díky jejich přímočarosti tak zdrojový kód zůstává jednoduchý, čitelný, vývoj je rychlejší a také je sníženo riziko výskytu chyb.

Při navigaci vozidla, zejména při práci s mapovými podklady, je univerzitní kampus považován za 2D plochu. Výškové rozdíly napříč areálem jsou dostatečně nízké, aby se daly považovat za zanedbatelné. Při navigaci tedy není třeba plánovat cestu tak, aby vozidlo nemuselo podstupovat dlouhé stoupání do kopce a zbytečně plýtbat energií. Občasné krátké stoupání pak může způsobit zpomalení pohybu vozidla, za takovým účelem je ale na ovládacím serveru běžícím na autonomní platformě samotné implementován PI regulátor, který právě v takovémto případě zaznamenaný pokles rychlosti vykompenzuje zvýšením okamžitého kroutivého momentu poskytovaného elektromotorem ve snaze

udržet rychlosť na požadované hodnotě. PI regulace je probrána v sekci 3.3 – Regulace PI regulátorem.

6.2 Řízení vozidla pomocí detekce grafických kódů

Narozdíl od řízení pomocí GPS dopadly pokusy o prototyp řízení za využití detekce grafických kódů úspěšněji. Obyčejné následování grafického kódu bylo s rozumnou přesností a úspěšností zprovozněno již při prvním testování na fyzickém vozidle. I přes relativní úspěch testování má však toto řešení také svá úskalí pramenící především z jednoduchosti řešení, proto není tento způsob řízení v praxi používán. Přesto výzkum a testování tohoto řešení, podobně jako v případě řízení pomocí GPS, přineslo nejen řadu poznatků, ale také technické zázemí a využitelný zdrojový kód.

6.2.1 Implementace řízení

Při implementaci tohoto druhu řízení byla na vozidlo upevněna obyčejná webkamera. Vstup z webkamery analyzoval program nezávislý na řídícím softwaru, který je subjektem této práce. Analyzační software společně s kamerou byly kalibrovány na určitou velikost grafického kódu. Po správné kalibraci dokázal software správně určit vzdálenost kódu od vozidla na základě jeho nakonfigurovaných rozměrů. Dále dokázal software spočítat také vzdálenost od středu obrazu, jímž procházela osa vozidla. Tato hodnota byla následně využita pro výpočet natočení kol vozidla. Software pro analýzu obrazu ovšem není předmětem této práce.

Software analyzující vstup z kamery implementuje obyčejný TCP server, jež na požádání vrací údaje odečtené z webkamery. Ve své odpovědi vrací server vzdálenost detekovaného kódu od vozidla a vzdálenost detekovaného kódu od středu obrazu. Řídící software si opakovaně o tyto hodnoty žádá a na jejich základě počítá odchylku mezi směrem, kterým vozidlo momentálně směruje, a úhlem, pod kterým na grafický kód nahlíží. Natočení kol je následně nastaveno na hodnotu této odchylky, nejvýše však 20° , tedy maximální úhel natočení přední soupravy, aby autonomní platforma vždy směřovala nejkratší cestou k detekovanému kódu. Aby nedošlo ke kolizi mezi vozidlem a objektem, na kterém je upevněn grafický kód, je software naprogramován tak, aby vozidlo zastavil v momentě, kdy je detekovaná vzdálenost kódu menší nebo rovna dvěma metrům.

Výpočet odchylky mezi osou vozidla a pozicí kódu je opět velmi jednoduchý. Představme si trojúhelník, jehož tři vrcholy jsou objektiv kamery, střed projekce a grafický kód. V takovém trojúhelníku bude přeponu představovat úsečka mezi objektivem kamery a grafickým bodem. Odchylku samotnou představuje úhel mezi přeponou a úsečkou spojující střed projekce. Délku této strany trojúhelníku neznáme, známe ovšem délky zbylých dvou stran, a víme, že je daný trojúhelník pravoúhlý. Pro výpočet úhlu tedy můžeme využít poměr přepony ku protilehlé straně. Délkou protilehlé strany je zde vzdálenost bodu od středu projekce, délku přepony je vzdálenost bodu od vozidla. Samotný

poměr nám představuje sinus hledaného úhlu. Na tento poměr aplikujeme funkci arkus sinus, funkci inverzní k funkci sinus, a získáme hledanou odchylku.

6.2.2 Výsledek testování

Řešení se při testování projevilo jako funkční, ovšem omezené a v praxi hůře použitelné. Jako hlavní problém samozřejmě vyvstává absence analýzy okolí vozidla. Stejně jako při řízení pomocí GPS, i zde si vozidlo není vědomo svého okolí. Mohlo by tak snadno dojít k situaci, kdy autonomní platforma ohrozí zdraví návštěvníků univerzitního kampusu, nebo poničí majetek jejich či majetek univerzity. Pokud by mělo dojít k reálnému využití takto řízeného vozidla, analýza okolního prostředí samozřejmě bude nezbytná. Je však třeba zmínit, že v případě testování se jednalo teprve o první prototyp, jenž měl sloužit k vyzkoušení daného způsobu řízení. Cílem prvního testování samozřejmě nebylo vytvořit plně funkční systém.

Jako druhý, a z hlediska využití vozidla také podstatně závažnější, problém, se nabízí omezená flexibilita řešení. Zatímco při řízení pomocí GPS stačí pro změnu cesty autonomní platformy pouze jinak nadefinovat waypoints, případně aktualizovat digitální mapové podklady, v případě řízení pomocí grafických kódů je třeba fyzicky rozmístit grafické kódy tak, aby software vždy věděl, kde má v aktuální situaci platformu směřovat. Grafické kódy musí být vidět, nesmí být například blokovány zaparkovaným vozem, poničené cizí osobou nebo vlivem stáří či počasí, apod. Dále lze tímto způsobem nadefinovat pouze jednu cestu, aby software vždy dokázal deterministicky zvolit správný směr pohybu vozidla. Posledním projevem neflexibility daného způsobu řízení je také nutnost fyzicky přesunout grafické kódy při každé změně požadované cesty.

Za problém lze považovat také fakt, že vozidlo při detekci kamery směruje přímo k nejbližšímu kódu. Kódy samozřejmě musí být rozmištěny mimo silnici či chodník, aby neblokovaly cestu. Ovšem kódy umístěné mimo silnici dokážou vozidlo svést z cesty, v lepším případě pouze na trávník, v horším případě jej navedou přímo do nečekané překážky. Tento problém však je řešitelný například vynucením určitého odstupu od kódu a udržováním kódu v určité vzdálenosti od boku vozidla. V tomto případě by ale všechny kódy musely být umístěny pouze po jedné straně vozidla. Druhým možným řešením je převod detekovaného grafického kódu na waypoint na mapě. Toto řešení ovšem vyžaduje využití mapových podkladů a vstupů z GPS pro řízení platformy, a zabíhá již do podkapitoly 6.3 – Řízení vozidla kombinací vstupů z GPS a kamery a za využití mapových podkladů

6.2.3 Možné využití v praxi

I přes určitá omezení tohoto způsobu řízení ovšem není možné daný způsob prohlásit za nepoužitelný. Ačkoliv je nevhodný při navigaci kampusem pomocí předem neznámé tras, může skvěle posloužit například pro přesné zaparkování vozidla po dosažení cíle. K parkování lze samozřejmě využít například také horizontální značení parkovacího místa. Očekává se ovšem, že autonomní platforma bude objíždět velké množství budov spadajících pod kampus Vysoké školy báňské. V ta-

kovém případě ovšem nemusí vždy jít zaparkovat, například z důvodu obsazenosti parkovacích míst nebo z důvodu velké vzdálenosti nejbližšího parkovacího místa od budovy. Toto by mohlo způsobit problémy třeba při případné implementaci automatizovaného nakládání a vykládání vozidla. Navíc možnost vytvářet nová parkovací místa rezervovaná pouze pro autonomní výzkumná vozidla není díky nutnosti tvorby nového vodorovného značení praktická.

Jako alternativa se v takovéto situaci nabízí právě využití grafického kódu, který stačí pouze vytisknout ve správné velikosti a umístit na vertikální plochu. Vozidlo by po detekování kódu mohlo přesně zaparkovat v předdefinované vzdálenosti od kódu. Takto by bylo možné přesně a relativně jednoduše dovést vozidlo až do přesně určené finální pozice. Tato funkctionalita dokáže umožnit nejen dříve zmíněné robotizované nakládání a vykládání autonomní platformy, jež má sloužit k rozvozu materiálu napříč kampusem, ale také přesné zaparkování v garáži nebo jiných vnitřních prostorech, ve kterých nelze využít technologii GPS, zejména díky nedostatku pokrytí.

6.3 Řízení vozidla kombinací vstupů z GPS a kamery a za využití mapových podkladů

Jasným výsledkem předchozího výzkumu tedy je nutnost implementace řízení za pomocí kombinace více vstupů. Tato implementace tak bude kombinovat vyhledávání cesty pomocí mapových podkladů k nalezení správné cesty a vstupy ze soustavy kamer a LiDARů k zajištění bezpečnosti provozu, vyhýbání se překážkám, udržování správné trajektorie, aby vozidlo např. odchylkou GPS nesjelo z vozovky, ale také k vyhledávání zatáček a křižovatek, na kterých poté pomocí mapových podkladů proběhne rozhodnutí o následujícím pohybu autonomní platformy, aby bylo korektně dosaženo cíle. Cíle budou definovány waypoints. Při vývoji bude využito nejen poznatků, ale také kódů z předchozího výzkumu.

Bohužel, vývoj tohoto způsobu autonomního řízení je náročný a navazuje na předchozí výzkum, projekt bude ve vývoji ještě nějakou dobu a přesahuje rozsah této publikace. Navíc byla práce na projektu autonomního řízení negativně ovlivněna pandemií koronaviru SARS-CoV-2 [2] a stoupající cenou kryptoměn [4]. Pandemie koronaviru ztěžila testování vozidla a práci s fyzickým vozidlem vynucením práce z domova, způsobila silný nedostatek webkamer, které byly na projektu potřeba kvůli snímání okolí, a kombinace vysokého zájmu těžařů o grafické karty a snížené produkce grafických karet nebo počítačových čipů znemožnila získání hardwaru potřebného pro implementaci plně autonomního řízení [5, 6]. Práce však bude do budoucna stále pokračovat.

Kapitola 7

Práce s mapovými podklady

Při práci s mapovými podklady jsou využity hned dvě mapové služby, Mapy.cz [37] společnosti Seznam.cz, a.s., a komunitní mapový projekt OpenStreetMap [39]. Mapy.cz jsou využívány především pro své grafické podklady. Dále je využíváno API Mapy.cz [25] pro implementaci webové aplikace. Důvody využití služby Mapy.cz souvisí především s extenzivním, avšak jednoduchým API, ale také s jejím českým původem. Vývojáři Mapy.cz spolupracují přímo s českými úřady a často tak dokážou zaznamenat změny v silniční síti rychleji než zahraniční nebo komunitní mapové služby. Drobou roli hraje také vlastenecký sentiment. OpenStreetMap poté slouží jako poskytovatel veřejně dostupných vektorových mapových podkladů.

Při vývoji projektu byl věnován čas výběru správnému zdroji vektorových mapových podkladů. Mapy.cz působily jako primární kandidát na zdroj podkladů, a to především díky svých již dříve zmíněných časnějších aktualizací na základě změn v silniční síti. Při studování dokumentace API Mapy.cz byl však objeven znepokojující fakt – Mapy.cz veřejně poskytuje pouze grafické podklady. Na dotaz, zda je možné využít vektorové podklady společnosti Seznam.cz, a.s., bylo odpovězeno, že v současné době jsou tyto podklady používány pouze interně a existují pouze v binárním formátu. Společnost Seznam.cz, a.s. vyjádřila ochotu exportovat a poskytnout svá data pro akademické účely, ovšem pro jednoduchost využití bylo rozhodnuto, že budou využity podklady poskytované projektem OpenStreetMap.

7.1 Stahování mapových podkladů a jejich zpracování

Projekt OpenStreetMap, zvolený poskytovatel mapových podkladů, umožňuje data exportovat mimo jiné také ve formátu XML [26, 20]. Pro získání souboru stačí vytvořit obyčejný HTTP GET [16] požadavek a požadovanou exportovanou oblast specifikovat v parametrech URI. Server v odpovědi vrátí soubor s mapovými podklady. Soubor lze takto jednoduše získat například pomocí programu *curl* [38].

Pro stahování a export mapových podkladů byl vytvořen separátní skript implementovaný v jazyce Python. Tento skript dokáže stáhnout soubor s mapovými podklady, zpracovat jej, uložit zpracovaná data ve vlastním formátu a následně smazat soubor s původními mapovými podklady ve formátu XML [20].

Implementace separátního skriptu pro získání mapových podkladů s sebou přinásí určité množství výhod. Skript je možné periodicky spouštět jako cronjob [22], ať už přímo v operačním systému, nebo například jako kontejner v Kubernetes [12]. Dále je možné část zpracování podkladů převést z plánovacího softwaru do separátního programu, a tím zjednodušit logiku samotného plánovacího softwaru. A nakonec, využití vlastního formátu pro uložení dat nám umožní jednodušeji přidat do souboru vlastní data a dodatečný kontext, viz sekce 9.2 – Chytřejší práce s mapovými podklady.

K načtení textového XML souboru je využita knihovna *osmium*. Z XML souboru jsou přečteny uzly a cesty. Relace jsou ignorovány. Při zpracování cest jsou uloženy pouze cesty povolených typů, ostatní typy cest jsou ignorovány. Mezi ignorované typy patří např. *footway*, *corridor* nebo *steps*. Podklady OSM často obsahují například chodníky či chodby, je ovšem velmi nežádoucí, aby se vozidlo pokoušelo například projíždět chodbami v interiérech budov nebo vyjíždět do schodů. Po načtení všech cest jsou nakonec z mapových podkladů odstraněny také uzly, ke kterým nevede žádná cesta. Uzly v reálném světě představují například křižovatky nebo spojení chodeb. Ovšem v případě například uzlu nacházejícího se v interiéru budovy, tedy spojení chodeb v budově, není třeba uzel ukládat v paměti softwaru a dále s ním pracovat, neboť chodby jsou již dříve z podkladů odstraněny a uzel by neměl reálné využití. Zpracovaná data jsou dále skriptem naformátována a uložena ve formátu JSON [19].

7.2 Vyhledání nejbližší silnice na mapě

Vyhledání nejbližší silnice je důležité nejen za účelem určení aktuální pozice vozidla na mapě, ale také při umisťování waypointů, které se vždy umisťují na nejbližší definovanou silnici (viz sekce 5.1 – Definice cílů cesty vozidla). Autonomní platforma se samozřejmě pohybuje převážně po silnici, a plánovací software musí vědět, kde se vozidlo momentálně nachází i v momentě, kdy nepřesná GPS (viz sekce 6.1.2 – Nepřesnost GPS) umístí autonomní platformu mimo definovanou silnici. Dále musí plánovací software vědět, na které silnici se nachází daný waypoint, aby dokázal správně naplánovat cestu.

Cestu lze v mapových podkladech definovat dvěma způsoby. Počátečním a koncovým bodem, nebo bodem a vektorem určujícím směr a délku cesty. Oba způsoby definice cesty jsou využity při určování nejbližší silnice. Dále je uvažováno, že nejkratší cestou z libovolného bodu k libovolné silnici je úsečka kolmá k dané silnici, pokud taková úsečka existuje. V opačném případě je nejkratší cestou úsečka spojující daný bod s bližším koncem silnice.

Pro určení nejbližší silnice relativně k danému bodu na mapě využijeme výpočet průsečíků dvou přímek [7]. Jako první přímku uvažujeme přímku definovanou počátkem a koncem silnice. K zís-

kání druhé přímky je následně třeba vzít normálový vektor k vektoru definujícímu danou silnici. Získaný normálový vektor umístíme do uvažovaného bodu, tedy například do aktuální pozice vozidla nebo neupravené pozice uživatelem definovaného waypointu. Tímto získáme přímku kolmou k přímce, na níž leží uvažovaná silnice. Následně na získané přímky aplikujeme funkci výpočtu průsečíku přímek, a zkontrolujeme, zda se daný průsečík nachází na úsečce představující silnici. Pokud ano, nejkratší cestu z uvažovaného bodu k dané silnici definujeme jako úsečku z uvažovaného bodu do průsečíku přímek. Pokud ne, nejkratší cesta bude cesta z daného bodu k blížšímu z okrajových bodů silnice. Tento algoritmus následně stačí v cyklu aplikovat na všechny silnice definované v mapových podkladech, a takto nalézt silnici, ke které vede nejkratší cesta. Tímto získáme nejen nejbližší silnici, ale také bod na dané silnici, ve kterém se nachází vozidlo nebo ve kterém by měl být umístěn waypoint.

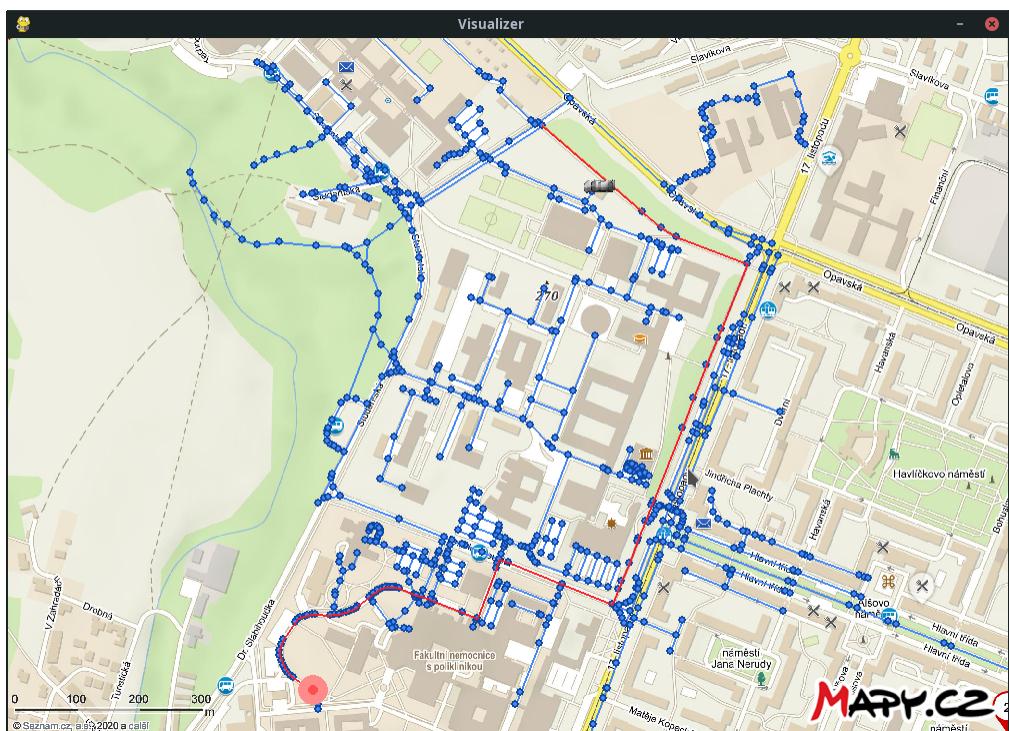
7.3 Plánování cesty za využití mapových podkladů

Ve finální verzi softwaru budou mapové podklady sloužit jako jeden ze vstupů při plánování cesty. Současné návrhy budoucí implementace počítají s nalezením optimální cesty v mapových podkladech a jejím následováním za využití vstupů z kamer, GPS, a v budoucnu případně také z LiDARu. Kamery budou při analýze okolí mimo jiné detektovat také silnice. Jakmile dojde k detekci křižovatky, na řadu nastoupí právě GPS v kombinaci s mapovými podklady. Software za využití mapových podkladů určí, o kterou křižovatku se jedná, a kterým směrem má vozidlo pokračovat, a případně, zase pomocí vstupu z kamer či LiDARů, provede vozidlo zatáčkou.

Kamerový systém i software pro analýzu obrazu jsou v době psaní práce stále ve vývoji. Navigaci za využití grafických vstupů tak nebylo prozatím možné plně implementovat. Prozatím byl však alespoň implementován algoritmus sloužící k hledání optimální cesty areálem univerzity.

K účelu hledání nejkratší cesty univerzitním kampusem byl zvolen algoritmus prohledávání grafu do šířky (Breadth First Search) [21]. Důvodem volby tohoto algoritmu byla především jeho jednoduchost. Implementace průchodu grafu do šířky není složitá, zároveň však hledání cesty funguje dostatečně rychle. Algoritmus v dané implementaci nevyužívá heurestiku, jež na malém grafu není nutná. Zároveň nebene v potaz délky jednotlivých cest, pouze počet cest detekovaných při průchodu grafem. Mapu univerzity tedy lze považovat za nevážený graf. Práce s délkami cest by bezpochyby byla vhodná v provozu v předem neznámých oblastech, v kampusu univerzity při současné implementaci ovšem není třeba – její benefit by byl zanedbatelný až nulový.

Práce s váženým grafem však v budoucnu může reálně být potřeba, a to v případě, pokud opravdu dojde k definici váhy cesty na základě její frekventovanosti v dané denní době, za předpokladu, že tato váha bude numerická (koeficient), ne binární (cesta je buď průjezdná, nebo neprůjezdná), a dojde tak k násobení délky dané části silnice v podkladech definovaným koeficientem. Tato funkcionalita je momentálně uvažována pouze v teoretické rovině, při její možné implementaci však nebude současný algoritmus průchodu neváženým grafem stačit.



Obrázek 7.1: Nejkratší cesta nalezená za využití algoritmu BFS (Zdroj mapových podkladů: Seznam.cz, a.s., zdroj grafiky vozidla viz [43])

Kapitola 8

Webová aplikace

V kapitole 5.1 – Definice cílů cesty vozidla bylo předestřeno využití webové aplikaci ke vzdálenému ovládání vozidla. V současné době lze autonomní platformu ovládat pomocí vizualizéru, jenž je součástí plánovacího softwaru. Toto řešení je ovšem dočasné a slouží především k ulehčení vývoje a zachytávání chyb. Při skutečném využití by bylo přinejlepším nepraktické. Využívání vizualizéru, mimo své negativní dopady na nároky ovládacího softwaru díky nutnosti vykreslovat vozidlo na mapu a vyobrazovat jeho pohyb, znemožňuje kontejnerizaci a automatickou orchestraci [12] softwaru pro plánování cesty a vynucuje spuštění ovládacího softwaru na zařízení, ze kterého chceme autonomní platformu ovládat. Za další nevýhodu lze považovat nutnost instalovat interpreter jazyka Python na cílovém stroji nebo daný interpreter přibalit k distribuci plánovacího softwaru.

Webová aplikace je v dnešní době již víceméně standardním řešením, jež řeší všechny dříve zmíněné problémy. Software pro plánování cesty vozidla může běžet nezávisle na uživatelském rozhraní v kontejnerizovaném prostředí a vystavovat své REST API pod nakonfigurovaným portem. Backend webové aplikace může běžet na stejném stroji či clusteru jako plánovací software, nebo může být spuštěn kdekoli jinde. Uživatel poté webovou aplikaci může jednoduše využívat za pomocí libovolného moderního webového prohlížeče. Prohlížeč si od serveru vyžádá statické soubory a vykreslí webovou aplikaci, jež následně očekává vstup od uživatele a zároveň graficky vyobrazuje aktuální informace o vozidle. Uživateli tak stačí mít k dispozici libovolné zařízení s moderním prohlížečem a přístupem k síti. Není třeba instalace žádného softwaru, může být využito libovolné zařízení, logika plánování cesty a logika grafického vyobrazování informací o vozidle je od sebe oddělená, a všechny potřebné aktualizace jsou proveditelné na straně backendu.

8.1 Implementace frontendu webové aplikace

Pro práci s mapovými podklady je využíváno API mapové služby Mapy.cz [37] (viz kapitola 7 – Práce s mapovými podklady), jež umožňuje nejen vykreslování vozidla a waypointů, ale také registraci interakce s mapou a implementaci ovládání vozidla a vytváření waypointů. Webová stránka je

samozřejmě definována značkovacím jazykem HTML a logika webové aplikace je naprogramována v jazyce Javascript. Knihovna služby Mapy.cz je jediná závislost využitá v implementaci frontendu webové aplikace, s její výjimkou je využit pouze čistý Javascript.

Frontend webové aplikace po svém načtení prohlížečem začne periodicky obnovovat aplikaci pomocí HTTP requestu na backend aplikace. Po přijetí odpovědi serveru jsou automaticky aktualizovány mapové podklady. Dále frontend definuje vytváření a mazání waypointů, jež definují cíl cesty vozidla. Po kliknutí na prázdné místo na mapě je poslan HTTP dotaz na API serveru. Server následně notifikuje plánovací software o uživatelově snaze vytvořit waypoint. Software pro plánování cesty waypoint vytvoří na nejbližší cestě definované v mapových podkladech vozidla. Po vytvoření waypointu je vynucena aktualizace mapových podkladů, aby se nově vytvořený waypoint promítl na mapě. Kliknutím na waypoint je poté možné daný waypoint smazat. Mazání waypointu je provedeno způsobem analogickým k vytváření waypointu, pouze je volán jiný endpoint backendu.

8.2 Implementace backendu webové aplikace

Pro tvorbu backendu webové aplikace byl zvolen programovací jazyk Kotlin spárovaný s frameworkem *Ktor* [40]. Důvod volby těchto technologií byla snaha ukázat modularitu systému skládajícího se z více komponent využívajících protokol HTTP pro komunikaci a možnost využití libovolné technologie a jazyka v takto nadefinovaném systému. Pro uvažovanou aplikaci by samozřejmě jazyk Python byl naprostě validní volba, v praxi je ovšem často vhodné využití většího množství jazyků a technologií, především díky lišících se předností různých jazyků, nebo dostupnosti knihoven či frameworků pro daný jazyk nebo platformu. Druhým důvodem volby jazyka Kotlin byla snaha vyzkoušet využití frameworku *Ktor* v praxi.

Programovací jazyk Kotlin je moderní objektově orientovaný jazyk s prvky funkcionálního programování. Jazyk Kotlin je tvořen firmou JetBrains a je možné jej kompilovat pro více platform. Primárně je jazyk Kotlin kompilován pro platformu JVM, kde tak může využívat celý ekosystém jazyka Java. Dále však může být kompilován do nativního strojového kódu nebo transpilován do jazyka Javascript. Jazyk Kotlin v současné době slouží jako oficiální jazyk pro tvorbu aplikací mobilního operačního systému Android [36].

Framework *Ktor* je technologie implementovaná v jazyce Kotlin sloužící k implementaci asynchronních REST API [40]. Samotné API webové aplikace ovšem není vyvíjeno s důrazem na asynchronicitu, neboť požadavky na výkon rozhraní nikdy nebudou tak vysoké, aby odůvodnily čas strávený implementací asynchronního programu.

Backend definuje REST API [13], jež, kromě vydávání statických souborů, implementuje endpointy analogické REST API softwaru pro plánování cesty popsaném v podkapitole 5.3 – REST API. Rozdíl mezi API plánovacího softwaru a API backendu webové aplikace je ve validaci prováděné daným API. API plánovacího softwaru z důvodu jednoduchosti implementuje pouze minimální validaci, slouží pouze k internímu použití, a nemělo by tedy být vystaveno veřejně. REST API bac-

kendu webové aplikace je určeno k veřejnému použití, a za tímto účelem také implementuje určitou validaci svých vstupů. Dále také z hlediska logického dělení funkcionality systému není vhodné, aby software pro plánování cesty vozidla uměl vydávat statické soubory.

Kromě frameworku *Ktor* jsou na backendu využity také knihovny *org.json:json* [41] sloužící k práci s plaintextovým formátem JSON [19] a *io.github.rybalkinsd:kohhttp* [42] implementující DSL pro jazyk Kotlin sloužící k vytváření HTTP dotazů.

Kapitola 9

Budoucí vývoj vozidla

Jak již bylo zmíněno dříve, práce na projektu, jemuž se věnuje tato práce, bude dále pokračovat. V současné době je třeba dokončit především autonomní řízení a navigaci vozidla. Dále se však nabízí velká škála vylepšení projektu, aby bylo umožněno platformu využívat v reálném provozu. Tato rozšíření a vylepšení nejsou v momentální době uvažována zejména z důvodu časových restrikcí.

9.1 Vstup z LiDARu

Pro zpřesnění ovládání a zlepšení analýzy okolí by bylo vhodné vstup ze soustavy kamer doplnit soustavou LiDARů. Soustava LiDARů umožní softwaru lépe určit např. vzdálenosti překážek. Většina analýzy vstupu z LiDARu by ovšem s nejvyšší pravděpodobností byla provedena na stejném serveru, jenž analyzuje vstup z kamer. Server by informace zpracoval a na svém vstupu vrátil pouhou kompliaci již zpracovaných dat. Kód, jímž se zabývá tato práce, by tak nejspíše nebylo třeba příliš upravovat.

9.2 Chytřejší práce s mapovými podklady

Po univerzitním kampusu se na denní bázi pohybuje velké množství lidí. Vysoká koncentrace lidí může pohyb vozidla zpomalit, zároveň však vozidlo může představovat nebezpečí pro zdraví osob v jeho okolí. Ačkoliv je autonomní platforma navrhována s vysokým důrazem na bezpečnost, bylo by pro zvýšení bezpečnosti a zároveň také plynulosti provozu lepší, aby se vozidlo vyhýbalo frekventovaným cestám. Při následném vývoji řídícího softwaru by bylo možné do souboru s mapovými podklady zakódovat také informace o frekventovanosti silnice. Tato informace by následně mohla posloužit při vyhledávání cesty, aby vozidlo využívalo cesty méně využívané, a ještě více tak bylo zmírněno jakékoli riziko spojené s provozem vozidla.

9.3 Metriky, chytřejší logování, monitorování

Automatizované monitorování logů [10] a metrik [9] může v praxi signifikantně usnadnit provoz a ladění softwaru. Za účelem automatizace monitorování aplikací proto také vznikla velká spousta nástrojů, mezi něž se řadí například Prometheus [9], Grafana [9], Kibana nebo Logstash [10]. V projektu autonomního řízení by bylo vhodné implementovat automatizované monitorování právě pomocí těchto nástrojů, ideálně dříve, než dojde k využití vozidla v reálném provozu.

Extenzivní logování zpráv ve formátu JSON by umožnilo zautomatizovat sledování logů nástrojem Kibana [10] a notifikovat vývojáře platformy v případě zvýšeného výskytu chyb. Sledování logů poté lze doplnit nástrojem Prometheus [9], jež je možné využít ke sběru programátorem poskytovaných metrik. Metriky sbírané Prometheem nejsou předem definované, vývojář metriky definuje sám, pomocí Promethea tak lze sledovat teoreticky cokoliv. Často se ovšem tato technologie používá například ke sledování počtu a trvání dotazů na externí komponenty, ke sledování počtu chyb, apod. Využití Promethea umožní například sbírat informace o počtu chyb na sběrnici CAN, délky trvání dotazů na server analyzující okolní prostředí, atd. Sesbírané metriky následně lze graficky zobrazit za využití technologie Grafana. Monitorování metrik umožní nejen odhalení chyb, ale také zjednoduší ladění výkonnostní stránky aplikace právě díky možnosti sledování délky trvání dotazů nebo výpočtů.

Kapitola 10

Závěr

Cílem této práce je výzkum možností autonomního řízení a jejich následná implementace. Při výzkumu byly vyzkoušeny dva alternativní způsoby řízení vozidla. Výsledkem jejich zkoumání však je zjištění, že bude nutné zvolit tradiční metodu kombinace většího množství vstupů.

Výstupem zkoušení různých alternativ není software schopný plně autonomního řízení – za tímto účelem se zkoušené alternativy projevily jako nedostatečné. Přesto byl při zkoušení napsán zdrojový kód, jenž bude možné využít při budoucím vývoji vozidla. Výstup v podobě zdrojových kódů představuje softwarový simulátor využitelný jako náhrada za fyzické vozidlo při testování autonomního řízení, software pro zadávání příkazů vozidlu pomocí rozhraní CAN, základ softwaru plánujícího cestu autonomní platformy, webová aplikace umožňující vzdálené zadávání cílů cesty a monitorování vozidla, utilita sloužící k periodickému stahování mapových podkladů, nebo také utilita pro zařízení využívající operační systém Android, jež lze využít k logování GPS souřadnic za účelem jejich následné analýzy nebo využití zaznamenaných souřadnic při testování jiných softwarových komponent.

Bohužel byla práce částečně zpožděna. Zpoždění bylo způsobeno převážně externími faktory, jež nepřijemným způsobem ztížily nebo znemožnily vývoj a testování nejen softwaru, ale celého vozidla. Čas, jenž nemohl být věnován vývoji autonomního řízení, byl však věnován tvorbě podpůrného softwaru, jako jsou skript pro export mapových podkladů nebo webová aplikace, které by jinak nemusely být vůbec vyvinuty. Dále je třeba zmínit, že tato práce, i přes veškerou snahu vyvinutou při tvorbě softwaru, nepřinesla nové poznatky na poli autonomního řízení vozidel. Dokončením vývoje softwaru pro autonomní řízení, jenž je subjektem této práce, tak bude vyvinuto vozidlo s určitým komerčním potenciálem, práce však nebude představovat posun v oblasti autonomních vozidel.

Zdroje

- [1] Tim Wescott: *PID Without a PhD*
Dostupné z: <https://www.wescottdesign.com/articles/pid/pidWithoutAPhd.pdf>,
Wescott Design Services, 2018
- [2] Md Tanveer Adil, Rumana Rahman, Douglas Whitelaw, Vigyan Jain, Omer Al-Taan, Farhan Rashid, Aruna Munasinghe, Periyathambi Jambulingam: *SARS-CoV-2 and the pandemic of COVID-19*.
Dostupné z: <https://pmj.bmjjournals.com/content/97/1144/110>, The Fellowship of Postgraduate Medicine, 2021
- [3] Steve Corrigan: *Introduction to the Controller Area Network (CAN)*
Dostupné z: <https://www.ti.com/lit/an/sloa101b/sloa101b.pdf>, Texas Instruments, 2016
- [4] Bloomberg: *Bitcoin Touches \$64,000 High as Traders Eye Coinbase Listing* [online, cit. 2021-04-17].
Dostupné z: <https://www.bloomberg.com/news/articles/2021-04-13/bitcoin-rallies-to-all-time-high-as-traders-eye-coinsbase-listing>, Bloomberg, 2021
- [5] Bloomberg: *Crypto Miners Are Competing With Gamers for Nvidia Chips* [online, cit. 2021-04-17].
Dostupné z: <https://www.bloomberg.com/news/newsletters/2021-04-14/crypto-miners-are-competing-with-gamers-for-nvidia-chips>, Bloomberg, 2021
- [6] Bloomberg: *The Global Auto Plants Now Idle as Chip Supplies Dry Up* [online, cit. 2021-04-17].
Dostupné z: <https://www.bloomberg.com/news/articles/2021-03-24/the-global-auto-plants-now-idle-as-chip-supplies-dry-up>, Bloomberg, 2021
- [7] Gabriel Eng: *Intersection of two lines in Python* [online, cit. 2021-04-17].
Dostupné z: <https://stackoverflow.com/questions/20677795/how-do-i-compute-the-intersection-point-of-two-lines/51127674#51127674>, Stack Overflow, 2018

- [8] Nyen Thin Li, Lau Ting, Nor Husna, Heikal Husin: *GPS Systems Literature: Inaccuracy Factors And Effective Solutions*,
 Dostupné z: https://www.researchgate.net/publication/300083844_GPS_Systems_Literature_Inaccuracy_Factors_And_Effective_Solutions, International journal of Computer Networks & Communications, 2016
- [9] KC Kecheng: *Performance Monitoring with Prometheus and Grafana*,
 Dostupné z: https://performance-monitoring-with-prometheus.readthedocs.io/_downloads/en/latest/pdf/, KC Kecheng, 2020
- [10] Pranita Bavaskar, Onkar Kemker, Aditya Sinha: *Log Analysis with ELK Stack Tool*,
 Dostupné z: https://www.researchgate.net/publication/343775739_A_SURVEY_ON_LOG_ANALYSIS_WITH_ELK_STACK_TOOL, Sandip University, 2019
- [11] S.A.I.B.S. Arachchi, Indika Perera: *Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management*,
 Dostupné z: https://www.researchgate.net/publication/326406017_Continuous_Integration_and_Continuous_Delivery_Pipeline_Automation_for_Agile_Software_Project_Management, University of Moratuwa, 2018
- [12] Vitor Silva, Marite Kirikova, Gundars Alksnis: *Containers for Virtualization: An Overview*
 Dostupné z: https://www.researchgate.net/publication/325534952_Containers_for_Virtualization_An_Overview, Riga Technical University, 2018
- [13] Otávio Freitas Ferreira Filho, Maria Alice Grigas Varella Ferreira: *Semantic Web Services: A RESTful Approach* Dostupné z: <https://github.com/otaviofff/restful-grounding/blob/master/papers/official-paper-icwi.pdf>, University of São Paulo, Polytechnic School, 2009
- [14] Mark Allman: *An Evaluation of XML-RPC* Dostupné z: <https://www.icir.org/mallman/pubs/A1103b/A1103b.pdf>, ACM Performance Evaluation Review, 2003
- [15] IEEE: *IEEE-754, Standard for Floating-Point Arithmetic* Dostupné z: <https://standards.ieee.org/standard/60559-2020.html>, IEEE, 2008
- [16] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee: *RFC2616: Hypertext Transfer Protocol – HTTP/1.1* Dostupné z: <https://www.rfc-editor.org/rfc/pdf/rfc2616.txt.pdf>, RFC Editor, 1999
- [17] Flask: *Flask – web development, one drop at a time* Dostupné z: <https://buildmedia.readthedocs.org/media/pdf/flask/latest/flask.pdf>, Flask, 2017

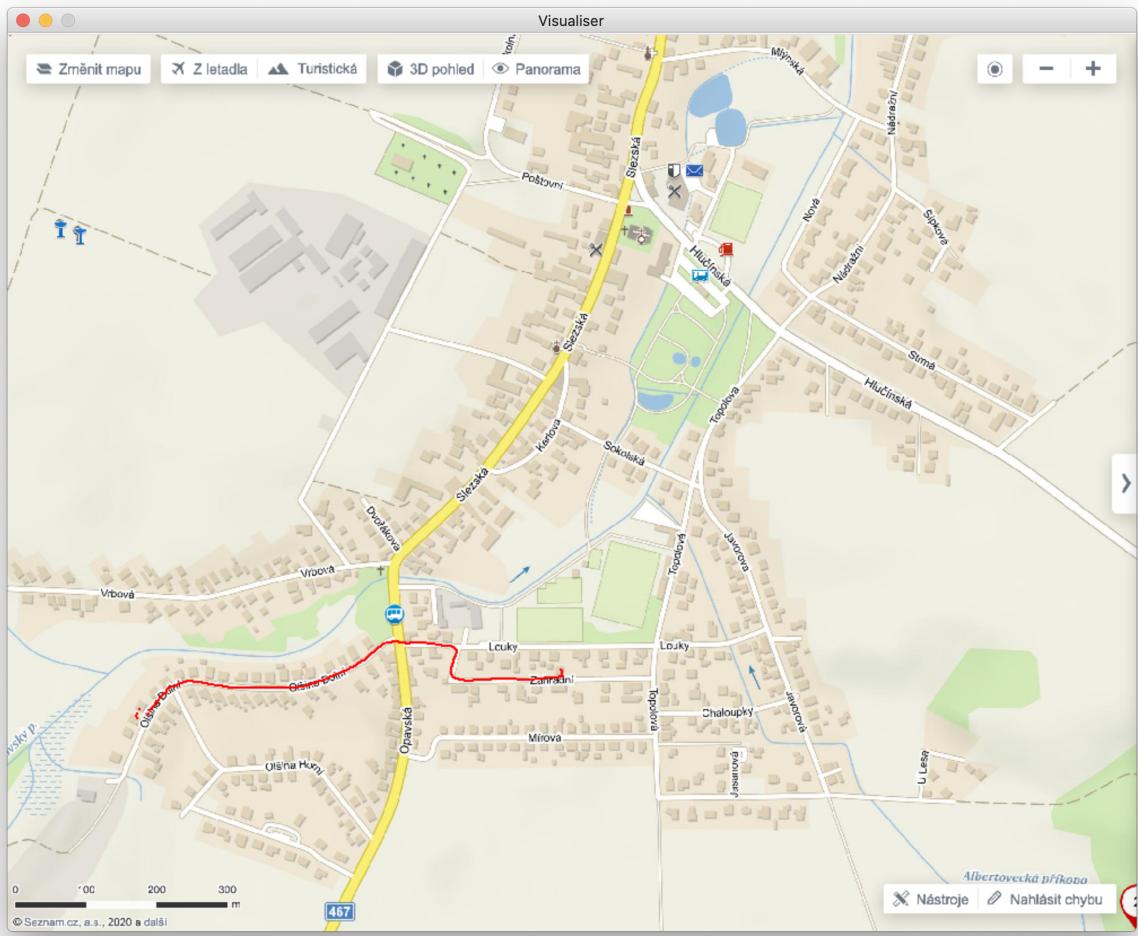
- [18] Oren Ben-Kiki, Clark Evans, Ingy döt Net: *YAML Ain't Markup Language (YAMLTM)*, Version 1.2 Dostupné z: <https://yaml.org/spec/1.2/spec.pdf>, Oren Ben-Kiki, Clark Evans, Ingy döt Net, 2009
- [19] ECMA International: *ECMA-404: The JSON Data Interchange Syntax* Dostupné z: https://www.ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf, ECMA International, 2017
- [20] World Wide Web Consortium: *Extensible Markup Language (XML) 1.0 (Fifth Edition)* Dostupné z: <https://www.w3.org/TR/2008/REC-xml-20081126/>, World Wide Web Consortium, 2008
- [21] Jason J Holdsworth: *The Nature of Breadth-First Search* Dostupné z: https://www.researchgate.net/publication/2727226_The_Nature_of_Breadth-First_Search, James Cook University, 1999
- [22] A. Kishore/Sachin: *Crontab: Everything You Want To Know* Dostupné z: http://www.appsdba.info/docs/oracle_apps/R12/Crontab.pdf, APPS DBA, 2011
- [23] Information Sciences Institute, University of Southern California: *RFC 793: Transmission Control Protocol* Dostupné z: <https://tools.ietf.org/pdf/rfc793.pdf>, Information Sciences Institute, University of Southern California, 1981
- [24] J. Postel: *User Datagram Protocol* Dostupné z: <https://tools.ietf.org/pdf/rfc768.pdf>, Information Sciences Institute, University of Southern California, 1980
- [25] Seznam.cz, a.s.: *Mapy.cz API* [online],
Dostupné z: <https://api.mapy.cz/>, Seznam.cz, a.s., 2021
- [26] Overpass: *Overpass API* [online],
Dostupné z: <https://dev.overpass-api.de/overpass-doc/en/>, Overpass, 2021
- [27] gRPC: *gRPC* [online],
Dostupné z: <https://grpc.io/>, gRPC Authors, 2021
- [28] GeoPy: *GeoPy* [online],
Dostupné z: <https://geopy.readthedocs.io/en/stable/>, GeoPy, 2021
- [29] Martijn Braam: *gpsd-py3* [online],
Dostupné z: <https://github.com/MartijnBraam/gpsd-py3>, Martijn Braam, 2017
- [30] Google Inc.: *Protocol Buffers - Google's data interchange format* [online],
Dostupné z: <https://github.com/protocolbuffers/protobuf>, Google Inc., 2021

- [31] OpenStreetMap: *PyOsmium – Python bindings to Osmium Library* [online],
Dostupné z: <https://osmcode.org/pyosmium/>, OpenStreetMap, 2021
- [32] PyGame: *PyGame* [online],
Dostupné z: <https://www.pygame.org/>, PyGame, 2021
- [33] Brian Thorne: *python-can* [online],
Dostupné z: <https://github.com/hardbyte/python-can>, Brian Thorne, 2021
- [34] Sebastián Ramírez: *FastAPI* [online],
Dostupné z: <https://fastapi.tiangolo.com/>, Sebastián Ramírez, 2021
- [35] Encode: *Starlette* [online],
Dostupné z: <https://www.starlette.io/>, Encode, 2021
- [36] Maxim Shafirov: *Kotlin on Android. Now official* [online, cit. 2021-04-29],
Dostupné z: <https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/>, JetBrains s.r.o., 2017
- [37] Seznam.cz, a.s.: *Mapy.cz* [online],
Dostupné z: <https://en.mapy.cz/>, Seznam.cz, a.s., 2021
- [38] Daniel Stenberg: *curl* [online],
Dostupné z: <https://curl.se/>, Daniel Stenberg, 2021
- [39] OpenStreetMap: *OpenStreetMap* [online], citováno
Dostupné z: <https://www.openstreetmap.org/>, OpenStreetMap, 2021
- [40] JetBrains s.r.o.: *Ktor* [online],
Dostupné z: <https://ktor.io/>, JetBrains s.r.o., 2021
- [41] Sean Leary: *JSON in Java* [online],
Dostupné z: <https://github.com/stleary/JSON-java>, Sean Leary, 2021
- [42] Sergei Rybalkin: *kohttp – Kotlin DSL http client* [online],
Dostupné z: <https://github.com/rybalkinsd/kohttp>, Sergei Rybalkin, 2021
- [43] Nora Nure: *Car Top View* [online, cit. 2021-04-29],
Dostupné z: https://www.clipartkey.com/view/moxobx_car-top-view-png/, Nora Nure
- [44] Filip Peterek: *car-client* [online],
Dostupné z: <https://github.com/fpeterek/car-client>, Filip Peterek, 2021
- [45] Filip Peterek: *car-webapp* [online],
Dostupné z: <https://github.com/fpeterek/car-webapp>, Filip Peterek, 2021

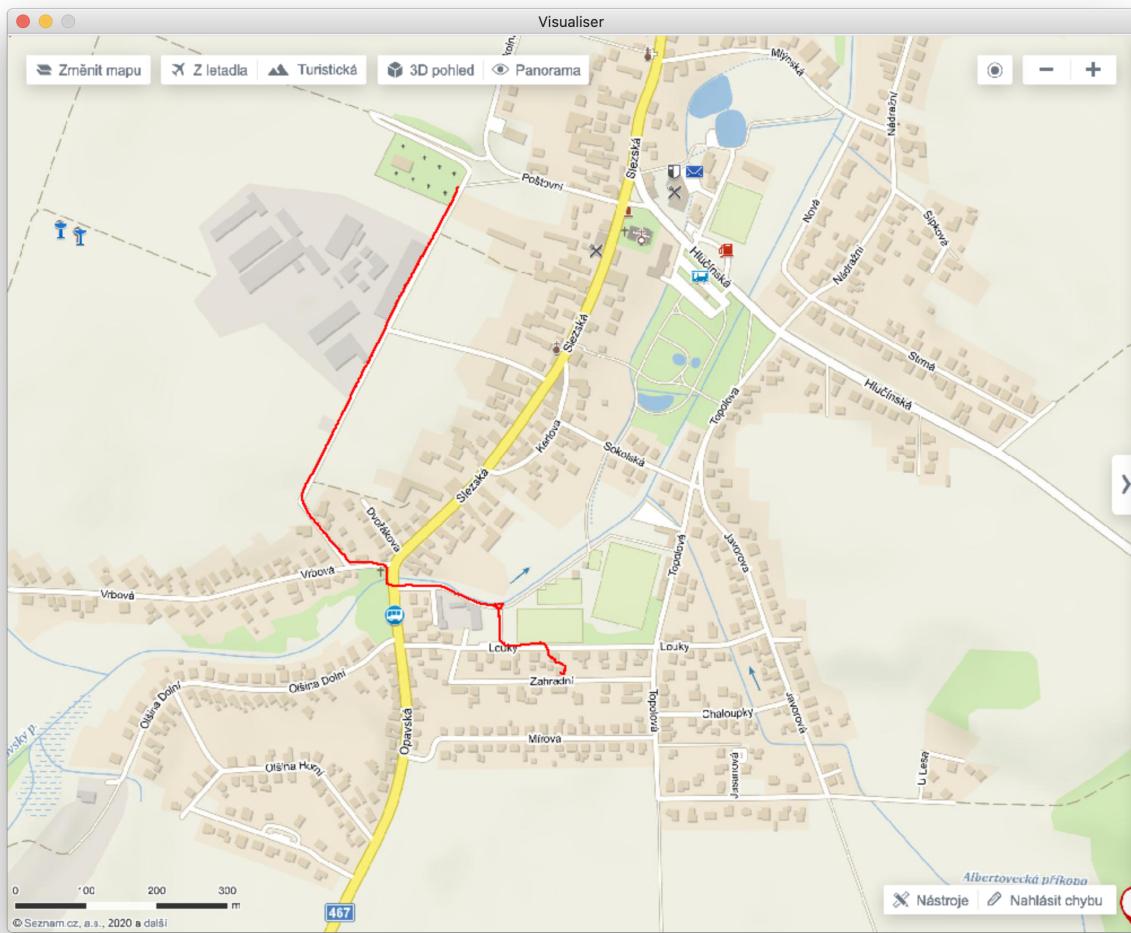
- [46] Filip Peterek: *car-map-downloader* [online],
Dostupné z: <https://github.com/fpeterek/car-map-downloader>, Filip Peterek, 2021
- [47] Filip Peterek: *car-can* [online],
Dostupné z: <https://github.com/fpeterek/car-can>, Filip Peterek, 2021
- [48] Filip Peterek: *car-simulator* [online],
Dostupné z: <https://github.com/fpeterek/car-simulator>, Filip Peterek, 2021
- [49] Filip Peterek: *GeoLogger* [online],
Dostupné z: <https://github.com/fpeterek/GeoLogger>, Filip Peterek, 2021

Příloha A

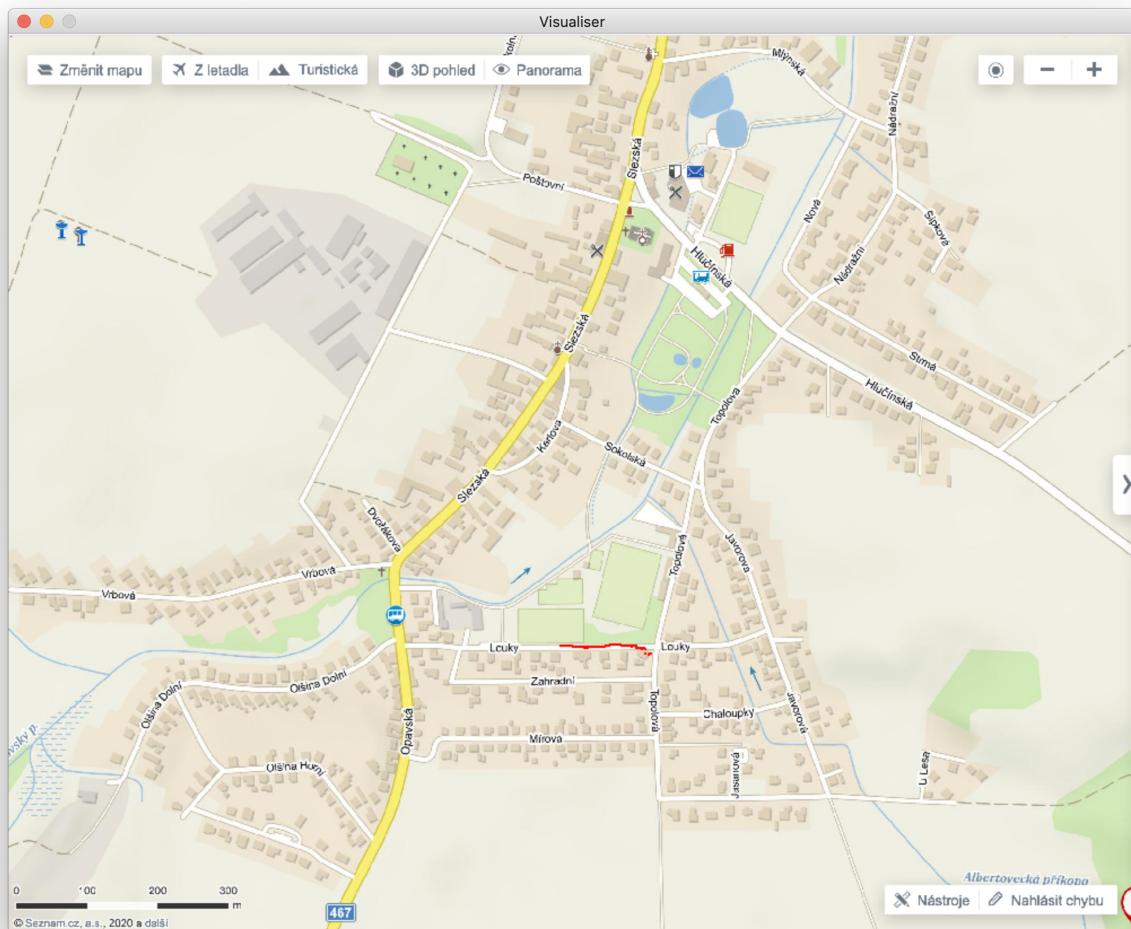
Výsledky měření pomocí GPS



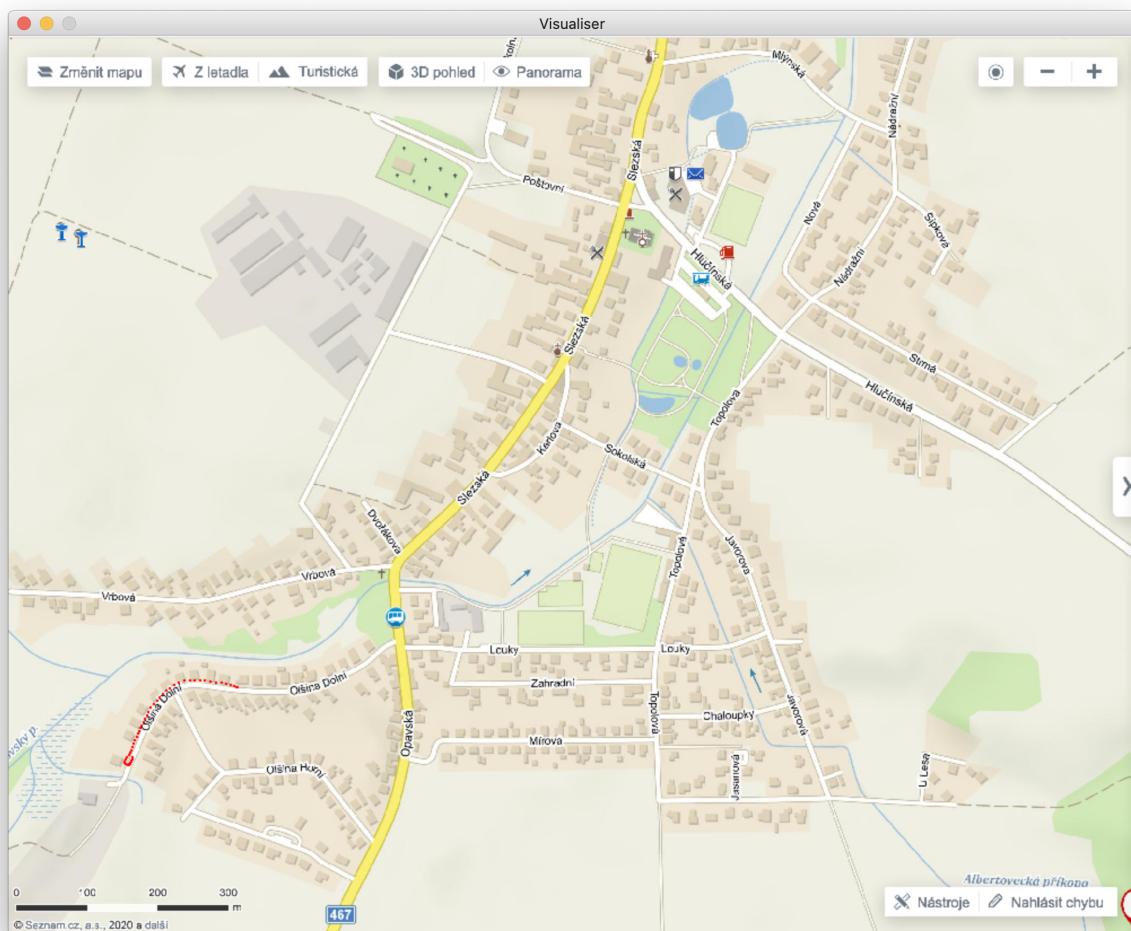
Obrázek A.1: Krátká chůze zaznamenaná pomocí GPS (Zdroj grafických podkladů: Seznam.cz, a.s.)



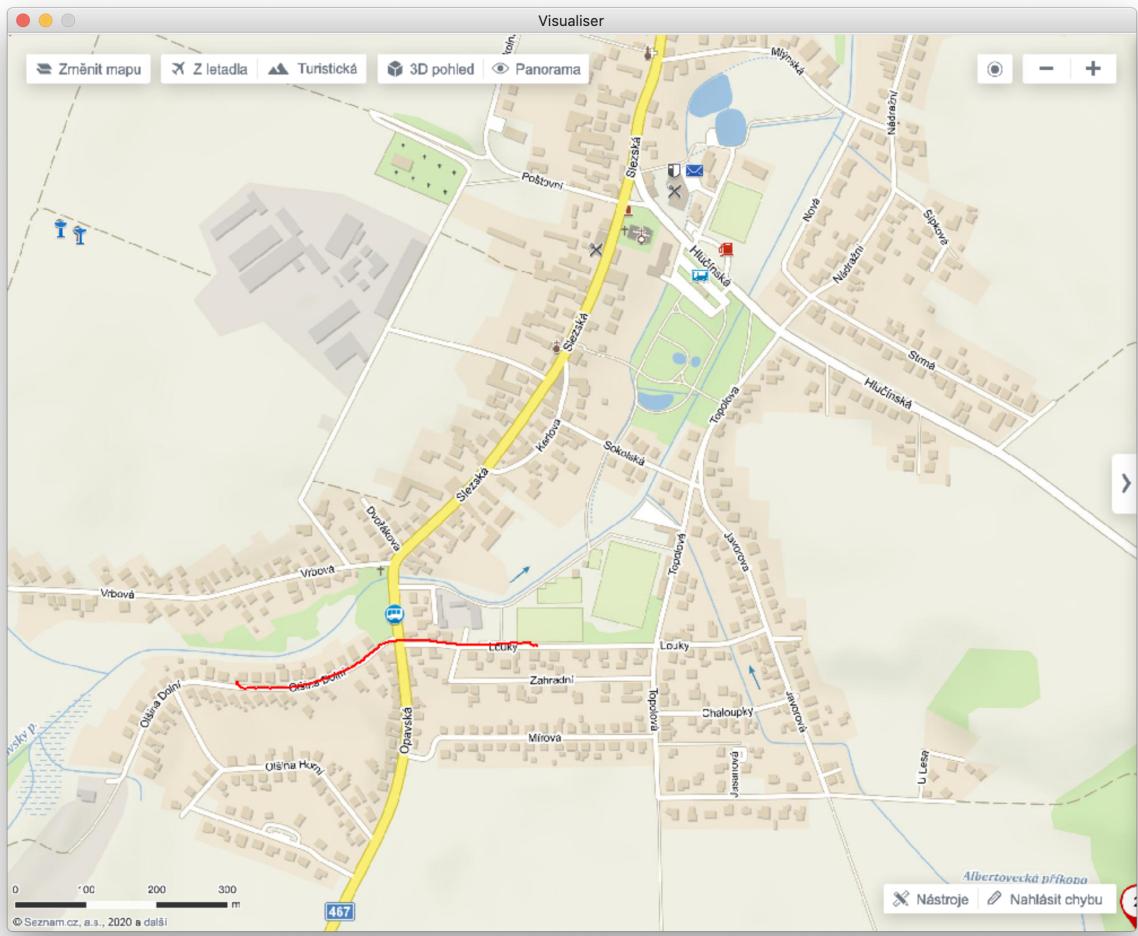
Obrázek A.2: Delší chůze, vždy po chodníku nebo silnici, zaznamenaná pomocí GPS (Zdroj grafických podkladů: Seznam.cz, a.s.)



Obrázek A.3: Rovná chůze středem cesty zaznamenaná pomocí GPS (Zdroj grafických podkladů: Seznam.cz, a.s.)



Obrázek A.4: Krátká jízda osobním automobilem zaznamenaná pomocí GPS (Zdroj grafických podkladů: Seznam.cz, a.s.)



Obrázek A.5: Krátká chůze zaznamenaná pomocí GPS (Zdroj grafických podkladů: Seznam.cz, a.s.)