VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

NÁVRH SIMULAČNÍHO PROSTŘEDÍ PRO TESTOVÁNÍ PROVOZU AUTONOMNÍCH VOZIDEL

DESIGN OF A SIMULATION ENVIRONMENT FOR TESTING THE OPERATION OF AUTONOMOUS VEHICLES

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Jan Šůstek

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Stanislav Věchet, Ph.D.

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Bc. Jan Šůstek**
Studijní program:	Mechatronika
Studijní obor:	bez specializace
Vedoucí práce:	**doc. Ing. Stanislav Věchet, Ph.D.**
Akademický rok:	2022/23

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Návrh simulačního prostředí pro testování provozu autonomních vozidel

Stručná charakteristika problematiky úkolu:

Modelování provozu autonomních vozidel v prostředí měst patří k aktuální problematice v oblasti mobilní robotiky. Cílem práce je navrhnout simulační prostředí a potřebné SW nástroje umožnující simulovat provoz autonomních vozidel v reálně existujících městech. Motivací je pak možnost porovnání ruzných strategií řízení takových vozidel a vhodnost pro konkrétní městskou infrastrukturu.

Cíle diplomové práce:

1) Navrhněte architekturu simulačního prostředí s využitím některého z dostupných SW nástrojů.
2) Navrhněte sadu nástrojů vhodných pro přípravu simulačního prostředí.
3) Vytvořte testovací polygon se základem v reálném prostředí.
4) Ověrte funkcionalitu navrženého řešení.

Seznam doporučené literatury:

SOLEM, J. E., Programming Computer Vision with Python: Tools And Algorithms For Analyzing Images, 1st edition, 2012.

LIGHT, R. A., Mosquitto: server and client implementation of the MQTT protocol, The Journal of Open Source Software, vol. 2, no. 13, May 2017, DOI: 10.21105/joss.00265.

BALÁTĚ, J.: Technické prostředky automatického řízení. Praha, SNTL 1986.

ROS.org. ROS.org | Powering the world's robots. [online]. 2.11.2016 [cit. 2016-11-02]. Dostupné z: http://www.ros.org/.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2022/23

V Brně, dne

L. S.

prof. Ing. Jindřich Petruška, CSc. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Tato diplomová práce se zabývá návrhem simulačního prostředí pro testování autonomních vozidel. V teoretické části byla provedena rešerše dostupných simulátorů autonomního řízení. Dále byly představeny nástroje běžně používané v autonomních vozidlech, jako jsou senzory nebo softwarové moduly. V praktické části byl z dostupných řešení vybrán simulátor CARLA. Nejprve je vysvětlena instalace samotného simulátoru CARLA. Poté byla pomocí programu Roadrunner vytvořena simulační mapa, která napodobuje konkrétní ulici v Brně. Následně je ukázána práce se simulátorem CARLA. Na závěr je zhodnocena práce se simulátorem CARLA a ukázány konkrétní výstupy simulace.

KLÍČOVÁ SLOVA

CARLA simulator, autonomní vozidla, simulace, Roadrunner, simulační prostředí, senzory

ABSTRACT

This thesis deals with the design of simulation environment for testing autonomous vehicles. In the theoretical part, the search of available autonomous driving simulators was carried out. Furthermore, the tools commonly used in autonomous vehicles such as sensors or software modules were presented. In the practical part, the CARLA simulator was selected from the available solutions. Firstly, the installation of CARLA simulator is explained. Then, the simulation map was created by the Roadrunner to simulate a specific street in Brno. Afterwards, the work with the CARLA simulator is shown. Finally, the work with the CARLA simulator is evaluated and the concrete outputs of simulation are shown.

KEYWORDS

CARLA simulator, autonomous vehicles, simulation, Roadrunner, simulation environment, sensors

BIBLIOGRAFICKÁ CITACE

ŠŮSTEK, Jan. *Návrh simulačního prostředí pro testování provozu autonomních vozidel* [online]. Brno, 2023 [cit. 2023-05-18]. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/145866. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Stanislav Věchet.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, zpracoval jsem ji samostatně pod vedením doc. Ing. Stanislava Věcheta, Ph.D. a s použitím informačních zdrojů uvedených v seznamu.

V Brně dne 20. května 2023 ……..……..

Jan Šůstek

PODĚKOVÁNÍ

Chtěl bych poděkovat panu doc. Ing. Stanislavu Věchetovi, Ph.D. za odborné vedení práce a cenné rady, které mi pomohly tuto práci zkompletovat.

OBSAH

ÚVOD

Autonomní vozidla mají potenciál způsobit revoluci v dopravě, snížit počet dopravních nehod, snížit emise uhlíku a vytvořit nové možnosti mobility a dopravy. Tato vozidla se mohou pohybovat po silnicích bez zásahu člověka a spoléhat se na pokročilé senzory, algoritmy strojového učení a sofistikované řídicí systémy. Vývoj a testování autonomních automobilů je však složitý a náročný úkol, který vyžaduje vysokou úroveň bezpečnosti a spolehlivosti.

Jedním z klíčových nástrojů pro vývoj a testování autonomních automobilů je simulace. Simulátory hrají při vývoji a zavádění autonomních vozidel zásadní roli z několika důvodů. Zaprvé poskytují bezpečné a kontrolované prostředí pro testování a zdokonalování algoritmů autonomního řízení. Simulace různých scénářů jízdy umožňuje vývojářům posoudit kvalitu jejich systémů bez rizik spojených s testováním v reálném světě. Mohou vylepšovat algoritmy a ověřovat chování autonomních vozidel v různých situacích.

Simulátory také umožňují vyhodnocovat autonomní systémy v široké škále náročných a složitých scénářů, které jsou nebezpečné vytvářet v reálném světě. Simulátory mohou například replikovat nepříznivé povětrnostní podmínky, hustý provoz nebo vzácné okrajové případy, které jsou klíčové pro testování robustnosti a spolehlivosti autonomních systémů. Vystavením autonomních vozů těmto simulovaným scénářům mohou vývojáři zlepšit jejich schopnosti.

Další významnou výhodou simulátorů je jejich schopnost shromažďovat velké množství označených trénovacích dat pro algoritmy strojového učení. Trénink autonomních systémů s daty z reálného světa může být nákladný, časově náročný, a dokonce i nebezpečný. Simulátory poskytují nákladově efektivní a škálovatelné řešení, které lze doplnit o data z reálného světa a zlepšit výkonnost modelů strojového učení. Simulovaná data také umožňují vývojářům vytvářet specifické scénáře a systematicky vyhodnocovat chování a výkonnost jejich algoritmů.

1 REŠERŠE

V rešerní části této práce jsou představena dostupná simulační prostředí, následně zhodnocena a vybráno z nich jedno, které bude dále použito. V další kapitole jsou prozkoumány nástroje, které autonomní auta používají.

1.1 DOSTUPNÉ SIMULAČNÍ PROSTŘEDÍ

Testování autonomních vozidel v normálním provozu je nákladné, nebezpečné a často právně nedovolené. Z tohoto důvodu vznikají simulátory. Nejdůležitějším parametrem simulátorů je to, jak dobře dokáží reprezentovat skutečné prostředí. V této kapitole představíme některé z dostupných simulátorů.

1.1.1 MATLAB / SIMULINK

Matlab nabízí Automated Driving Toolbox™. Tento toolbox umožňuje simulovat a testovat autonomní řízení a systémy ADAS (Advanced driver-assistance system). Do tohoto nástroje se také dají nahrát data z HERE HD Live Map nebo z OpenDRIVE. Je uzpůsoben také pro testování HIL (Hardware In the Loop), simulaci senzorů, plánování cesty a řídicí logiky, díky čemuž může generovat a simulovat různé jízdní scénáře. Můžete simulovat výstupy kamer, radarů a lidarů ve fotorealistickém 3D prostředí a detekci objektů a hranic jízdních pruhů. Matlab také nabízí integraci s jinými simulátory, např. CarSim, PreScan nebo Gazebo. [1]

Obrázek 1 - Automated Driving Toolbox™

1.1.2 CARLA

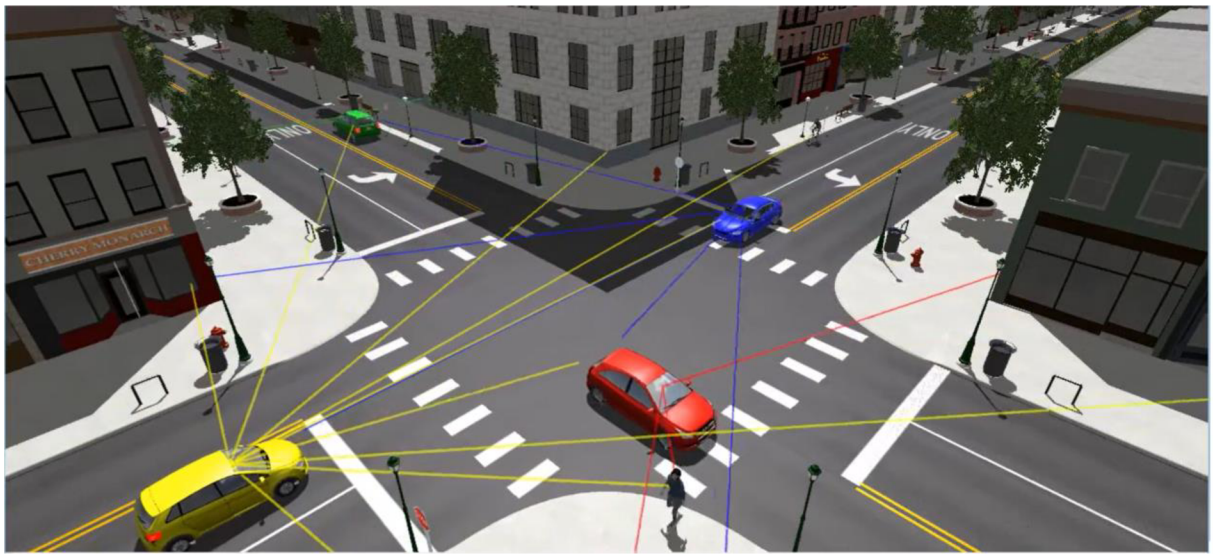
CARLA je open-source projekt, který je vytvořen pro výzkum oblasti autonomního řízení. Je založen na Unreal Engine. CARLA má vytvořené Python API, které slouží k ovládání simulace. Architektura tohoto simulačního prostředí je typu klient-server. Úlohy související se simulací,

jako aktualizace stavu světa, aktérů, senzorů, výpočet fyzikálních dějů, běží na serveru. Klient řídí logiku chodců, aut a nastavuje podmínky. CARLA nabízí kvalitní simulaci reálné fyziky (tření kol nebo odpružení). CARLA podporuje propojení s různými platformami, jako jsou ROS, Autoware, Apollo nebo AirSim. [2]

Obrázek 2 - CARLA simulator [3]

1.1.3 CarSim

Nejnovější verze CarSim podporuje pohybující se objekty a senzory, což je důležité pro simulace systémů ADAS a autonomních vozidel (AV). Je zde k dispozici až 200 objektů s nezávislými polohami a pohyby. Tyto objekty zahrnují statické objekty, jako jsou stromy a budovy, a dynamické objekty, jako jsou chodci, vozidla, zvířata a další. Klíčovou vlastností CarSim je to, že má možnost propojení s jinými softwary, jako jsou Matlab a LabVIEW. [4]



Obrázek 3 - CARSIM

1.1.4 PRESCAN

Pomocí Simcenter Prescan mohou inženýři virtuálně ověřit funkce ADAS a automatizovaných vozidel. Přednost PreScanu je v replikování reálných dopravních scénářů. K vytvoření realistického prostředí pomáhá možnost importovat informace o prostředí z OpenStreetMap nebo Google Earth. Je zde také možnost vytvářet vlastní sady snímačů, ovládání a řízení. PreScan také podporuje simulaci HIL. PreScan umí dobře provádět výpočty vstupů snímačů na základě fyzikálních zákonitostí. Systém PreScan navíc nabízí jedinečnou funkci nazvanou Vehicle Hardware-In-the-Loop (VeHIL). Ta umožňuje uživatelům vytvořit kombinovaný reálný a virtuální systém. [5] [6]

Obrázek 4 – PreScan [7]

1.1.5 GAZEBO

Gazebo je open source systém. Umožňuje simulaci vnitřních i venkovních prostorů. Je zde rozvinutá možnost modifikace fyzikálních vlastností objektů, jako hmotnosti, tření, apod. Pro vizualizaci využívá Object-Oriented Graphics Rendering Engine (OGRE). Další knihovna umožňuje komunikaci mezi jednotlivými prvky Gazebo systému. Každá scéna v Gazebo se skládá ze světa a modelu. Svět reprezentuje simulační prostředí a model je jakýkoliv 3D objekt v tomto prostředí. Tyto modely si může uživatel sám vytvořit. Gazebo je často využívaný ve spojení s ROS prostředím. [8]

Obrázek 5 – Gazebo [9]

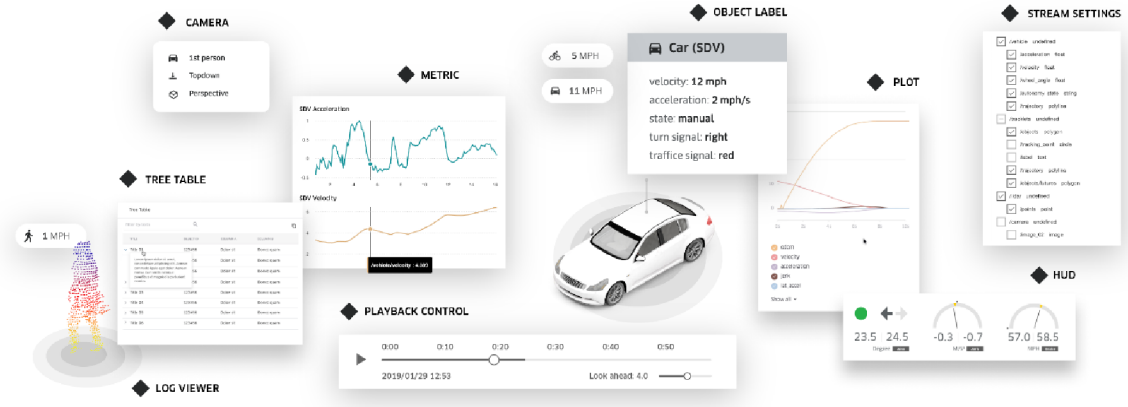
1.1.6 LGSVL

Simulní prostředí od LG představuje hotové open-source řešení pro testování algoritmů pro autononí vozidla. Využívá herní engine Unity. Simulátor nabízí Python API, díky kterému se dají ovládat prvky v simulaci. LGSVL nabízí kamery, LiDAR, IMU, GPS a radar a také možnost volit stav počasí, denní dobu, ostatní účastníky provozu. LGSVL také podporuje propojení s ROS, Autoware nebo Apollo. [10]

Obrázek 6 – LGSVL [11]

1.1.7 UBER AVS: AUTONOMOUS VISUALIZATION SYSTEM

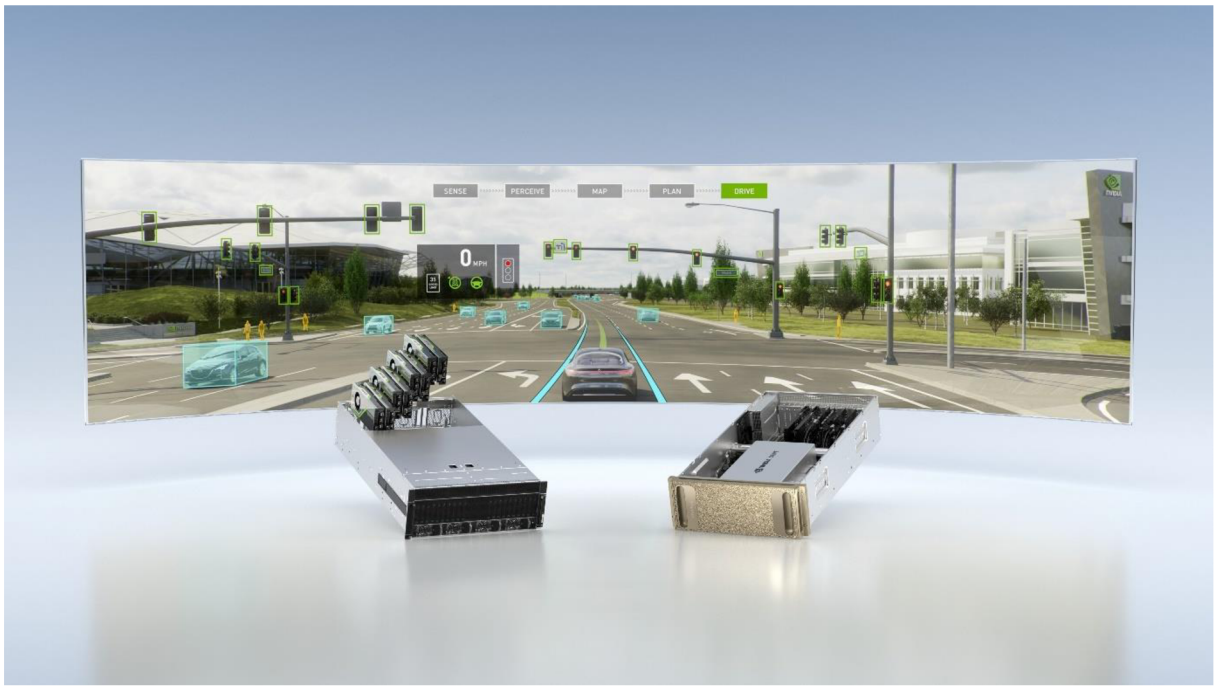
AVS je nástroj od firmy Uber ATG. AVS vytváří vizualizace dat shromážděných z různých senzorů a systému samořiditelných aut. AVS se skládá ze dvou prvků. Prvním prvkem je XVIZ, což je specifikace, která popisuje data generovaná autonomními auty. Druhým prvkem AVS je streetscape.gl, nástroj vytvářející webové rozhraní pro zobrazení dat v protokolu XVIZ. [12] [13]



Obrázek 7 - Uber AVS

1.1.8 NVIDIA DRIVE

NVIDIA Drive nabízí jak softwarové, tak hardwarové řešení pro testování autonomních vozidel. Tento systém je rozdělen do tří podčástí: NVIDIA Drive Sim, NVIDIA OVX Server a DRIVE Constellation Vehicle. Drive Sim je 3D simulační pro vytvoření virtuálního světa. NVIDIA OVX Server je server pro získávání dat ze simulace a posílání je do hardwaru. DRIVE Constellation Vehicle je vozidlo, které je vybaveno NVIDIA hardwarem a senzory a zpracovává data ze senzorů ze simulace. [14]



Obrázek 8 - NVIDIA DRIVE [15]

1.1.9 RFPRO

RFpro nabízí vysoce realistické simulační prostředí pro vývoj autonomních systémů. Poksytuje také řadu nástrojů, aby vývojáři mohli vytvořit vlastní hardwarové nebo softwarové komponenty. Další funkcí tohoto simulátoru je jeho spojení s programem Simulink od Mathworks. [16]

Obrázek 9 – rFpro [17]

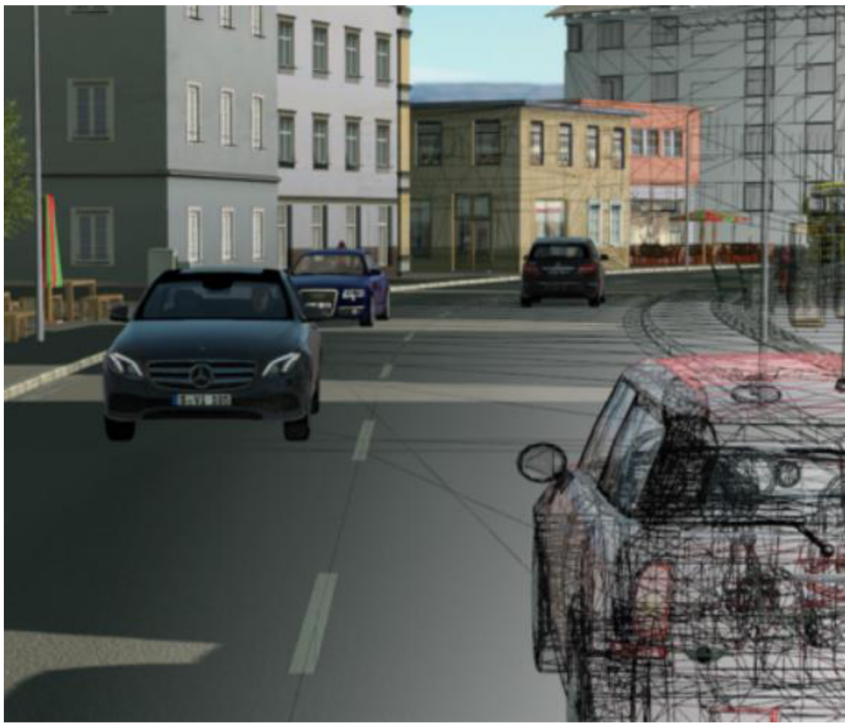
1.1.10 COGNATA SIMULATOR

Cognata simulátor je vytvořen tak, aby poskytoval virtuální prostředí, které kopíruje scénáře reálného řízení. Využívá 3D modely reálného prostředí s vysokým rozlišením, včetně silnic, budov a dalších objektů, a poskytuje tak vysoce realistické simulační prostředí. Cognata umožňuje testování ve virtuálním městě, dálnici i terénu a to za jakéhokoliv počasí. [18] [19]

Obrázek 10 - Cognata simulator

1.1.11 VIRES VIRTUAL TEST DRIVE

VTD Road Designer je program, který slouží k vytvoření virtuálního 3D světa. Je zde široká knihovna 3D objektů všech typů. VTD umožňuje simulaci komplexních dopravních situací. VTD může být dále využito v aplikacích MIL (Model in the Loop), SIL (Software in the Loop), HIL (Hardware in the Loop), DIL (Driver in the Loop) a VIL (Vehicle in the Loop). [20]



Obrázek 11 - VIRES VIRTUAL TEST DRIVE

1.1.12 IPG CarMaker

CarMaker je simulační prostředí, které bylo vytvořeno pro vývoj a testování osobních vozidel ve všech fázích vývoje (MIL, SIL, HIL, VIL). Vizualizační nástroj MovieNX zaručuje fotorealistickou kvalitu virtuálního 3D prostředí. Všechny komponenty a senzory můžou být nahrazeny vlastními modely. CarMaker taky nabízí řadu možností pro vyhodnocování získaných dat. [21]

Obrázek 12 - IPG CARMAKER

1.1.13 UDACITY SIMULATOR

Udacity simulator byl vytvořen pro potřeby kurzu Self-Driving Car Engineer. Tento simulátor běží na platformě Unity. Jsou zde dva módy: tréninkový, kde může uživatel ovládat vozidlo a nahrát si data pro svůj model, a automatický mód, který je určen pro testování vytvořených algoritmů. [22]

Obrázek 13 - Udacity simulator

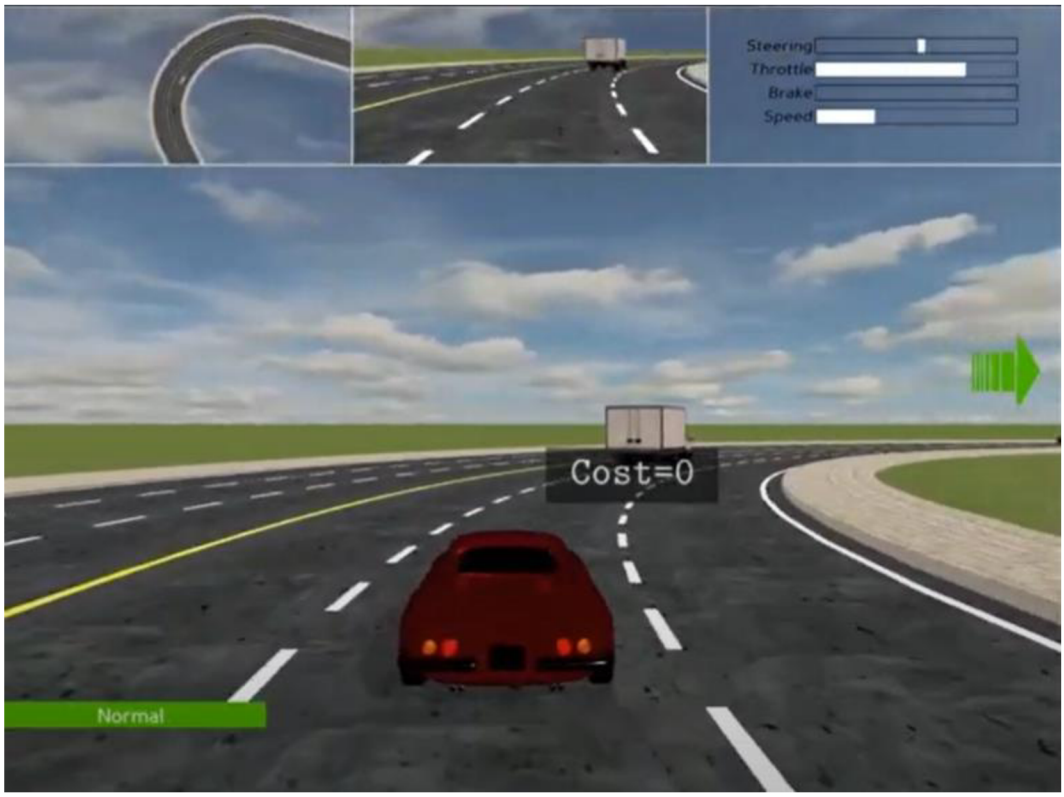
1.1.14 AIRSIM

AirSim je simulátor pro testování aut nebo i jiných zařízení, např. dronů. Je vytvořen jako Unreal Engine plugin. Simulátor taky podporuje vývoje Software-in-the-Loop nebo Hardware-in-the-Loop. Pro kontrolu simulace je vytvořeno API, které podporuje různé programovací jazyky, Python, C++ nebo Java. [23]

Obrázek 14 - AirSim

1.1.15 METADRIVE

MetaDrive simulátor nabízí různé mapy a nastavení provozu. Je jednoduchý na instalaci a provoz. Podporuje generování scén pro testování zpětnovazebného učení. [24]



Obrázek 15 – MetaDrive [25]

1.1.16 GYM-DUCKIETOWN

Gym-Duckietown je jedoduchý simulátor v prostředí Duckietown. Zde jako vozidlo figuruje tzv. Duckiebot. Jsou zde nadefinované mapy, které se dají přizpůsobovat. Ačkoliv je oproti jiným simulátorům graficky jednodušší, tak dokáže posloužit pro testování různých algoritmů. [26]

Obrázek 16 - Gym-Duckietown

1.1.17 SROVNÁNÍ

Matlab je vhodný pro výpočty, ale jeho hlavní nevýhodou je omezená možnost vytvoření reálného prostředí. PreScan dokáže důvěryhodně simulovat reálné prostředí a dokáže spolupracovat s programem Matlab. Gazebo je velmi flexibilní. Nevýhodou u Gazebo je zdlouhavý ruční proces k vytvoření světa, protože uživatel musí sám vytvořit modely a definovat fyzikální vlastnosti. IPG CarMaker nabízí vytváření velmi přesných virtálních prototypů vozidla. Zároveň lze všechny komponenty nahradit konkrétními modely od zákazníka. CARLA i LGSV vyžadují vyšší nároky na výkon pro vytvoření kvalitních simulací. U obou simulátorů je vytvořeno API k ovládání simulací. CARLA je založená na Unreal Engine, LGSVL na Unity. Výhodou Carly oproti LGSVL je, že všechny moduly má importované v sobě a nemusí se připojovat na externí moduly.

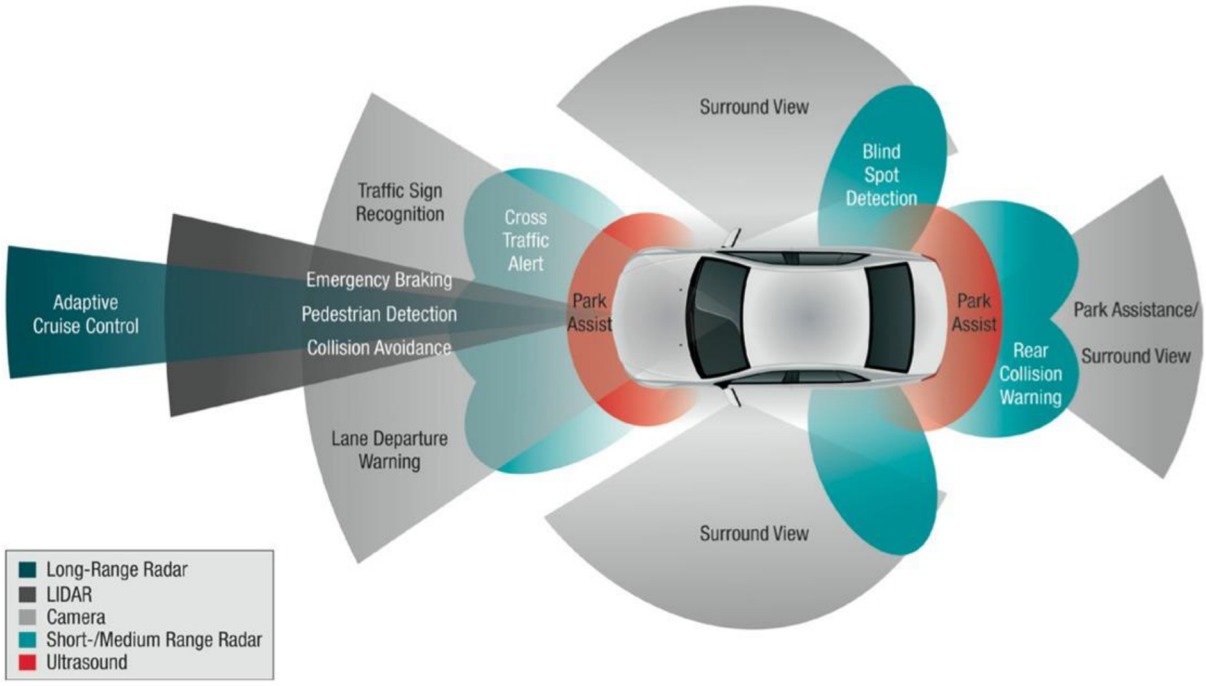
Ze zmíněných simulátorů se pro komplexní testování algoritmů nejvíce hodí prostředí CARLA, LGSVL nebo CarMaker.

1.2 NÁSTROJE BĚŽNÉ U AUTONOMNÍCH AUT

Autonomní vozidla potřebují ke své funkci sbírat data z okolí a tyto data vyhodnocovat. Aby se mohlo vozidlo chovat co nejvíce jako člověk, vědět kde se nachází a jak se bezpečně pohybovat, tak využívá senzory a počítačové systémy.

1.2.1 SENZORY

V současné době neexistuje jedna součástka, která by dokázala dodat všechny informace pro autonomní vozidlo. Proto se využívá kombinace senzorů, které dohromady dávají data pro řízení auta.



Obrázek 17 - Senzorika u autonomního auta [29]

LIDAR

Lidar se používá pro měření vzdálenosti. Měří, za jakou dobu se vyslaný laserový paprsek vrátí od snímaného objektu zpět. Z měření se získá mračno bodů, který se dá poté převést do 3D mapy okolí vozidla. Dosah lidaru dosahuje až 250 metrů. Tato technologie je velice precizní a přesná, problémy mohou nastat v husté mlze nebo při sněžení. Nevýhodou Lidaru je jeho vysoká cena. [27] [28]

RADAR

Radar využívá rádiových vln k měření vzdálenosti a rychlosti objektů. Tyto senzory detekují objekty do 100 metrů. Výhodou radaru je jeho nenáchylnost k povětrnostním podmínkám. Hlavní nevýhodou je úzké zorné pole, proto se musí využít více těchto senzorů k pokrytí celého rozsahu. Tyto senzory jsou používány i v současných autech kvůli tempomatům nebo detekci kolizí. [27] [28]

KAMERA

Kamera vytváří digitální obraz snímané oblasti. Hlavní výhodou kamery je, že může detekovat barvy a textury. Díky tomu je možné identifikovat dopravní značky nebo jízdní pruhy. Nevýhodu je jejich ovlivnění intenzitou světla a špatnými povětrnostními podmínkami. [27] [28]

ULTRAZVUKOVÝ SNÍMAČ

Tyto snímače využívají ultrazvukové vlny. Tento senzor také funguje na principu měření času odezvy a vypočítá z něj vzdálenost. Dokáže velmi dobře pracovat na krátkých vzdálenostech. Funguje dobře v husté mlze i v deští , ale může ho negativně ovlivnit rušivé zvukové vlny nebo velké změny teploty. [27] [28]

GPS

GPS je globální družicový polohový systém a umožňuje získat přesnou polohu automobilu. Nevýhodou použití technologie GPS pro autonomní vozidla je, že existuje mnoho faktorů, které mohou zhoršit kvalitu GPS. Mnoho vozidel využívá GPS pro navigaci, ale pro plně autonomní auto nemá dostatečnou přesnost a musí se využívat v kombinaci s jinými senzory. [27] [28]

IMU

Inerciální měřící jednotka je složena z akcelerometrů, gyroskopů a magnetometrů. Dokáže odvodit lineární zrychlení a uhlovou rychlost vozidla v prostoru. Funguje za všech podmínek. Nevýhodou je, že signály naintegrovávají chybu v důsledku matematické integrace zrychlení. Často se IMU používá v kombinaci s GPS pro určení přesné pozice. [28]

ENKODÉR

Enkodéry převádějí lineární nebo úhlovou polohu na analogový nebo digitální signál. Enkodéry mohou být inkrementální nebo absolutní. Používájí se k informaci o ujeté vzdálenosti. [28]

1.2.2 SOFTWARE

V této podkapitole budou představeny moduly, které se u autonomních vozidel používají.

VNÍMÁNÍ OKOLÍ

Vnímání okolí (angl. perception) pracuje s nezpracovanými daty ze senzorů, tak aby zjistil informace o okolí vozidla. Tento algoritmus obsahuje práci s objekty, jako jsou chodci, vozidla nebo jízdní pruhy, a poté rekonstruuje okolí automobilu. Tato technologie je založená na počítačovém vidění a strojovém učení. Fúze informací z více senzorů může vytvořit detailní model světa. [30]

LOKALIZACE A MAPOVÁNÍ

Tento modul pracuje s výstupem z vnímání okolí a daty ze senzorů a díky těmto informacím aktualizuje mapu a odhaduje polohu vozidla. Tomuto systému se říká také SLAM (Simultánní lokalizace a mapování). Systémy SLAM založené na filtraci využívají inkrementální integrace dat ze senzorů, a díky tomu aktualizují polohu a mapu okolí. Využívá se zde Kalmanův filtr nebo částicový filtr. Optimalizační systémy SLAM nejdříve zjistí rozdíly mezi vytvořenou

mapou a novými daty, z těchto zjištění poté aktualizují mapu.Využívají se zde optimalizační techniky jako minimalizace chybové funkce nebo Gauss-Newtonova metoda. [30]

PREDIKCE

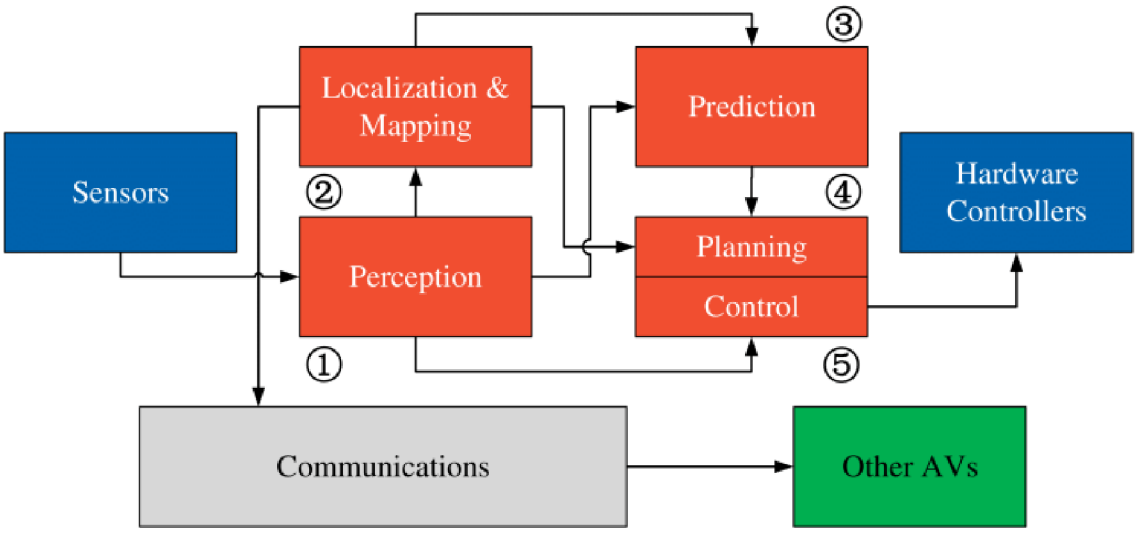
Predikce analyzuje pohyb ostatních účastníků provozu, a tak umožňuje předpovídat další pohyb vozidla. Buď se využívá predikce, která je založena na datech, nebo na modelu. [30]

PLÁNOVÁNÍ

Plánování vychází z informací z předchozích modulů. Naplánovaná trasa je seznam bodů, po kterých má auto jet, aby dorazilo do cíle. K plánování trasy se používají algoritmy Dijkstra, A*, apod. Naplánování trasy je složitá věc, protože do úvahy se musí brát i pravidla silničního provozu. [30]

ŘÍZENÍ

Podle trajektorie, která vychází z předchozích modulů, posílá příkazy pro přidání plynu, brždění nebo zatáčení. Parametry regulátoru se odhadují pomocí minimalizace chybové funkce mezi ideálním a pozorovaným stavem. Nejčastěji využívané přístupy jsou PID nebo LQR nebo kombinace obou. [30]



Obrázek 18 - Software u autonomních vozidel [30]

2 POSTUP A VÝSLEDKY ŘEŠENÍ

2.1 VÝBĚR SIMULÁTORU

V rešeršní části byly porovnány různé simulační nástroje. Z těchto nástrojů bylo nejdříve potřeba vybrat jeden, který nejvíce vyhovuje našim požadavkům. Hlavní požadavky na simulační prostředí byly:

- Open-source nástroj
- Možnost vytvářet vlastní mapy
- Schopnost ovládání simulace
- Realistické simulační prostředí
- Dobrá dokumentace
- Aktivní vývoj

Na základě těchto požadavků byl vybrán simulátor CARLA. V Carle je dostatek předvytvořených map a zároveň je zde možnost si vytvořit vlastní mapu. CARLA nabízí vytvořené senzory pro snímání prostředí. Zároveň je zde vytvořeno Python API pro kontrolu simulace a pro vytváření a testování algoritmů pro autonomní řízení.Výhodou Carly může být také spojení s jinými nástroji jako je ROS nebo Autoware.

V této části bude ukázána práce s Carlou od instalace, přes vytvoření vlastní mapy, až po samotné fungování v simulátoru.

2.2 INSTALACE CARLY

2.2.1 ZPŮSOBY INSTALACE

CARLA může být nainstalovaná dvěma způsoby:

- Instalace balíčku (Package instalation) – jednodušší a rychlejší varianta. Obsahuje CARLA server a knihovnu pro klienta. Tahle varianta je vhodná pro seznámení s Carlou, ale pokud uživatel chce využít všechny možnosti rozšíření a vývoje, tak musí využít druhou možnost.
- Instalace ze zdroje (Build from source) – pokročilejší verze. Tato verze slouží pro vývoj a je zde možno modifikovat více prvků. K této verzi je potřeba stáhnout i Unreal Engine, který se využívá k instalaci a jeho editor k úpravě prostředí. Zároveň je zde Python API, které nám umožňuje ovládat simulaci. Tato verze je náročnější pro výkon počítače.

K využití plného potenciálu CARLA simulátoru byla vybrána verze ze zdroje. O této verzi bude tedy psáno i v dalších kapitolách.

2.2.2 SYSTÉMOVÉ POŽADAVKY

Pro tuto práci byla vybrána verze ze zdroje a operační systém Windows.

- Operační systém: Windows 64 bitový
- Grafický procesor: Je doporučeno, aby měl velikost operační paměti alespoň 8GB pro co nejrealističtější simulace. CARLA poběží i s menší operační pamětí, ale nevyužije se celý její potenciál.
- Místo na disku: 165 GB

- Alespoň dva TCP porty

2.2.3 POSTUP INSTALACE

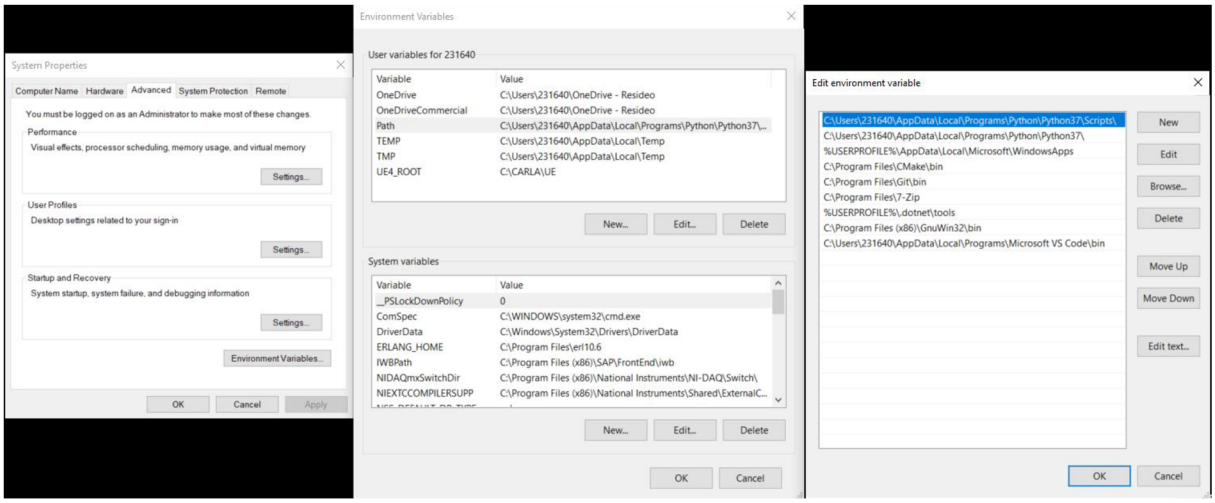
V následujících podkapitolách bude popsán postup instalace. Nejdříve budou nainstalovány programy, které jsou nutné pro fungování CARLA simulátoru a následně proběhne instalace samotného CARLA simulátoru.

INSTALACE NUTNÝCH PROGRAMŮ

Nejdříve je potřeba nainstalovat dodatečné programy, bez kterých nám simulátor CARLA nebude fungovat. Jsou to tyto programy:

- Cmake [32]
- Git [33]
- Make 3.81 [34]
- 7Zip [35]
- Python3 x64 [36] – Na základě poznatků z práce s CARLA simulátorem byla vybrána verze 3.7.9, kterou simulátor podporuje.

Poté, co jste všechny tyto programy nainstalovali, je potřeba přidat umístění souboru do systémové proměnné "PATH (Cesta)". Systémové proměnné najdeme v okně "Proměnné prostředí (Environment Variables)", které můžeme vyhledat např. pomocí položky "Hledat" v liště "Start".



Obrázek 19 - Přidání souboru do systémové proměnné

Abychom mohli v instalaci pokračovat, potřebujeme verzi pip3 (oficiální správce balíčků) 20.3 nebo novější. Toho dosáhneme pomocí následujících příkazů v příkazovém řádku:

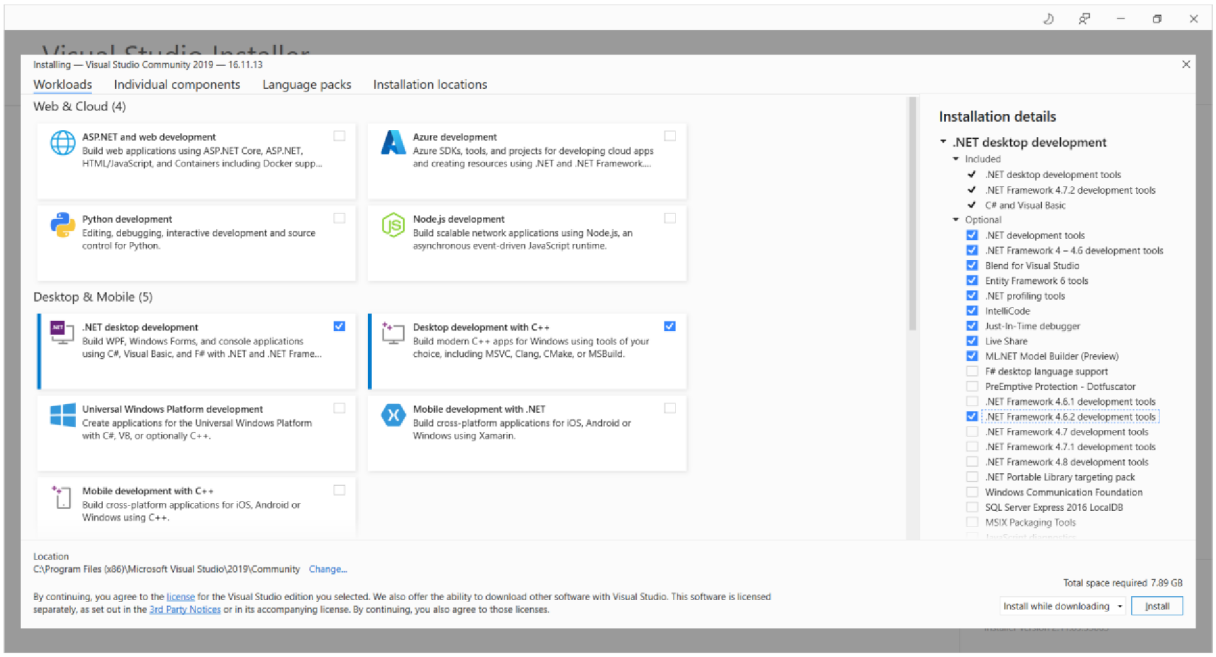
```
pip3 install --upgrade pip

pip3 install --user setuptools

pip3 install --user wheel
```

INSTALACE VISUAL STUDIO

Dalším krokem je instalace Visual Studia 2019 verze Community. Visual Studio se může stáhnout z [37]. Na dalším obrázku je ukázáno, co zakliknout u instalace. K nainstalovaným věcem se musí ještě dodatečně doinstalovat Windows 8.1. SDK z [38].



Obrázek 20 - Instalace Visual Studio

INSTALACE UNREAL ENGINE

Před instalací je nejdříve potřeba propojit účet Github s účtem na UnrealEngine.com. To se provede podle následujících kroků:

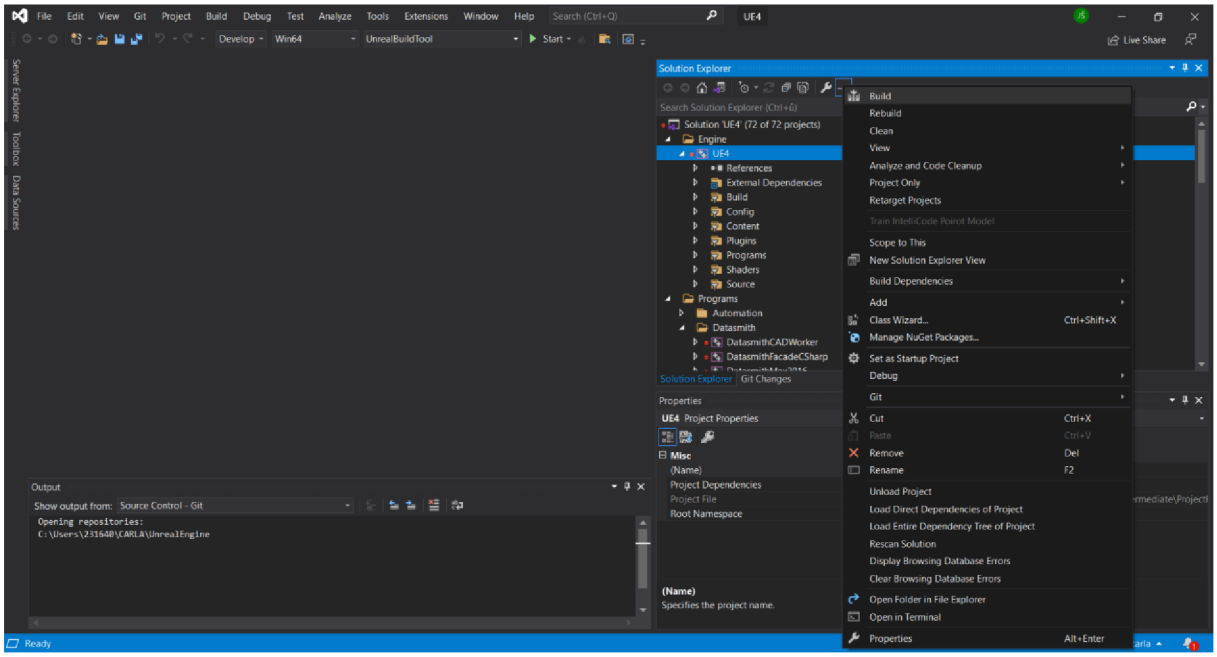
- Založení účtu Github na [39]
- Založení účtu Epic Games [40]
- Po přihlášení na UnrealEngine.com účtem Epic Games přejít na záložku Personal, v ní do záložky Connections, kliknout na tlačítko Apps a zde pod ikonou Github připojit k účtu z Github
- Pro dokončení propojení odkliknout potrvzovací tlačítko v příchozím e-mailu.

Nyní přejdeme k samotné instalaci Unreal Engine:

- V příkazovém řádku pomocí příkazu cd přejdeme do složky, kde chceme Unreal Engine nainstalovat. Volíme umístění adresáře, co nejblíže disku C. Pokud bude cesta k adresáři příliš dlouhá, může to způsobovat problémy v dalších krocích.
- Naklonujeme Unreal Engine pomocí příkazu:

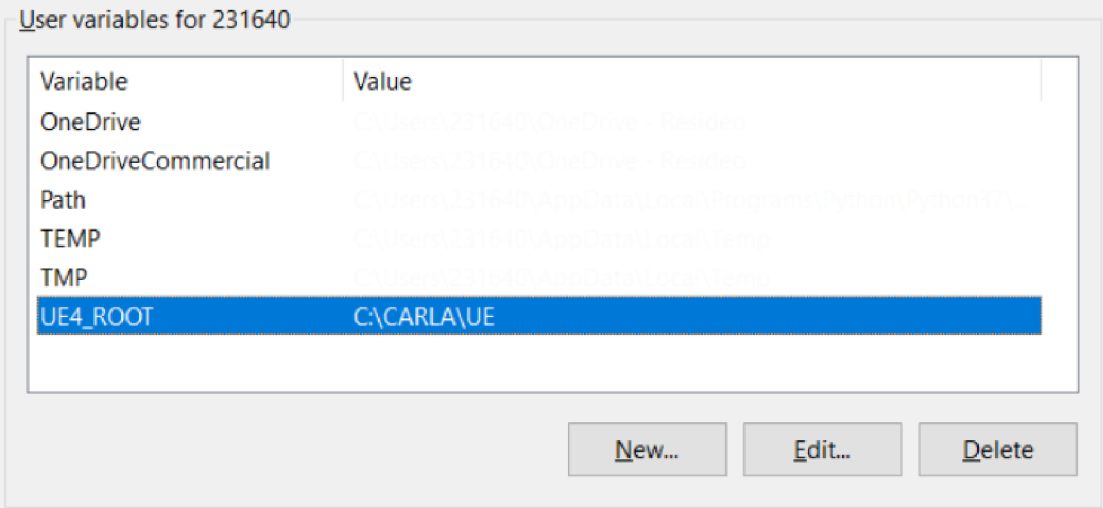
```
git clone --depth 1 -b CARLA
https://github.com/CARLAUnreal/UnrealEngine.git
```

- Po dokončení předchozího kroku spustíme v instalační složce postupně konfigurační soubory Setup.bat a GenerateProjectFiles.bat
- Po dokončení konfigurace otevřeme soubour UE4.sln pomocí Visual Studia. Zde ve vrchní liště zvolíme "Development Editor", "Win64" a "UnrealBuildTool". Vpravo po kliknutí na záložku "Solution Explorer", klikneme pravým tlačítkem na "UE4" a zmáčkneme příkaz "Build".



Obrázek 21 - Instalace Unreal Engine

- Restartujeme počítač
- Přejdeme do "Proměnného prostředí (Environment Variables)", klikneme na "Nový (New)" a vytvoříme novou proměnnou a pojmenujeme ji "UE4_ROOT" a přidáme cestu k instalačnímu adresáři Unreal Engine.



Obrázek 22 - Přidání Unreal Engine do Proměnného prostředí

INSTALACE CARLA SIMULATORU

Nyní je nainstalovaný Unreal Engine i ostatní programy a může být nainstalována CARLA:

- V příkazovém řádku pomocí příkazu "cd" přejdeme do složky, kde chceme CARLA simulátor nainstalovat. V příkazovém řádku zadáme příkaz:

```
git clone https://github.com/CARLA-simulator/CARLA
```

- Ke stažení nejaktuálnějších prvků simulátoru zapneme program Update.bat
- V souboru CARLA\Util\InstallersWin\install_zlib.bat na řádku 51 přepíšeme verzi ZLIB na 2.1.13. (Aktuální verzi zlib dokážeme zjistit na stránce zlib.net)

Všechny následující příkazy je nutno spouštět v příkazovém řádku x64 Native Tools Command Prompt for VS 2019, který najdeme v nabídce "Start". V něm pomocí příkazu cd změníme složku na cestu k CARLA adresáři.

- Dalším krokem je zkompilování Python API. Python API nám umožňuje kontrolovat simulaci. Zkompilování Python API provedeme příkazem:

```
make PythonAPI
```

Jestli zkompilování proběhlo správně, zjistíme tak, že ve složce CARLA\PythonAPI\CARLA\dist najdeme soubory egg a whl. Pokud zde tyto soubory nejsou, tak spustíme příkaz ještě jednou.

- Následujícím příkazem zkompilujeme a spustíme server a Unreal Engine. Tento příkaz využíváme vždy když chceme zapnout simulátor. [31]

```
make launch
```

CHYBY PŘI INSTALACI

Při instalaci CARLA simulátoru se mohou vyskytnout nějaké typické chyby. Příklady těchto chyb, jejich důvody a způsob vyřešení jsou v následující tabulce.

NÁZEV CHYBY	POPIS A POSTUP VYŘEŠENÍ
CMAKE ERROR	Na počítači by mělo být nainstalováno jen Visual Studio 2019. Abychom kompletně odinstalovali Visual Studio z počítače, tak přejdeme do adresáře Program Files (x86)\Microsoft Visual Studio\Installer\resources\app\layout a spustíme .\InstallCleanup.exe -full. V souboru CARLA\Util\InstallersWin\install_zlib.bat na řádku 51 přepsat verzi ZLIB na 2.1.13. (Aktuální verzi zlib dokážeme zjistit na stránce zlib.net)
B2 ERROR	Vyskytuje se při používání některých verzí Python. Já použil verzi 3.7.9, kterou CARLA simulátor podporuje.
LNK2038 error	Chyba kvůli Windows SDK, CARLA vyžaduje verzi Windows SDK 8.1.
LNK2001 error	Spustit příkaz "make clean", poté vymazat složku "Build" z adresáře "CARLA" a spustit znovu příkaz "make PythonAPI". Často se i po tomto postupu objeví tato chyba znova. Pokud ano, tak spustíme opakovaně příkaz "make PythonAPI" a chyba by měla zmizet.
C1083	Spustit opakovaně příkaz "make PythonAPI". Zkontrolovat, jestli máme nainstalovanou 64-bitovou verzi Python.
Failed to find XercesC	V souboru C:\CARLA\CARLA\Util\InstallersWin\install_xercesc.bat přepište na řádku 43 verzi xercesc do tvaru: set XERCESC_VERSION=3.2.3

Adresář PythonAPI/CARLA/dist neexistuje, ačkoliv po spuštění příkazu "make PythonAPI" se objevila zpráva o úspěšném založení	Pokud se toto stane, tak projděte výstup v příkazovém řádku, kde bude pravděpodobně nějaká z chyb výše. Pokud žádnou takovou chybu nenajdete, tak vymažte složku Build, spusťte příkaz "make clean" a znovu příkaz "make PythonAPI".

Tabulka 1 - Chyby při instalaci

2.3 VYTVOŘENÍ MAPY V ROADRUNNER

Roadrunner je nástroj od Mathworks, který umožňuje návrh 3D map pro simulace autonomního řízení. Tento nástroj je kompatibilní s CARLA simulátorem. Roadrunner nabízí možnost přidávat značky, svodidla, budovy nebo zeleň. Také se zde dá nadefinovat trasa vozidel v pruzích nebo semafory.

PRVKY V ROADRUNNER

V Roadrunner najdeme tyto prvky:

- Budovy
- Silnice
- Různé typy poškození vozovky
- Svodidla, ploty
- Značení
- Zeleň
- Semafory
- Světla
- Koleje
- Automobily
- Elektrické vedení

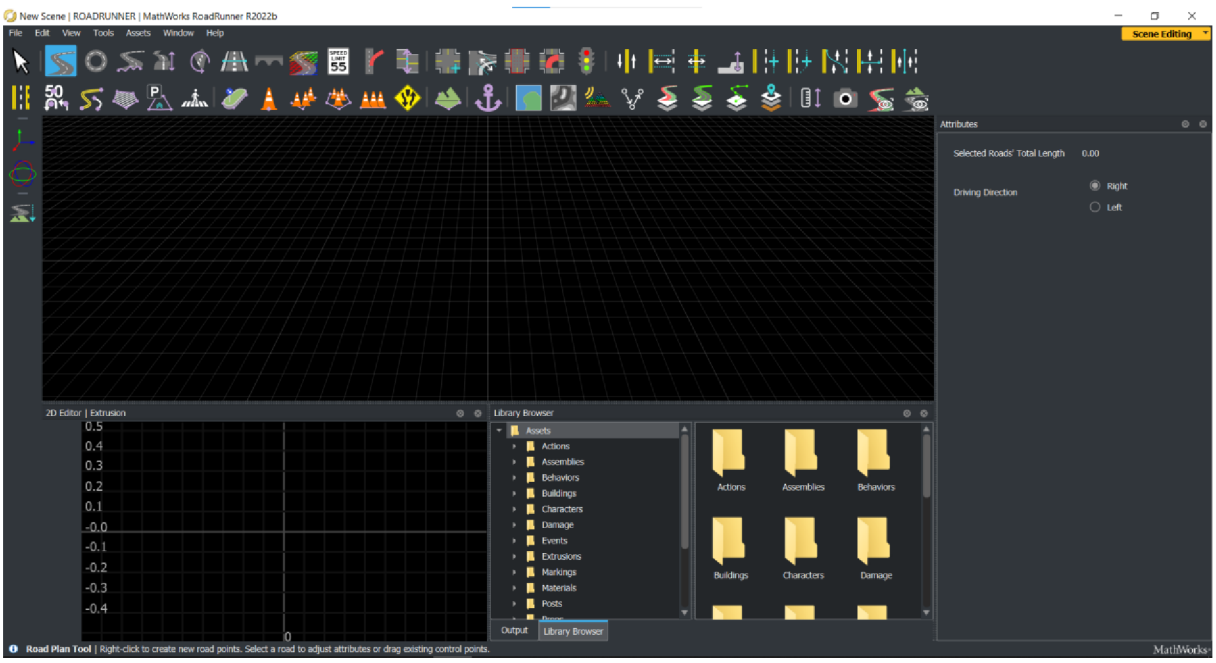
POUŽÍVÁNÍ ROADRUNNER

V této podkapitole bude ukázána práce s programem Roadrunner.

Po zapnutí programu Roadrunner je nejdříve potřeba vytvořit novou scénu a nový projekt.

Obrázek 23 - Roadrunner: založení nové scény

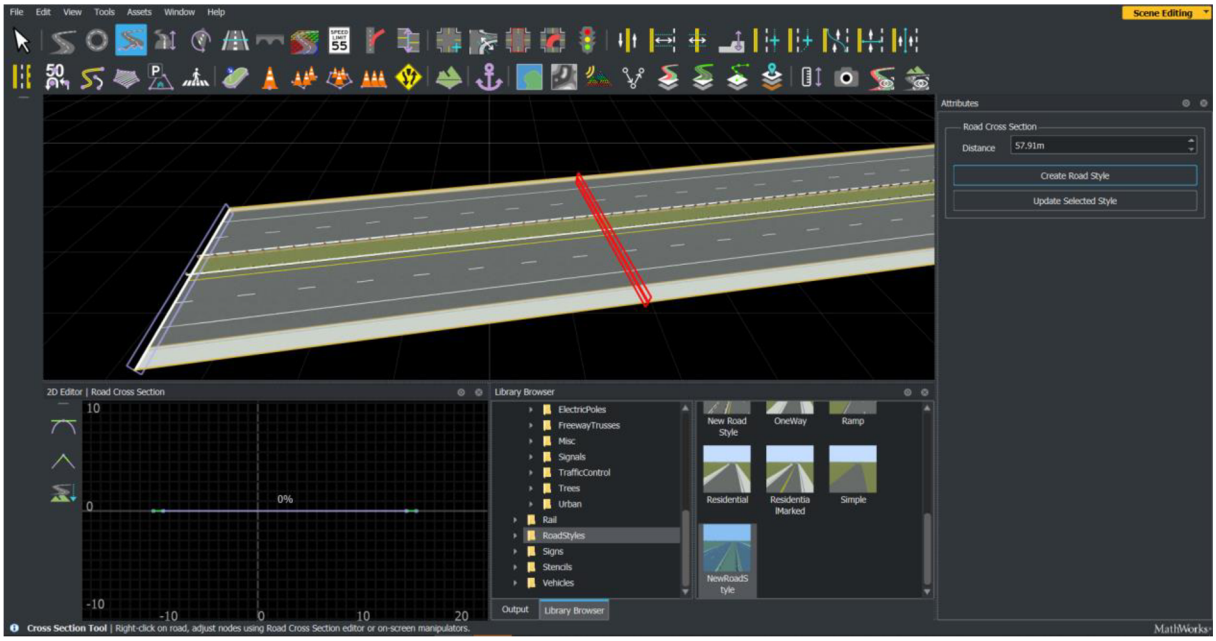
Po vytvoření scény se otevře editor, ve kterém se vytváří již samotná mapa. V horní části editoru je lišta s různými funkcemi pro vytváření a editaci map. Na pravé straně je okno s atributy, které se mění podle prvku, který aktuálně upravujeme, např. u vytváření cest je zde možnost zvolit materiál silnice nebo u vytváření značek barvu, text, rotaci apod. Ve spodní části okna se nachází 2D editor pro úpravu jednotlivých prvků a knihovna, ze které vybíráme prvky, které chceme do scény přidat, jako jsou budovy, značky, typy cest atd.



Obrázek 24 – Roadrunner: editor

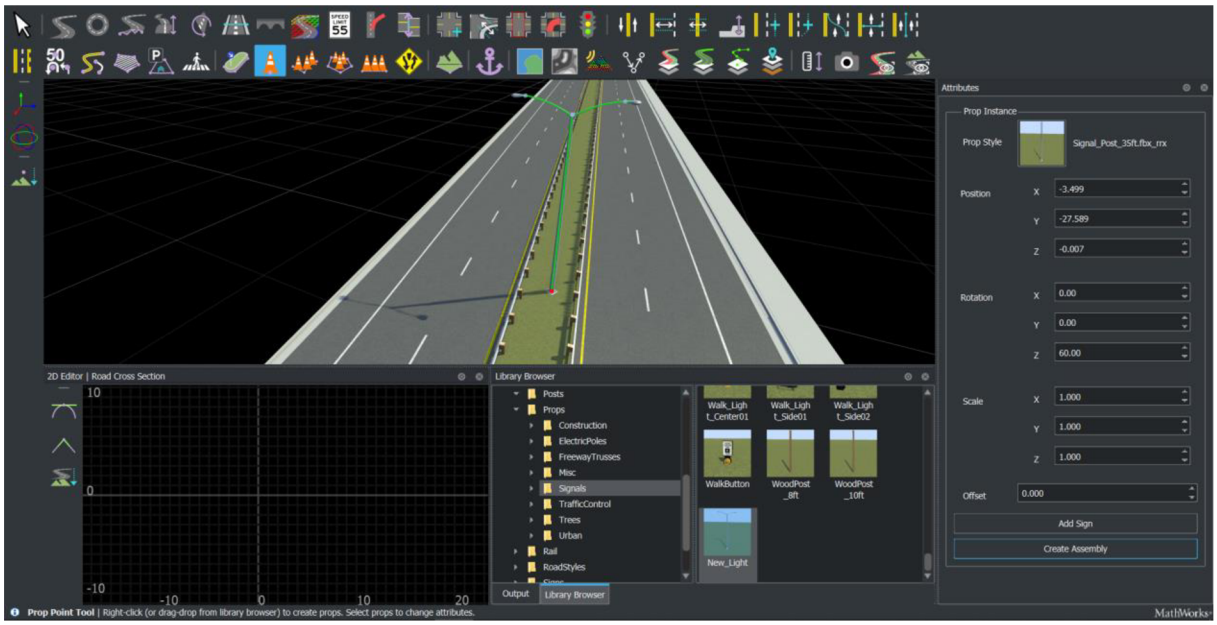
Nyní můžeme přejít na vytváření stylu silnice:

- V horní liště vybereme nástroj "Road Plan Tool" a z knihovny, ze složky "RoadStyles", vybereme styl cesty, který chceme použít. Pravým tlačítkem myši klikneme nejprve na začátek silnice a poté na konec úseku. Nyní je silnice vytvořena.
- Pomocí nástrojů "Lane Tool", "Lane Width Tool", "Lane Add Tool" můžeme upravit styl cesty.
- Pomocí nástroje "Lane Tool" v horní liště nadefinujeme funkci každého pruhu naší vytvořené silnice. Pro jízdní pruhy zvolíme typ "Driving" a vybereme směr, ve kterém se v těchto pruzích bude jezdit, pro prostřední pruh zvolíme "Curb", pro postranní pruh zvolíme "Shoulder" a na krajích silnice vytvoříme chodník pomocí typu "Sidewalk"
- V prostředním pruhu můžeme zadefinovat materiál. Z knihovny, ze složky "Material", vybereme materiál "Grass1" a přesuneme ho do kolonky "Material" v atributech pro prostřední pruh.
- Z knihovny, ze složky "Extrusions", můžeme vybrat svodidla a přidat je k prostřednímu pruhu přesunutím.
- Nyní, pokud máme hotový návrh naší silnice, tak můžeme použít nástroj "Cross Section Tool", kliknout pravým tlačítkem na cestu a v okně "Attributes" odklikneme "Create Road Style" a v knihovně se nám vytvoří nový styl silnice, který můžeme později využít.



Obrázek 25 - Roadrunner: vytvoření nového stylu silnice

Dále vytvoříme prvky, které potřebujeme do naší scény. Vytvoření pouličních světel je pomocí spojení prvků ze složky "Signals", která je podsložkou "Props". Vybereme "SignalPost" a na něj přidáme "Luminaire_Arm" a na konci ramena přidáme světlo "Luminaire_Head". Takhle vytvořené světlo nyní spojíme do jednoho celku pomocí tlačítka "Create Assembly".

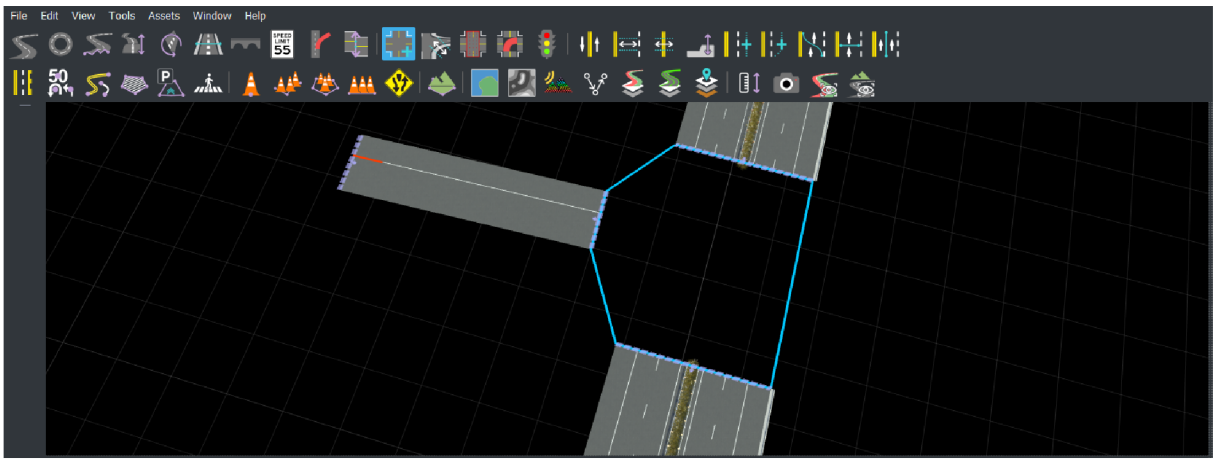


Obrázek 26 - Roadrunner: vytvoření nových prvků

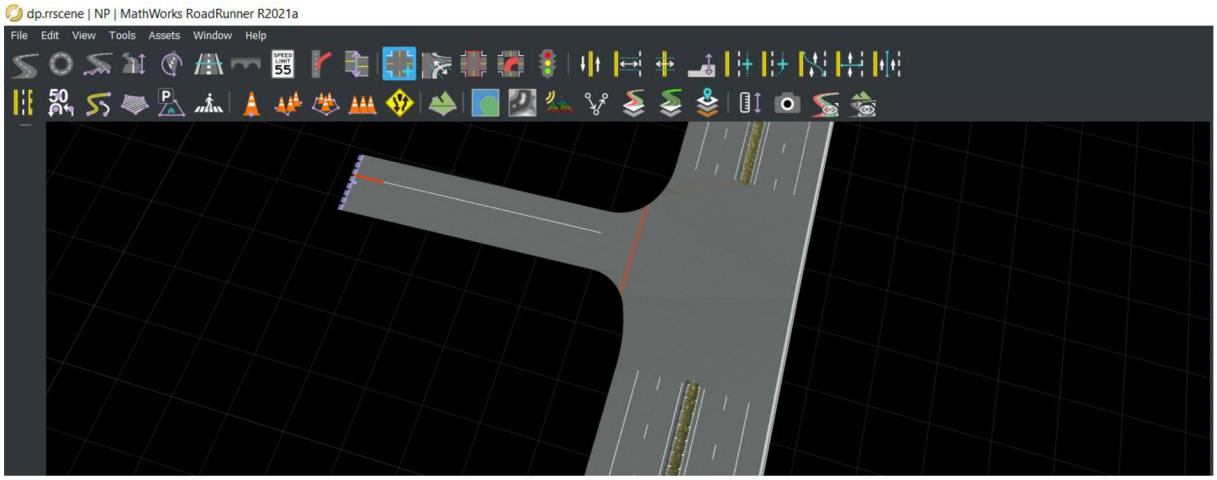
Dalším důležitým prvkem jsou značky a ukazatele. Nejdříve ze složky "Signals" vybereme sloup, na který budeme naši značku chtít přidat, a přetáhneme ho do mapy. Spoustu předvytvořených značek můžeme najít ve složce "Signs". Pokud nenalezneme potřebnou značku v této složce, tak můžeme nahrát vlastní obrázek. Obrázek přetáhneme do složky "Signs" v Roadrunneru a pravým kliknutím na nahraný soubor změníme typ souboru na "Sign". Vybranou značku můžeme přetáhnout na vybraný sloup v mapě. Druhou možností je vytvořit ukazatel přímo v Roadrunneru. Když přidáme do mapy sloup ze složky "Signals", tak v atributech máme možnost tlačítka "Add Sign". Tímto tlačítkem vytvoříme značku přímo na sloupu. V horní liště vybereme nástroj "Sign Tool" a ve 2D editoru ve spodní části okna můžeme tuto značku upravovat, přidat obrazce nebo text. Na takhle vytvořenou značku můžeme také přidat předvytvořené značky ze složky "Signs" nebo označení směru ze složky "Stencils". V pravé části okna můžeme upravovat ostatní atributy, jako je např. barva značky.

Obrázek 27 - Roadrunner: vytvoření značky

Dalším důležitým prvkem cest jsou křižovatky. Křižovatku vytoříme pomocí nástroje "Custom Junction Tool" a to tak, že pravým tlačítkem klikneme na konce cest z jednotlivých směrů, poté co vybereme všechny cesty, které vedou do křižovatky, zmáčkneme mezerník. Tímto se nám vytvoří nová křižovatka.

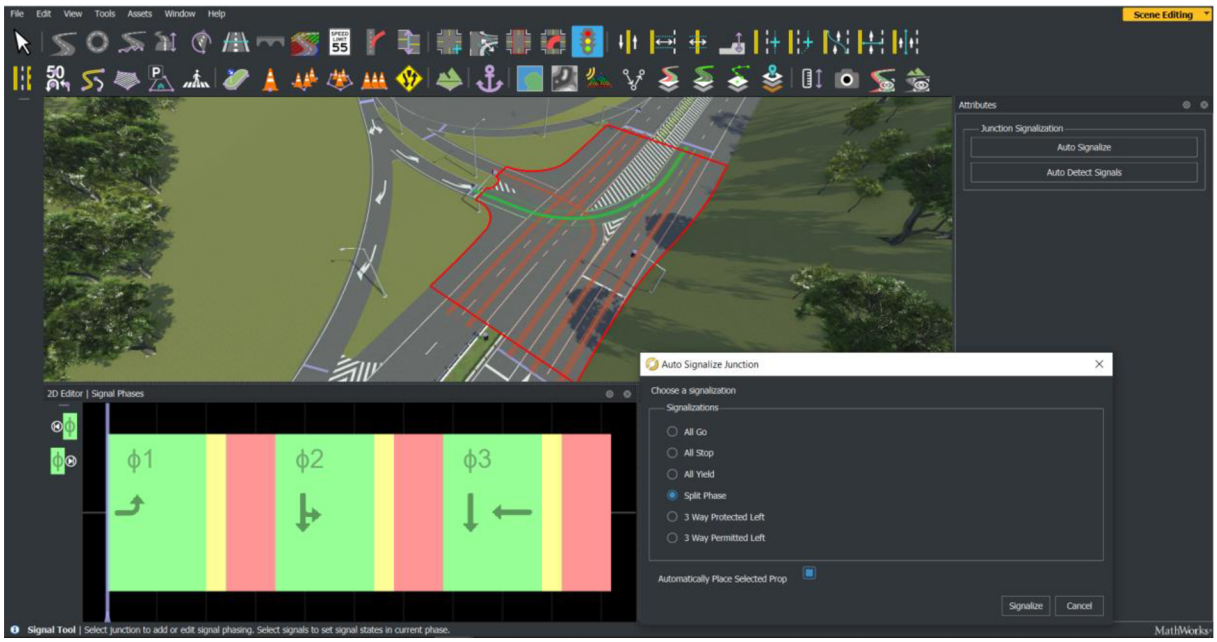


Obrázek 28 - Roadrunner: vytvoření křižovatky



Obrázek 29 - Roadrunner: vytvořená křižovatka

S křižovatkami taky souvisí světelná signalizace (semafory). Po tom, co vytvoříme křižovatku, můžeme pomocí nástroje "Signal Tool" vytvořit světelnou signalizaci pro tuto křižovatku. Vybereme křižovatku a v okně "Attributes" vybereme "Auto Signalize", otevře se nám okno s možnostmi signalizace. Já jsem pro svoji křižovatku vybral možnost "Split Phase" a zakliknul možnost "Automatically Place Selected Prop". Ve 2D editoru se nyní ukazují fáze signalizace. Po kliknutí na jednotlivé fáze můžeme vidět, které směry jsou zastaveny a které průjezdné, zároveň se zde dají měnit časy jednotlivých fází. Zároveň po kliknutí na jednotlivé semafory v mapě můžeme v atributech nastavit jednotlivé intervaly a styl semaforu. Nástrojem "Prop Point Tool" můžeme přidané semafory posouvat nebo některé z nich vymazat. Zde se ujistěme, že semafor je typu "Signal_xLight_Post01", protože jiný typ semaforu nefunguje správně v pozdější práci s Unreal Engine. Pokud zde je jiný typ semaforu, tak můžeme vybrat jiný typ ze složky "Signals".



Obrázek 30 - Roadrunner: Signal Tool

Při používání nástroje "Road Plan Tool" je možnost ve 2D editoru měnit výšku silnice. Pokud chceme vytvořit most, tak jedné cestě zadáme nižší výšku. Nyní zapneme nástroj "Road Construction Tool" a vybereme úsek na vyšší cestě a v okně s atributy změníme typ cesty na most ("Bridge").

Obrázek 31 - Roadrunner: vytvoření mostu

Pro vytvoření výjezdové silnice použijeme nástroj "Slip Road Tool". S použitím tohoto nastroje vytvoříme cestu, která se odpojí z naší hlavní cesty a připojí se na vedlejší cestu.

Obrázek 32 - Roadrunner: Slip Road Tool

Důležitým bodem před exportováním mapy do CARLA simulátoru je zkontrolovat, jestli na všech křižovatkách a odbočkách jsou správně nadefinované směry a možnosti odbočení. Tohle zkontrolujeme pomocí nástroje "Maneuver Tool" a kliknutím na konkrétní křižovatku, popř. zatáčku.

Obrázek 33 - Roadrunner: Maneuver Tool

Dalšími nástroji jsou:

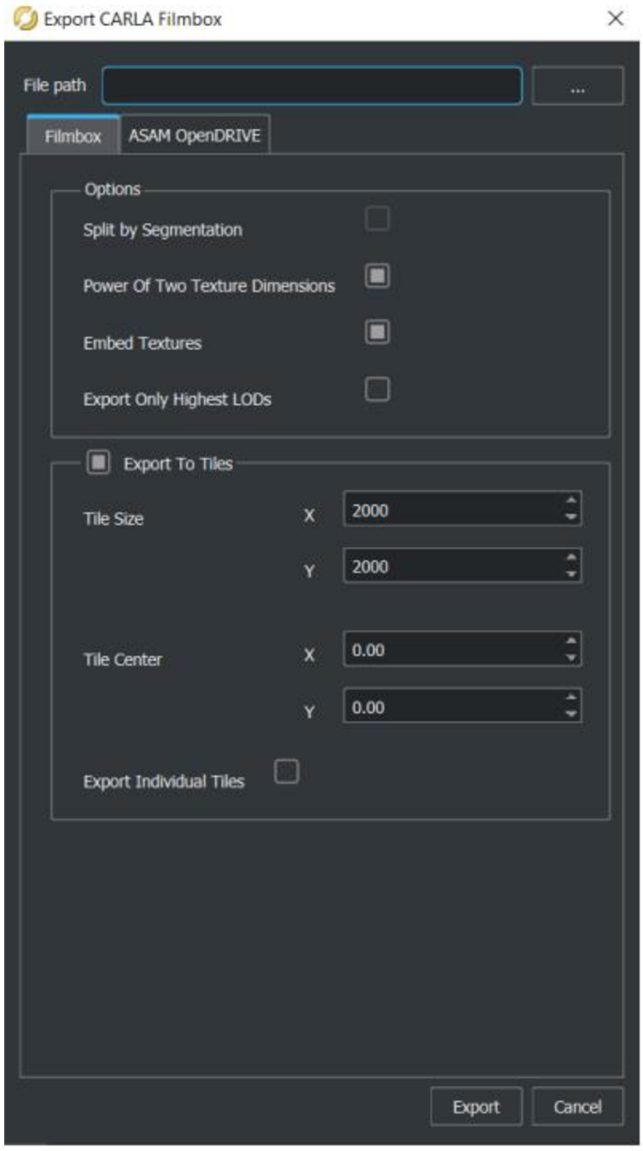
- Road Circle Tool – nástroj pro vytváření kruhových objezdů
- Cross Section Tool – nástroj pro úpravu konkrétního jednoho segmentu silnice. Pomocí tohoto nástroje se dá nastavit výška a sklon úseku. Tento nástroj taky umožní zadefinovat nový styl silnice, který se potom dá využívat dále na jiných úsecích.
- Road Height Tool – nástroj pro úpravu výšky konkrétního úseku silnice
- Road Superelevation Tool – nástroj pro úpravu úhlu natočení konkrétního úseku silnice. Můžeme ho využít pro vytvoření nakloněných zatáček nebo ramp.
- Road Chop Tool – nástroj pro rozdělení silnice do více jednotlivých úseků
- Road Construction Tool – nástroj pro vytvoření mostů
- Road CRG Tool – tento nástroj umožňuje vytvářet, vizualizovat nebo importovat data o povrchu silnic ve formátu CRG. OpenCRG je standard pro popis povrchu vozovky.
- Road Speed Limits Tool – nástroj pro nastavení rychlosti na daném úseku
- Split Road Tool – nástroj pro vytvoření výjezdové silnice
- Road Offset Tool – nástroj pro vychýlení jednoho úseku silnice vůči jinému úseku
- Custom Junction Tool – nástroj pro vytváření křižovatek
- Corner Tool – nástroj pro úpravu tvaru křižovatky
- Maneuver Tool – nástroj pro zobrazení nadefinovaných směrů a možností odbočení pro konkrétní odbočku nebo křižovatku a možnost upravovat tyto prvky.
- Signal Tool – nástroj pro vytvoření světelné signalizace (semaforů) pro vytvořenou křižovatku
- Lane Tool – nástroj pro definování vlastností jízdních pruhů
- Lane Width Tool – nástroj pro změnu šířky pruhu v konkrétním úseku
- Lane Offset Tool – nástroj pro odsazení (změnu výšky) konkrétního místa na silnici
- Sidewalk Height Tool – nástroj pro změnu výšky chodníku
- Lane Add Tool – nástroj pro přidání dalšího pruhu po celém zvoleném úseku silnice
- Lane Form Tool – nástroj pro přidání dalšího pruhu i s určením začátku a tvaru nového pruhu
- Lane Carve Tool – nástroj pro vyřezání konkrétní části jízdního pruhu, např. při vytváření odbočovacího pruhu
- Lane Chop Tool – nástroj pro useknutí pruhu v jednom místě

- Lane Split Tool – nástroj pro rozdělení pruhu po celé délce na dvě části
- Lane Marking Tool – nástroj pro značení jízdních pruhů. Různé druhy značení můžeme najít ve složce "Markings". Jeden pruh můžeme značit více typy značení, pomocí pravého tlačítka rozdělíme pruh na více částí.
- Marking Point Tool – nástroj pro přidání označení přímo na povrch silnice. Tyto označení najdeme ve složce "Stencils".
- Marking Curve Tool – nástroj pro vytvoření čáry na jakémkoliv místě. Typ čáry se vybírá ze složky "Markings"
- Marking Polygon Tool – nástroj pro vytvoření polygonu na jakémkoliv místě. Výplň polygonu se vybírá ze složky "Markings"
- Parking Tool – nástroj pro vytvoření parkovacích míst
- Crosswalk and Stop Line Tool – nástroj pro vytvoření přechodů v křižovatce a zastavovací čáry před křižovatkou. Přechod se vytvoří kliknutím pravým tlačítkem postupně na rohy křižovatky, které chceme přechodem spojit. Zastavovací čáry se vytvoří pravým kliknutím na modré značky při označení křižovatky.
- Traffic Island Tool – nástroj pro vytvoření dopravního ostrůvku na libovolném místě
- Prop Point Tool – nástroj pro přidání a upravování jednotlivých prvků, jako jsou budovy, stromy, značky, apod.
- Prop Curve Tool – nástroj pro vytvoření křivky, na kterou se po celé délce přidá zvolený prvek (stromy, budovy, apod.)
- Prop Polygon Tool - nástroj pro vytvoření polygonu, do kterého se rovnoměrně přidá zvolený prvek (stromy, budovy, apod.)
- Prop Span Tool – nástroj pro přidání konkrétního prvku ke křivce náležící jinému prvku, např. okraji silnice nebo chodníku
- Sign Tool – nástroj pro úpravu značek, přidávání textu, změna barvy, apod.
- Surface Tool – nástroj pro vytvoření libovolného povrchu okolo již vytvořených cest, např. plochy pro parkování, plocha pro čerpací stanici nebo přírodní terén
- Road Anchor Tool – nástroj, který přidá na určité místo tzv. "kotvu". Tento nástroj se vyplatí využívat při používání Roadrunner Scenario, kde pomocí kotev můžeme definovat pozici jednotlivých aktérů
- Aerial Imagery Tool – nástroj pro vložení leteckých snímků pro použití jako reference pro vytvoření map. Podporuje typy GeoTIFF (.tif, .tiff) a JPEG 2000 (.jp2).
- Elevation Map Tool – nástroj pro vložení dat výškových map typů DEM, IMG, JPEG 2000 a TIFF
- Point Cloud Tool – nástroj pro vložení dat z lidarů, tzv. mračna bodů.
- Vector Data Tool – nástroj pro vložení vektorových dat, například soubory Shape (.shp), soubory OpenStreetMap® (.osm), soubory GeoJSON (.json, .geojson) a soubory GPS Exchange (.gpx)
- OpenDRIVE Viewer Tool – nástroj pro vizualizaci OpenDRIVE dat
- Scene Builder Tool – nástroj pro vložení a vytváření dat z HD map. Aktuálně jsou podporovány mapy z HERE HD Live Map, TomTom HD Map, Apollo.
- SD Map Viewer Tool – nástroj pro vložení dat z SD Map. Podporovány jsou data z Zenrin Japan Map API 3.0 a OpenStreetMap
- World Settings Tool – nástroj pro nastavení geografické pozice vytvořené scény. Dá se zde zadat přesné souřadnice, kde se má scéna nacházet, střed scény a nebo její velikost
- Measurement Tool – nástroj pro změření vzdálenosti mezi body ve scéně
- Screenshot Tool – nástroj pro pořízení aktuálního náhledu editoru

- OpenDRIVE Export Preview Tool – tento nástroj vytvoří náhled vytvořeného OpenDRIVE souboru z aktuální scény
- Scene Export Preview Tool – nástroj pro vizualizaci souboru, který bude z aktuální scény vytvořen

EXPORT MAPY DO CARLA SIMULÁTORU

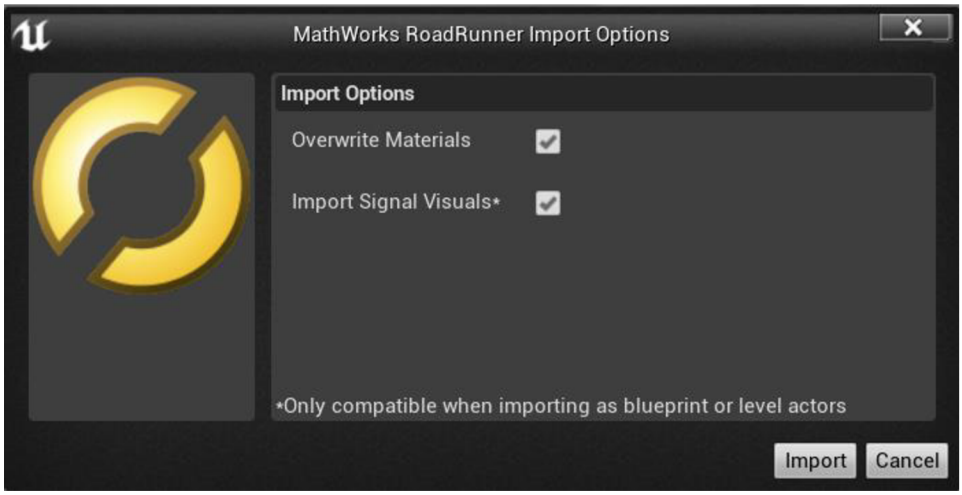
Pokud máme naši mapu již hotovou, tak ji můžeme exportovat do CARLA simulátoru. V horní nabídce rozbalíme nabídku File, zde nabídku Export a vybereme CARLA Filmbox (fbx., xodr., rrdata.xml). Vytvořená mapa může být velká maximálně 2000x2000 metrů, abychom ji mohli později naimportovat do Unreal Engine.



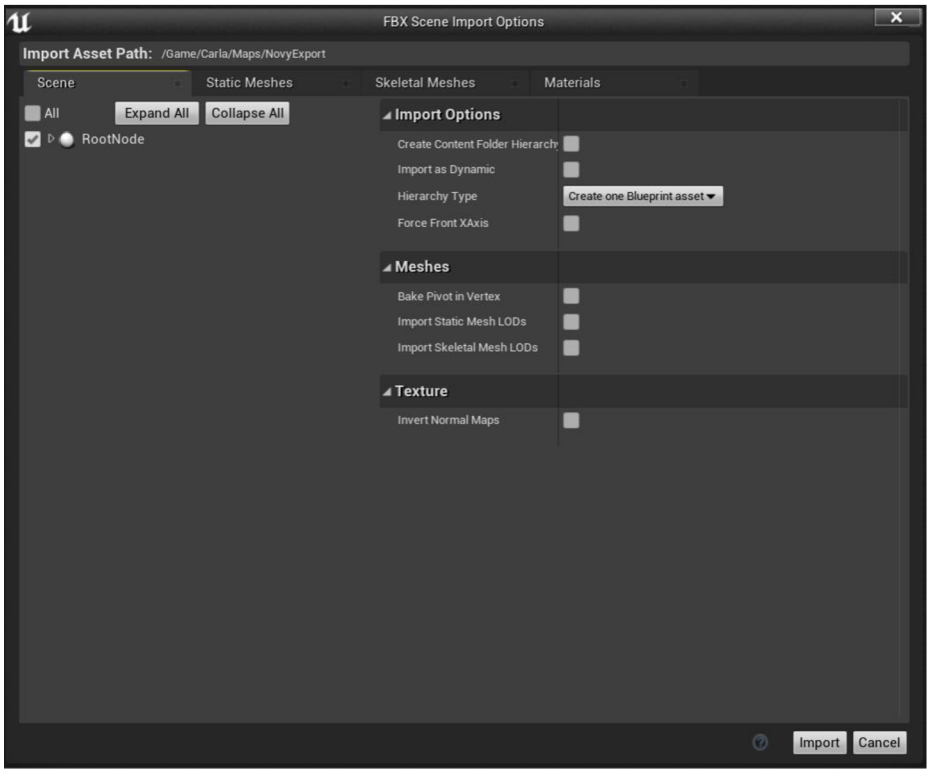
Obrázek 34 - Export mapy do CARLA simulátoru

Pro vložení vytvořené mapy z Roadrunneru budeme potřebovat stáhnout Roadrunner plugin do Unreal Engine. Tyto pluginy stáhneme zde [41]. Po stažení otevřeme složku Unreal/Plugins a

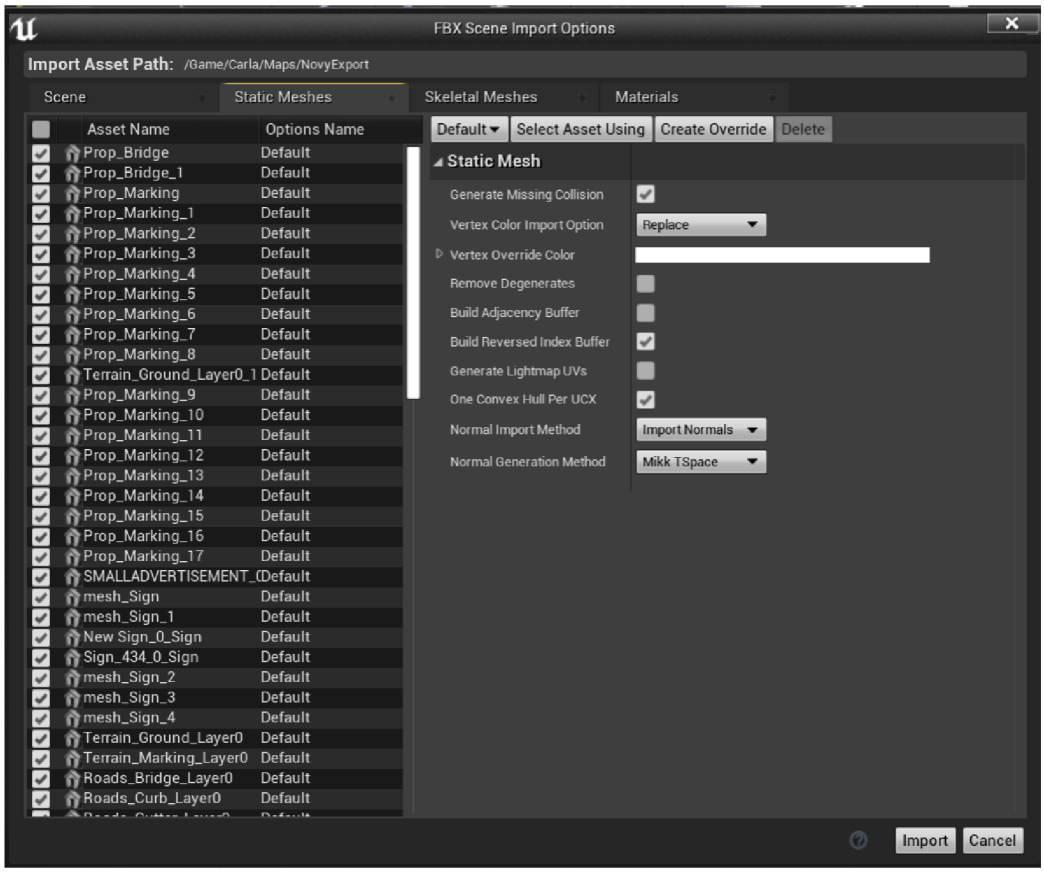
extrahujeme pluginy do složky CARLA/Unreal/CARLAUE4/Plugins. Následně ve složce "CARLAUE4" klikneme pravým tlačítkem na "CARLAUE4.uproject" a vybereme "Generate Visual Studio project files". Nyní pomocí příkazu "make launch", který spustíme v příkazovém řádku "x64 Native Tools Command Prompt for VS 2019", zapneme Unreal Engine. Nyní si vytvoříme složku v "Content Browser" ve spodní části Unreal Engine. Po vytvoření složky klikneme na tlačítko "Add/Import" a zde "Import to (aktuální adresář)" a vybereme soubor typu fbx., který jsme vygenerovali ze softwaru Roadrunner. Ve vyskakovacím okně zaškrtneme obě možnosti "Overwrite Materials" i "Import Signal Visuals" a klikneme na tlačítko "Import". V dalším vyskakovacím okně nadefinujeme nastavení podle obrázku níže a zmáčkneme "Import".



Obrázek 35 - Unreal Roadrunner Import nastavení



Obrázek 36 - Unreal Roadrunner Import nastavení



Obrázek 37 - Unreal Roadrunner Import nastavení

V případě, že je vkládaná mapa velká, tak proces trvá poměrně dlouho. Po dokončení se nám naše mapa načte v hlavním okně Unreal Engine. Zároveň pomocí záložky "File" a tlačítka "Save Current As..." naši mapu uložíme.

Druhou možností vložení mapy do Unreal Engine je přes příkaz "make import". Soubory, které nám vygeneroval Roadrunner, nakopírujeme do složky "Import". Poté v příkazovém řádku spustíme v CARLA adresáři příkaz "make import". Tato metoda nebyla vybrána jako hlavní, protože nedokáže naimportovat vytvořené značky a jiné prvky z Roadrunneru.

2.4 PRÁCE S CARLOU

V Unreal Engine si načteme naši vytvořenou mapu nebo si vybereme z map, které jsou předvytvořené. Poté, co se mapa načte, tak v pravém horním rohu klikneme na tlačítko "Play" (při zmenšeném okně je schováno pod dvěma šipkami). Nyní se nám zapnul tzv. svět a můžeme ho začít ovládat pomocí Python API.

CARLA má několik základních prvků:

- Klient – modul, který komunikuje se serverem. Přes klienta se získávají informace nebo ovládá simulace. Může být spuštěno více klientů v jeden čas.
- Svět (World) – naše simulační prostředí. Vždy existuje jen jeden svět.
- Aktéři (Actors) - všechny prvky v simulaci, jako jsou auta, chodci, sensory, apod.
- Blueprint – vlastnosti našich aktérů.

Kód, který využíváme ke komunikaci se serverem, má základní bloky, které nesmí chybět. Prvním takovým blokem je vložení základních knihoven, včetně CARLA knihovny. Tato knihovna se nachází v souboru typu egg. ve složce \CARLA\PythonAPI\CARLA\dist. [42]

```
import glob
import os
import sys

try:
      sys.path.append(glob.glob('../CARLA/dist/CARLA-*%d.%d-%s.egg' % (
        sys.version_info.major,
        sys.version_info.minor,
        'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
except IndexError:
    pass

import CARLA
```

Dále načteme všechny Python knihovny, které budeme využívat. Pokud některou z nich nemáme nainstalovanou, tak ji doinstalujeme pomocí příkazu "pip install [název_knihovny]".

```
import CARLA
import math
import random
import time
import numpy as np
import cv2
import open3d as o3d
from matplotlib import cm
```

V dalším bloku se připojíme k serveru, spojíme se se světem a načteme knihovnu s blueprinty [43]

```
actor_list = []

try:
      client = CARLA.Client('localhost', 2000)
      client.set_timeout(20.0)

      world = client.get_world()

      blueprint_library = world.get_blueprint_library()
```

VOZIDLO

Vozidlo, které do simulace přidáme, tak je součástí knihovny blueprintů, proto nejdříve jedno vozidlo z této knihovny vybereme a následně ho přidáme do simulačního světa [44].

```
bp = blueprint_library.filter('typ_vozidla')

vehicle =
world.spawn_actor(bp,random.choice(world.get_map().get_spawn_points()))
```

SENZORY

V následující části bude popsáno, jak do simulace přidat senzor. Nejdříve potřebujeme najít senzor v knihovně blueprintů pomocí příkazu:

```
blueprint_library.find('název_senzoru')
```

V následujícím příkazu se zadají informace o posunutí a natočení senzorů vůči vozidlu, na kterém se senzor bude nacházet:

```
CARLA.Transform(CARLA.Location(x=?,y=?,z=?),CARLA.Rotation(pitch=?,yaw=?,
roll=?))
```

Inicializaci senzoru a jeho přidání na vozidlo provedeme příkazem:

```
world.spawn_actor(sensor_blueprint,sensor_location,attach_to=vehicle)
```

Pro získání dat ze senzoru je využívána method listen(). Tento příkaz je závolán vždy, když senzor přijme nějaké data a parametr, který vstupuje do této funkce, se liší podle typu senzoru. Zmíněné části kódu se využijí vždy a dále záleží na uživateli, co přesně potřebuje ze simulace získat. Dále budou sepsány jednotlivé senzory, které se dají v Carle využít, jaké parametry se dají nastavit apod. [44]

RGB KAMERA

RGB kamera funguje jako klasická kamera. Uživatel může nastavit i post-processing, jako ztmavení okrajů obrazovky, přidání šumu, rozostření objektů, aj.. Základní parametry, které se dají u kamery nastavit, jsou:

- bloom intensity – nastavení intenzity tzv. bloom efektu, který zvýrazňuje světla vycházející z okrajů jasných oblastí na snímku, tak aby více reprezentoval skutečnou kameru
- fov – horizontální zorné pole ve stupních
- fstop – nastavení doby expozice
- image_size_x – šířka obrázku v pixelech
- image_size_y – výška obrázku v pixelech
- iso – citlivost kamery
- gamma – nastavení gama korekce
- lens_flare_intensity – nastavení odlesku objektivu
- sensor_tick – doba mezi snímky
- shutter_speed – rychlost uzávěrky [45]

```
camera_blueprint = blueprint_library.find('sensor.camera.rgb')

camera_location = CARLA.Transform(CARLA.Location(x=0.75, z=1.4))
camera = world.spawn_actor(sensor_blueprint,
sensor_location,attach_to=vehicle)
camera.listen(self.save_rgb_image)
```

HLOUBKOVÁ KAMERA

Hloubková kamera získává raw data (minimálně zpracovaná data) a zjišťuje vzdálenost od jednotlivých pixelů a z toho vytvoří hloubkovou mapu. CARLA nabízí možnost převést tyto RGB data do černobílého formátu s hodnotami v intervalu [0,255]. Základními vlastnostmi hloubkové kamery jsou:

- image_size_x – šířka obrázku v pixelech
- image_size_y – výška obrázku v pixelech
- fov – horizontální zorné pole ve stupních
- sensor_tick – doba mezi snímky [45]

```
depth_camera_bp = bp_lib.find('sensor.camera.depth')
depth_camera_bp.set_attribute('image_size_x', str(disp_size[0]))
depth_camera_bp.set_attribute('image_size_y', str(disp_size[1]))
depth_camera=world.spawn_actor(depth_camera_bp,camera_init_trans,
attach_to=vehicle)
depth_camera.listen(lambda image: depth_callback(image, sensor_data))
```

KAMERA PRO SÉMANTICKOU SEGMENTACI

Tato kamera klasifikuje každý objekt tak, že jej zobrazí jinou barvou podle jeho označení. Při spuštění simulace je všem aktérům přiřazeno označení. Vlastnosti, které se dají u sémantické kamery nastavit, jsou stejné jako u hloubkové kamery. [45]

```
sem_camera_bp = bp_lib.find('sensor.camera.semantic_segmentation')
sem_camera_bp.set_attribute('image_size_x', str(disp_size[0]))
sem_camera_bp.set_attribute('image_size_y', str(disp_size[1]))
sem_camera=world.spawn_actor(sem_camera_bp,camera_init_trans,
attach_to=vehicle)
sem_camera.listen(lambda image: sem_callback(image, sensor_data))
```

GNSS SENZOR

Senzor globálního navigačního satelitního systému (GNSS) poskytuje aktuální zeměpisnou šířku a zeměpisnou délku na základě výpočtu metrické polohy vozidla s počáteční polohou. Tomuto senzoru lze nastavit i šum nebo velikost odchylky. Atributy GNSS senzoru jsou:

- sensor_tick – doba mezi záznamy
- noise_alt_bias – střední hodnota v modelování šumu u měření nadmořské výšky
- noise_alt_stddev – směrodatná odchylka v modelování šumu u měření nadmořské výšky
- noise_lat_bias – střední hodnota v modelování šumu u měření zeměpisné šířky
- noise_lat_stddev – směrodatná odchylka v modelování šumu u měření zeměpisné šířky
- noise_lon_bias – střední hodnota v modelování šumu u měření zeměpisné délky
- noise_lon_stddev – směrodatná odchylka v modelování šumu u měření zeměpisné délky
- noise_seed – nastavení náhodného šumu [45]

```
gnss_bp = bp_lib.find('sensor.other.gnss')
gnss_sensor = world.spawn_actor(gnss_bp, CARLA.Transform(),
attach_to=vehicle)
gnss sensor.listen(lambda event: gnss callback(event, sensor data))
```

IMU SENZOR

Inerciální měřící jednotka poskytuje data z akcelerometru, gyroskopu a kompasu aktéra. Je zde možnost nastavit šum i odchylku u akcelerometru a gyroskopu. Vlastnosti IMU jednotky:

- noise_accel_stddev_x – směrodatná odchylka šumu u akcelerometru v ose x
- noise_accel_stddev_y – směrodatná odchylka šumu u akcelerometru v ose y
- noise_accel_stddev_z – směrodatná odchylka šumu u akcelerometru v ose z
- noise_gyro_bias_x – střední hodnota šumu u gyroskopu v ose x
- noise_gyro_bias_y – střední hodnota šumu u gyroskopu v ose y
- noise_gyro_bias_z – střední hodnota šumu u gyroskopu v ose z
- noise_gyro_stddev_x – směrodatná odchylka šumu u gyroskopu v ose x
- noise_gyro_stddev_y – směrodatná odchylka šumu u gyroskopu v ose y
- noise_gyro_stddev_z – směrodatná odchylka šumu u gyroskopu v ose z
- noise_seed – nastavení náhodného šumu
- sensor_tick – doba mezi záznamy [45]

```
imu_bp = bp_lib.find('sensor.other.imu')
imu_sensor = world.spawn_actor(imu_bp, CARLA.Transform(),
attach_to=vehicle)
imu_sensor.listen(lambda event: imu_callback(event, sensor_data))
```

LIDAR

Tento senzor představuje rotační Lidar. Virtuální paprsky jsou vysílány z pohledu senzoru, aby našli kolidující místa. Konfigurovatelné nastavení pro lidar je:

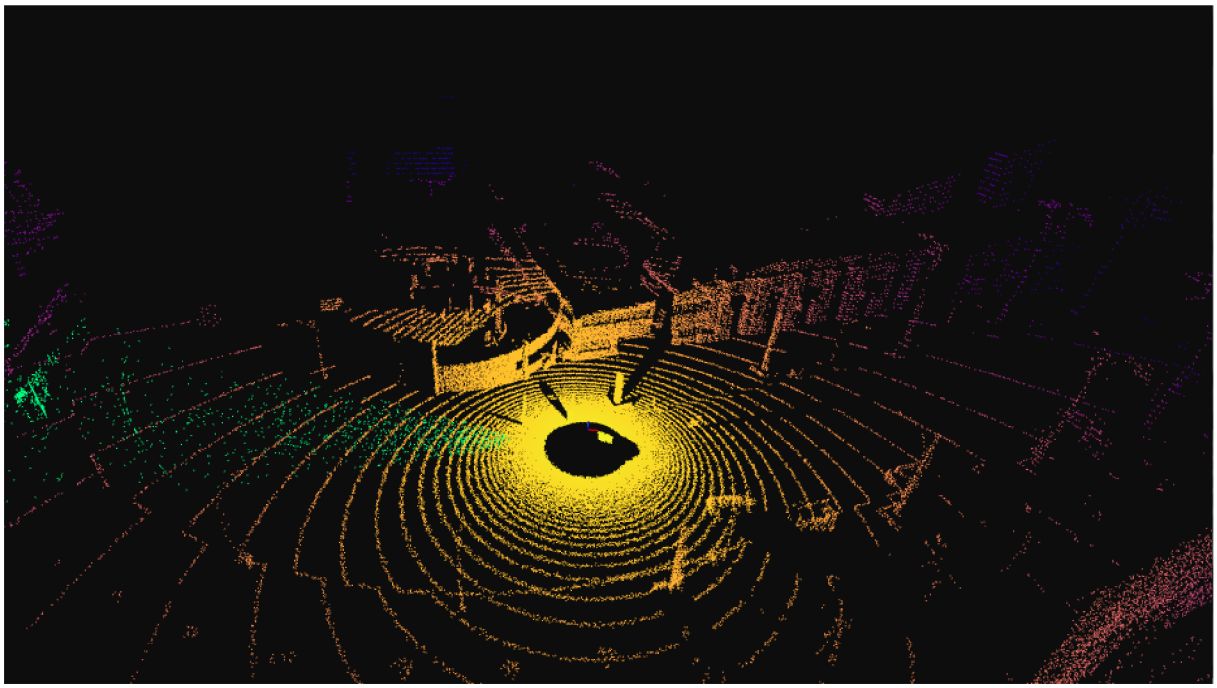
- channels – počet laserů
- range – maximální měřitelná vzdálenost
- points_per_second – body generované všemi lasery za sekundu
- rotation_frequency – frekvence otáčení lidaru
- upper_fov – zorné pole laseru na nejvyšší pozici
- lower_fov – zorné pole laseru na nejnižší pozici
- horizontal_fov - horizontální zorné pole ve stupních
- atmosphere_attenuation_rate – koeficient ztráty intenzity lidaru na metr
- dropoff_general_rate – podíl náhodně vyřazených bodů
- dropoff_intensity_limit – limitní hodnota intenzity, při jejímž překročení nedojde k vyřazení bodů
- dropoff_zero_intensity – pravděpodobnost, že každý bod s nulovou intenzitou bude vyřazen
- noise_stdev – nastavení směrodatné odchylky šumu
- sensor_tick – doba mezi záznamy [45]

```
lidar_bp =
self.world.get_blueprint_library().find('sensor.lidar.ray_cast_semantic')
lidar_bp.set_attribute('range', '100')
lidar = self.world.spawn_actor(lidar_bp, transform, attach_to=attached)
lidar.listen(self.save_semanticlidar_image)
```

RADAR

Data z radaru obsahují polární souřadnice, vzdálenost a rychlost každého měřeného bodu. U radaru můžeme nadefinovat tyto parametry:

- horizontal_fov – horizontální zorné pole ve stupních
- points_per_second – body generované všemi lasery za sekundu
- range – maximální měřitelná vzdálenost
- sensor_tick – doba mezi záznamy
- vertical_fov – vertikální zorné pole ve stupních [45]



Obrázek 38 - Lidar a Radar

```
radar_bp = self.world.get_blueprint_library().find('sensor.other.radar')
radar = self.world.spawn_actor(radar_bp, transform, attach_to=attached)
radar.listen(self.save_radar_image)
```

SENZOR KOLIZE

Senzor kolize nepředstavuje žádný skutečný senzor, ale dává informaci o kolizi, kdykoliv se aktér srazí s nějakým objektem. Může se využít pro vyhodnocení algoritmů autonomního řízení. Senzor kolizí nemá žádné konfigurovatelné vlastnosti. [45]

```
collision_bp = bp_lib.find('sensor.other.collision')
collision_sensor=world.spawn_actor(collision_bp,CARLA.Transform(),
attach_to=vehicle)
collision_sensor.listen(lambda event:collision_callback(event,
sensor_data))
```

SENZOR PŘEJETÍ JÍZDNÍHO PRUHU

Stejně jako u senzoru kolize, tento senzor nepředstavuje skutečný senzor. Definice map mu dává informaci, kdykoliv dojde k najetí na jízdní pruh. Může se využít pro vyhodnocení algoritmů autonomního řízení. Tento senzor nemá žádné konfigurovatelné vlastnosti. [45]

```
lane_inv_bp = bp_lib.find('sensor.other.lane_invasion')
lane_inv_sensor = world.spawn_actor(lane_inv_bp, CARLA.Transform(),
attach_to=vehicle)
lane_inv_sensor.listen(lambda event: lane_inv_callback(event,
sensor_data))
```

DETEKTOR PŘEKÁŽEK

Detektor překážek zaregistruje událost pokaždé, když se před naším aktérem objeví nějaký objekt. Nastavitelné parametry u detektoru překážek jsou:

- Distance – detekovatelná vzdálenost k překážce
- Hit_radius – poloměr detekování překážek
- Only_dynamics – detekování jen dynamických prvků
- Debug_linetrace – viditelnost detektoru
- Sensor_tick – doba mezi záznamy [45]

```
obstacle_bp = bp_lib.find('sensor.other.obstacle')
obstacle_bp.set_attribute('hit_radius','0.5')
obstacle_bp.set_attribute('distance','50')
obstacle_sensor = world.spawn_actor(obstacle_bp, CARLA.Transform(),
attach_to=vehicle) obstacle_sensor.listen(lambda event:
obstacle_callback(event, sensor_data, camera, K))
```

DVS KAMERA

Událostní kamery (senzory dynamického vidění) mají rozdíl oproti klasickým kamerám v tom, že nesnímají obraz s pevnou frekvencí, ale asynchronně ve formě proudu událostí (v závislosti na změně jasu na jednotlivý pixel). Jsou to tedy kamery, které dokáží poskytovat vysoce kvalitní informace i v náročných dynamických podmínkách. U DVS kamery můžeme nastavit:

- positive_treshold – hranice spojená s přírůstkem jasu
- negative_treshold – hranice spojená s úbytkem jasu
- sigma_positive_treshold – směrodatná odchylka bílého šumu pro pozitivní události
- sigma_negative_treshold – směrodatná odchylka bílého šumu pro negativní události
- refractory_period_ns – doba, po kterou pixel nemůže vyvolat událost poté, co ji vyvolal
- use_log – použít logaritmickou stupnici intenzity [45]

```
dvs_camera_bp = bp_lib.find('sensor.camera.dvs')
dvs_camera_bp.set_attribute('image_size_x', str(disp_size[0]))
dvs_camera_bp.set_attribute('image_size_y', str(disp_size[1]))
dvs_camera=world.spawn_actor(dvs_camera_bp,camera_init_trans,
attach_to=vehicle)
dvs_camera.listen(lambda image: dvs_callback(image, sensor_data))
```

OPTICKÁ PRŮTOKOVÁ KAMERA

Každý pixel zaznamenaný touto kamerou promítá rychlost daného bodu do roviny obrazu. Rychlost pixelu je zakódována v rozsahu [-2,2].

- image_size_x – šířka obrázku v pixelech
- image_size_y – výška obrázku v pixelech
- fov – horizontální zorné pole ve stupních
- sensor_tick – doba mezi snímky [45]

```
opt_camera_bp = bp_lib.find('sensor.camera.optical_flow')
opt_camera_bp.set_attribute('image_size_x', str(disp_size[0]))
opt_camera_bp.set_attribute('image_size_y', str(disp_size[1]))
opt_camera = world.spawn_actor(opt_camera_bp, camera_init_trans,
attach_to=vehicle)
opt_camera.listen(lambda image: opt_callback(image, sensor_data))
```

CHODCI

Kromě vozidel jsou důležitou součástí simulací i chodci. Pro vytvoření chodce nejdříve najdeme v knihovně blueprint pro chodce. Následně přidáme chodce do simulace, ale jen na místa, kde se chodci mohou pohybovat. Pro chodce musíme taky vytvořit ovladač, který bude ovládat jejich chování. Vytvoření chodců do simulace může být pomocí následujícího kódu. [46]

```
world = client.get_world()
blueprintsWalkers                                                    =
world.get_blueprint_library().filter("walker.pedestrian.*")
walker_bp = random.choice(blueprintsWalkers)
spawn_points = []
for i in range(100):
    spawn_point = CARLA.Transform()
    spawn_point.location = world.get_random_location_from_navigation()
    if (spawn_point.location != None):
        spawn_points.append(spawn_point)
batch = []
for spawn_point in spawn_points:
    walker_bp = random.choice(blueprintsWalkers)
    batch.append(CARLA.command.SpawnActor(walker_bp, spawn_point))
results = client.apply_batch_sync(batch, True)
for i in range(len(results)):
    if results[i].error:
        logging.error(results[i].error)
    else:
        walkers_list.append({"id": results[i].actor_id})
batch = []
walker_controller_bp                                                 =
world.get_blueprint_library().find('controller.ai.walker')
for i in range(len(walkers_list)):
    batch.append(CARLA.command.SpawnActor(walker_controller_bp,
CARLA.Transform(), walkers_list[i]["id"]))
results = client.apply_batch_sync(batch, True)
for i in range(len(results)):
    if results[i].error:
        logging.error(results[i].error)
    else:
        walkers_list[i]["con"] = results[i].actor_id
```

POČASÍ

Dalším důležitým parametrem pro realistickou simulaci je kontrola počasí. To uděláme pomocí následujícího kódu [46]:

```
weather = CARLA.WeatherParameters(
    cloudyness=?,
    precipitation=?,
    sun_altitude_angle=?)

world.set_weather(weather)

print(world.get_weather())
```

POTÍŽE PŘI PRÁCI S CARLA SIMULÁTOREM

Při práci s CARLA simulátorem mohou nastat různé potíže, nejčastější jsou v tabulce níže.

NÁZEV CHYBY	POPIS A POSTUP VYŘEŠENÍ
RuntimeError: time-out of 2000ms while waiting for the simulator	Simulátor se nedokázal načíst za dobu, kterou jsme mu nastavili. V řádku client.set_timeout() změníme čas z 2 sekund na delší čas, např. 20 sekund.
ImportError: No module named 'CARLA'	Skript nemůže najít egg. soubor. Zkontrolujte, jestli existuje složka \CARLA\PythonAPI\CARLA\dist a jestli v ní je soubor typu egg.. Pokud ne, tak spusťte znovu příkaz "make PythonAPI".

Máme zapnutý autopilot, ale auto po chvíli vyjede na krajnici	Při používání autopilota musíme zapnout synchronní mod.
	`settings = world.get_settings()`
	`settings.synchronous_mode = True`
	`settings.fixed_delta_seconds = 0.05`
	`world.apply_settings(settings)`

Po zapnutí vlastní mapy z Roadrunneru se auta propadají pod silnici	Mapy v Roadrunneru musí být vytvořené tak, že alespoň nějaká část je v nulové nadmořské výšce. Pokud ji vytvoříme celou v nenulové nadmořské výšce, tak se může stát, že auta propadají silnicí.

Tabulka 2 - Potíže při práci se simulátorem CARLA

2.5 VÝSLEDKY

V této kapitole bych rád zhodnotil CARLA simulátor. Zde je několik pozitiv simulátoru CARLA:

- CARLA simulátor nabízí obrovské množství možností pro simulaci. Přesně modeluje reálné podmínky, jako jsou dopravní situace, povětrnostní podmínky nebo povrch vozovky.
- CARLA má vytvořené Python API pro ovládání simulace. Python API je dobře zdokumentováno a i ukázáno na předpřipravených příkladech.
- CARLA podporuje velké množství senzorů, které se používají v autonomních autech.
- Výhodou je také široká uživatelská základna, se kterou je možné řešit různé potíže.

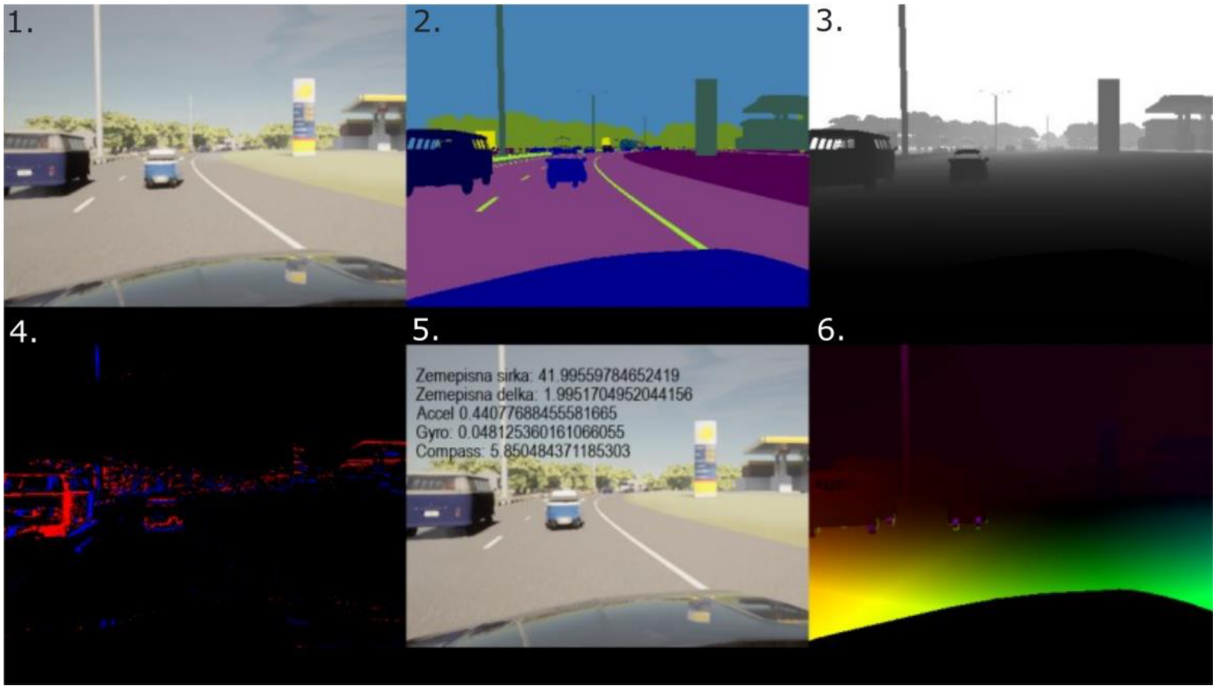
Negativa CARLA simulátoru spatřuji v:

- Neúplné dokumentaci – zejména instalační manuály nejsou tak robustní, aby si mohl uživatel CARLA simulátor nainstalovat bez potíží.
- Výpočetním výkonu – pro náročnější simulace je potřebné výkonnější vybavení.
- CARLA je open-source projekt a tudíž při různých aktualizacích dojde k neočekávaným chybám a CARLA nemusí fungovat. Uživatel musí být neustále v obraze o tom, co se vyvíjí a co je nového v aktualizacích.

V programu Roadrunner byla vytvořena mapa, která simuluje ulici Hradecká v Brně. Tuto cestu jsme nejdříve projeli skutečným autem s kamerou. Na obrázcích níže jsou porovnávací obrázky mezi daty z CARLA simulátoru a z reálné kamery.

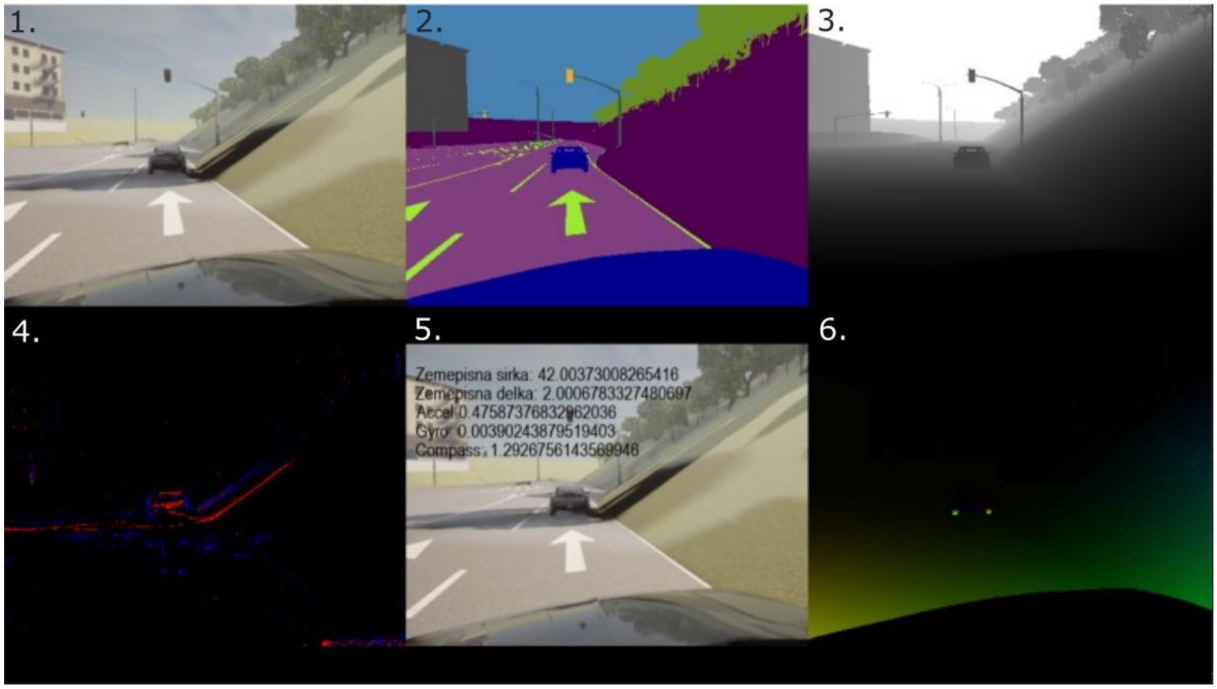
Data v obrázcích jsou řazeny následovně:

1. RGB kamera
2. Kamera pro sémantickou segmentaci
3. Hloubková kamera
4. DVS kamera
5. GNSS senzor, IMU senzor
6. Optická průtoková kamera



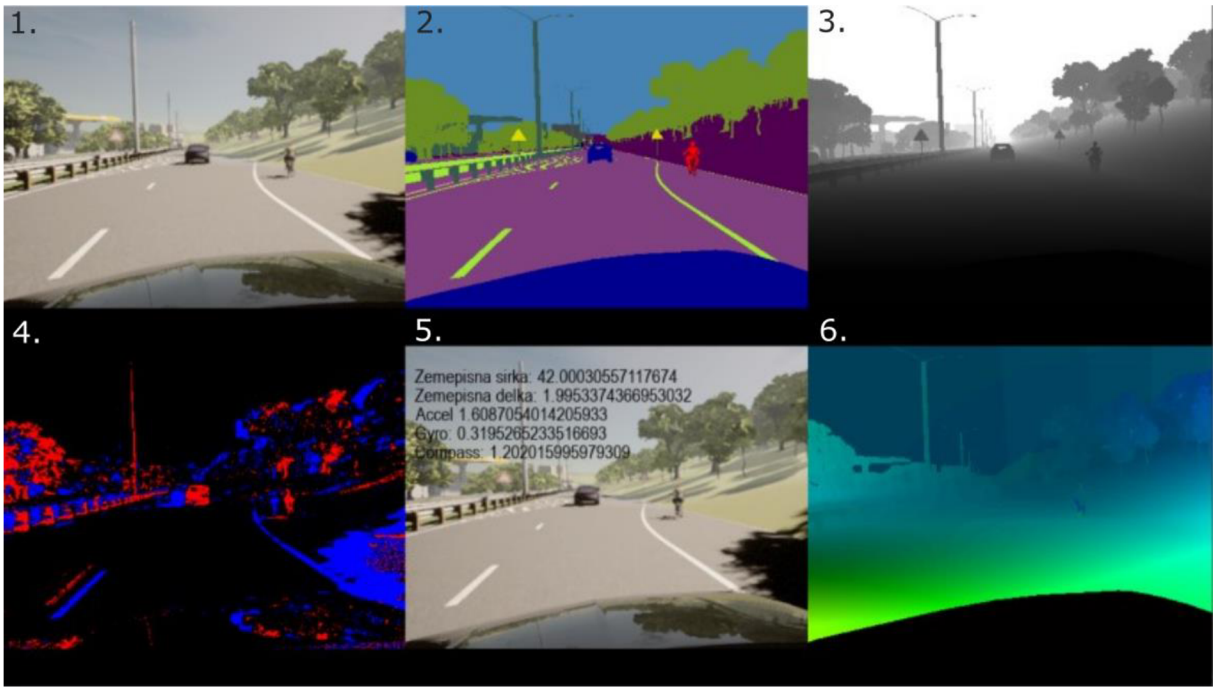
Obrázek 39 – Data ze simulátoru CARLA I

Obrázek 40 - Kamera z reálného vozu I



Obrázek 41 - Data ze simulátoru CARLA II

Obrázek 42 - Kamera z reálného vozu II



Obrázek 43 - Data ze simulátoru CARLA III

Obrázek 44 - Kamera z reálného vozu III

CARLA Simulátor mi vyšel z dostupných simulátorů jako nejlepší možnost. V průběhu práce jsem však přišel na to, že práce se simulátorem je náročná a zdlouhavá, což často zhoršuje omezená dokumentace. Výhodou je široká komunita uživatelů a fóra, kde je možnost některé věci dohledat. CARLA má velké množství předvytvořených map, které jsou velmi dobře zpracované, bohužel vytvoření nových scénářů je časově náročné. Poslední velkou nevýhodou je náročnost z hlediska výpočetního výkonu. I přes tyto problémy je důležité poznamenat, že simulátor CARLA je užitečným nástrojem v oblasti autonomního řízení. Jeho schopnost

modelovat složité scénáře z reálného světa, podporovat různé senzory a řídicí algoritmy z něj činí jeden z nejlepších simulátorů na trhu. Je však nezbytné, aby uživatelé byli připraveni na obtíže, s nimiž se mohou setkat.

ZÁVĚR

V rámci rešeršní části této práce byly představeny simulační nástroje pro autonomní vozidla. Byla provedena rešerše v oblasti senzoriky a softwaru využívaného v autonomním řízení. Výsledkem této rešerše bylo rozhodnutí použít CARLA simulátor jako simulační nástroj. Toto rozhodnutí bylo založeno na několika faktorech:

- Open-source: CARLA je open-source projekt, což umožňuje přizpůsobovat a rozšiřovat simulátor podle svých potřeb.
- Možnost vytvářet vlastní mapy: CARLA poskytuje uživatelům flexibilitu vytvářet vlastní simulační prostředí a mapy, což je klíčové pro testování v konkrétním prostředí.
- Ovládání simulace: CARLA nabízí Python API, přes které můžeme simulaci kontrolovat. Je možné ovládat aktéry v simulaci, senzoriku nebo počasí, tak aby se simulace přibližovala co nejvíce reálným podmínkám.
- Realistické simulační prostředí: CARLA používá herní Unreal Engine a vizualizace je na výborné úrovni. Zároveň jsou zde nadefinovány fyzikální modely a realistické chování vozidel, což vytváří věrnou simulaci skutečného světa.
- Dokumentace a aktivní vývoj: CARLA má rozsáhlou dokumentaci a je neustále dále vyvíjena. Zároveň existuje široká základna uživatelů, kteří navzájem sdílí své zkušenosti.

V další části práce byla vytvořena simulační mapa prostřednictvím programu Roadrunner. Konkrétně byla vytvořena replika ulice Hradecká v Brně, která byla předtím projeta skutečným vozidlem s kamerou. Tato část práce podrobně popsala postup práce s programem Roadrunner.

Následovala instalace CARLA simulátoru, která byla provedena krok za krokem. Bylo popsáno a demonstrováno použití základních bloků Python API nezbytných pro správnou funkci simulace. Dále byly prezentovány dostupné senzory v simulátoru, včetně jejich výstupů, parametrů a inicializačních postupů.

V závěrečné části práce bylo provedeno srovnání výstupů kamery z reálného vozidla s výstupy z CARLA simulátoru. Toto srovnání poskytlo ucelený pohled na realističnost a přesnost CARLA simulátoru a potvrdilo jeho schopnost poskytovat spolehlivé simulační prostředí pro vývoj a testování autonomních vozidel. Zároveň zde byly shrnuty poznatky z práce s CARLA simulátorem a obtíže, které se vyskytly. Mezi největší problémy simulátoru CARLA bezpochyby patří to, že práce s ním je náročná a zdlouhavá, což často zhoršuje nepřesná dokumentace.

Cílem této práce bylo navrhnout a představit simulační prostředí pro autonomní vozidla. Vytvoření tohoto prostředí poskytuje základní kámen pro další projekty, které se zaměřují na výzkum algoritmů autonomního řízení a analýzu dynamiky vozidel. Věřím, že výsledky této práce budou sloužit jako solidní základ pro budoucí projekty v oblasti autonomního řízení.

SEZNAM POUŽITÝCH ZDROJŮ

[1] THE MATHWORKS, INC. Automated Driving Toolbox. Automated Driving Toolbox [online]. [cit. 2023-03-25]. Dostupné z: https://www.mathworks.com/products/automated-driving.html

[2] CARLA TEAM. CARLA Simulator [online]. 2023 [cit. 2023-03-25]. Dostupné z: https://CARLA.org/

[3] CARLA TEAM. Maps and navigation [online]. In: . [cit. 2023-03-25]. Dostupné z: https://CARLA.readthedocs.io/en/latest/core_map/

[4] MECHANICAL SIMULATION CORPORATION. CarSim Overview [online]. [cit. 2023-03-25]. Dostupné z: https://www.carsim.com/products/carsim/index.php

[5] THE MATHWORKS, INC. PreScan [online]. [cit. 2023-03-25]. Dostupné z: https://www.mathworks.com/products/connections/product_detail/prescan.html

[6] Simcenter Prescan [online]. SIEMENS PRODUCT LIFECYCLE MANAGEMENT SOFTWARE INC. SIEMENS. 2019 [cit. 2023-03-25]. Dostupné z: http://www.shzongkun.com/Uploads/file/Siemens-XC/Siemens%20PLM%20Simcenter%20PreScan%20fs_tcm27-70172.pdf

[7] SIEMENS. Simcenter Prescan 2019.1 [online]. In: . [cit. 2023-03-25]. Dostupné z: https://resources.sw.siemens.com/en-US/video-simcenter-prescan-20191

[8] KAUR, Prabhjot, Samira TAGHAVI, Zhaofeng TIAN a Weisong SHI. A Survey on Simulators for Testing Self-Driving Cars [online]. 2021 [cit. 2023-03-25]. Dostupné z: https://arxiv.org/pdf/2101.05337.pdf. Department of Computer Science, Wayne State University, Detroit, USA.

[9] OPEN SOURCE ROBOTICS FOUNDATION, INC. Car and city simulation in Gazebo [online]. In: . 2017 [cit. 2023-03-25]. Dostupné z: https://www.youtube.com/watch?v=97JRYhKLhSY&t=2s

[10] LG ELECTRONICS. SVL SIMULATOR [online]. In: . [cit. 2023-03-25]. Dostupné z: https://www.svlsimulator.com/

[11] LG ELECTRONICS INC. LGSVL Simulator: An Autonomous Vehicle Simulator\ [online]. In: . [cit. 2023-03-25]. Dostupné z: https://www.svlsimulator.com/docs/archive/2020.06/getting-started/

[12] UBER ATG. AVS: Autonomous Visualization System [online]. In: . [cit. 2023-03-25]. Dostupné z: https://avs.auto/#

[13] WIGGERS, Kyle. VENTUREBEAT. Uber open-sources Autonomous Visualization System, a web-based platform for vehicle data [online]. In: . [cit. 2023-03-25]. Dostupné z: https://avs.auto/#/

[14] NVIDIA CORPORATION. NVIDIA DRIVE End-to-End Solutions for Autonomous Vehicles [online]. In: . [cit. 2023-03-25]. Dostupné z: https://developer.nvidia.com/drive#drive-sim

[15] NVIDIA CORPORATION. NVIDIA DRIVE Constellation [online]. In: . [cit. 2023-03-25]. Dostupné z: https://developer.nvidia.com/drive/constellation

[16] RFPRO. SIMULATION SOFTWARE: THE WORLD'S MOST ACCURATE SIMULATION ENVIRONMENT [online]. In: . [cit. 2023-03-25]. Dostupné z: https://rfpro.com/simulation-software/

[17] RFPRO. K4556 – HILLSIDE [online]. In: . [cit. 2023-03-25]. Dostupné z: https://rfpro.com/digital-models/digital-road-models-public-roads/k4556-hillside/

[18] COGNATA. Autonomous and ADAS Vehicles Simulation Software [online]. In: . [cit. 2023-03-25]. Dostupné z: https://www.cognata.com/

[19] CISION US INC. Cognata Collaborates with Microsoft to Help Mobility Companies Evaluate Sensors for Automated Driving [online]. In: . [cit. 2023-03-25]. Dostupné z: https://www.prnewswire.com/il/news-releases/cognata-collaborates-with-microsoft-to-help-mobility-companies-evaluate-sensors-for-automated-driving-301714423.html

[20] VIRES SIMULATIONSTECHNOLOGIE GMBH. VTD - Enabling safety validation in autonomous driving and ADAS system simulation [online]. In: . [cit. 2023-03-25]. Dostupné z: https://vires.mscsoftware.com/#

[21] IPG AUTOMOTIVE GMBH. CarMaker [online]. In: . [cit. 2023-03-25]. Dostupné z: https://ipg-automotive.com/en/products-solutions/software/carmaker/

[22] UDACITY. A self-driving car simulator built with Unity [online]. In: . [cit. 2023-03-25]. Dostupné z: https://github.com/udacity/self-driving-car-sim

[23] MICROSOFT RESEARCH. Home - AirSim [online]. [cit. 2023-03-25]. Dostupné z: https://microsoft.github.io/AirSim/

[24] LI, Quanyi, Zhenghao PENG, Lan FENG, Qihang ZHANG, Zhenghai XUE a Bolei ZHOU. MetaDrive [online]. [cit. 2023-03-25]. Dostupné z: https://metadriverse.github.io/metadrive/

[25] LI, Quanyi, Zhenghao PENG, Lan FENG, Qihang ZHANG, Zhenghai XUE a Bolei ZHOU. MetaDrive introduction video [online]. In: . [cit. 2023-03-25]. Dostupné z: https://www.youtube.com/embed/3ziJPqC_-T4

[26] CHEVALIER-BOISVERT, Maxime, Florian GOLEMO, Yanjun CAO, Bhairav MEHTA a Liam PAULL. Self-driving car simulator for the Duckietown universe [online]. [cit. 2023-03-25]. Dostupné z: https://github.com/duckietown/gym-duckietown

[27] VARGHESE, Jaycil Z. a Randy G. BOONE. Overview of Autonomous Vehicle Sensors and Systems [online]. [cit. 2023-04-10]. Dostupné z: http://ieomsociety.org/IEOM_Orlnado_2015/papers/140.pdf

[28] S. Campbell et al., " ," 2018 29th Irish Signals and Systems Conference (ISSC), Belfast, UK, 2018, pp. 1-4, doi: 10.1109/ISSC.2018.8585340.

[29] MICHIGAN TECHNOLOGICAL UNIVERSITY. Benchmarking Sensors for Vehicle Computer Vision Systems [online]. In: . [cit. 2023-04-10]. Dostupné z: https://www.mtu.edu/mtri/research/project-areas/transportation/sensors-platforms/benchmarking-sensors/

[30] [FAN, Rui, Jianhao JIAO, Haoyang YE, Yang YU, Ioannis PITAS a Ming LIU. Key Ingredients of Self-Driving Cars [online]. 2019 [cit. 2023-04-10]. Dostupné z: https://arxiv.org/pdf/1906.02939.pdf]

[31] CARLA TEAM. Windows build - CARLA Simulator [online]. [cit. 2023-05-05]. Dostupné z: https://CARLA.readthedocs.io/en/latest/build_windows/

[32] KITWARE, INC. Download | CMake [online]. [cit. 2023-05-05]. Dostupné z: https://cmake.org/download/

[33] SOFTWARE FREEDOM CONSERVANCY, INC. Git - Downloads [online]. [cit. 2023-05-05]. Dostupné z: https://git-scm.com/downloads

[34] FREE SOFTWARE FOUNDATION, INC. Make for Windows [online]. [cit. 2023-05-05]. Dostupné z: https://gnuwin32.sourceforge.net/packages/make.htm

[35] PAVLOV, Igor. 7-Zip [online]. [cit. 2023-05-05]. Dostupné z: https://www.7-zip.org/

[36] PYTHON SOFTWARE FOUNDATION. Download Python [online]. [cit. 2023-05-05]. Dostupné z: https://www.python.org/downloads/

[37] CHOUDHARY, Vineet. DEVELOPER INSIDER. Download Visual Studio 2019 Web Installer / ISO (Community / Professional / Enterprise) [online]. [cit. 2023-05-05]. Dostupné z: https://www.python.org/downloads/

[38] MICROSOFT. Archiv sady Windows SDK a emulátoru | Microsoft Developer [online]. [cit. 2023-05-05]. Dostupné z: https://developer.microsoft.com/cs-cz/windows/downloads/sdk-archive/

[39] GITHUB, INC. GitHub [online]. [cit. 2023-05-05]. Dostupné z: https://github.com/

[40] EPIC GAMES, INC. The most powerful real-time 3D creation tool - Unreal Engine [online]. [cit. 2023-05-05]. Dostupné z: https://www.unrealengine.com/en-US

[41] THE MATHWORKS, INC. Downloading Plugins - MATLAB & Simulink [online]. [cit. 2023-05-05]. Dostupné z: https://www.mathworks.com/help/roadrunner/ug/downloading-plugins.html

[42] CARLA TEAM. Python API - CARLA Simulator [online]. [cit. 2023-05-05]. Dostupné z: https://CARLA.readthedocs.io/en/latest/python_api/

[43] CARLA TEAM. Foundations - CARLA Simulator [online]. [cit. 2023-05-05]. Dostupné z: https://CARLA.readthedocs.io/en/latest/foundations/

[44] CARLA TEAM. Actors - CARLA Simulator [online]. [cit. 2023-05-05]. Dostupné z: https://CARLA.readthedocs.io/en/latest/core_actors/

[45] CARLA TEAM. Sensors reference - CARLA Simulator [online]. [cit. 2023-05-05]. Dostupné z: https://CARLA.readthedocs.io/en/latest/ref_sensors/

[46] CARLA TEAM. Python API tutorial - CARLA Simulator [online]. [cit. 2023-05-05]. Dostupné z: https://CARLA.readthedocs.io/en/0.9.7/python_api_tutorial/

SEZNAM OBRÁZKŮ

SEZNAM TABULEK