Úvod

V dnešní době je na dálnicích velká snaha o dynamické řízení dopravy obzvláště v místech s uzavírkami jízdních pruhů či při vysoké hustotě provozu. Uzavření jízdního pruhu může nastat prakticky kdykoliv a to plánovaně (např. při pravidelné údržbě) a nebo neplánovaně (dopravní nehoda). Z toho vyplývá úkol pokusit se v tento moment řídit dopravu jak nejlépe to jde. Ani experti se ovšem neshodují, co znamená slovo nejlépe z předchozí věty. Může to být např. konkrétní hodnota intenzity či rychlosti, dojezdový čas či další parametry dopravního proudu.

Pokud jsme schopni získat surová data z daného úseku, např. intenzitu vztaženou na minutu pro každý pruh, a následně je vyhodnotíme, dostaneme přesné informace o úseku. Poté můžeme také tyto data vzít a pokusit se nasimulovat danou oblast. Otázkou zůstává jak. Ideálně tak, aby simulace byla blízká k již získaným informacím o úseku. V této práci se zaměříme na mikroskopické modely, podle kterých budeme simulace provádět. To znamená, že budeme simulovat každé jednotlivé vozidlo.

Po simulacích vyhodnotíme jednotlivé modely a zjistíme, které modely odpovídají reálným situacím. Předpoklad je takový, že některé modely budou přesnější např. při vysokých hustotách, některé zase naopak.

Ve chvíli, kdy získáme tento přehled modelů, můžeme již přesněji analyzovat danou situaci a do budoucna budeme vědět, že daný model simuluje tento stav nejlépe.

Hlavní cíle práce tedy spočívají v simulaci dálničního provozu na základě získaných reálných dat, a to pomocí různých mikroskopických modelů. Dalším krokem je porovnat výsledky z těchto simulací s reálnými daty. Tento způsob lze využít jako další možnost při řízení dané dálniční oblasti či při predikování např. plánovaných uzavírek.

Cíle práce

Cíle práce lze chronologicky formulovat následovně:

- Seznámení se se základními typy mikroskopických modelů (hlavně jejich matematickým popisem) v dopravě pro použití na dálnicích
- Seznámit se s kalibrací těchto modelů
- Seznámit se simulačním softwarem SUMO a možnostmi jeho využití v dané problematice.

Tato témata jsou náplní první, teoretické části práce. V druhé, praktické, části byly cíle práce následující.

- implementace všech mikroskopických modelů z teoretické části do simulačního softwaru SUMO
- kalibrace mikroskopických modelů z teoretické části

- nasimulovaní reálných dat, případně vytvoření vlastních scénářů pro mikroskopické modely
- Makroskopický popis výsledků simulace.
- Mikroskopický popis výsledků simulace.

1. Mikroskopické modely

Hlavní skupinou mikroskopických modelů jsou tzv. „Car Following“ modely. Tyto modely vycházejí z předpokladu, že pokud n-té vozidlo následuje $n-1$ vozidlo na homogenní dálnici, trajektorie n-tého vozidla bude stejná jako vozidla před ním (tedy vozidla $n-1$) až na posuny v čase a místě. (Vozidlo bude ve stejném místě jako předcházející v jiný čas a ve stejný čas bude na jiném místě.) Všechny další dispozice jednotlivých modelů v sobě tento předpoklad obsahují.

Na konci každého modelu budou pro přehlednost shrnuty parametry modelu, které se vyskytují v závěrečné rovnici.

1.1 Newellův model

V teorii dopravního proudu je Newellův model jeden ze základních modelů, popisujících chování řidiče ■ **vozidla?** ■ dle vozidla před ním. Řidič se snaží držet za vozidlem jedoucím před ním v konstantní vzdálenosti. Tento model vznikl v roce 1961 a postupně se vyvíjel ■ **vyvíjel se v co?** ■. Zde jej zmiňujeme hlavně z důvodu, že z Newellových úvah vychází mnohé novější modely, například *Intelligent Driver Model* (IDM), který si popíšeme později.

V článku [13] je přesně popsán nejen model, ale i jeho verifikace a porovnání s ostatními modely. Pro nás je důležitý popis modelu.

1.1.1 Popis modelu

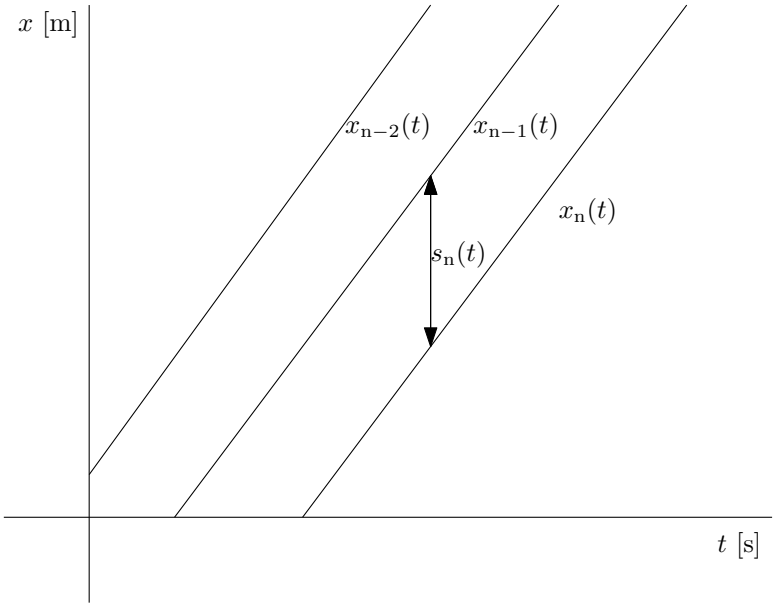
Jestliže n-té vozidlo jede za $n-1$ vozidlem (které jede za $n-2$ vozidlem atd.), cíl každého Car following modelu je zjistit závislost trajektorie n-tého vozidla $x_n(t)$ jeho pozici v čase t na $n-1$ vozidle, viz obrázek 1.1. (To také znamená, že není žádná náhodná spojitost mezi těmito vozidly.) Jestliže se $n-1$ vozidlo pohybuje konstantní rychlostí v,

$$x_{n-1} = x_n + vt,$$

n-té vozidlo také pojede průměrnou rychlostí v. Pokud by n-té vozidlo zrychlovalo, tak by došlo ke kolizi s $n-1$ vozidlem a naopak pokud by zpomalovalo, tak by se n-té vozidlo neustále vzdalovalo. Totéž platí pro všechna vozidla jedoucí za n-tým. Tento model se nezabývá zjištěním hodnoty rychlosti v, předpokládá se, že je určena buďto na základě omezení rychlosti či možnostech vozidla.

Vzdálenost $s_n = x_{n-1}(t) - x_n(t)$ mezi vozidly n a $n-1$ se může měnit v čase. Pokud je dálnice homogenní, tato vzdálenost zůstane konstantní okolo hodnoty s_n. Tato hodnota se mění na základě typu vozidla a také závisí na rychlosti v.

Předpokládejme, že existuje nějaký empirický vztah mezi rychlostí v a vzdáleností mezi vozidly s_n. Pokud rychlost v roste, je logické, že řidiči chtějí dosáhnout většího rozestupu mezi vozidly. Tato závislost mezi v a s_n je znázorněna na obrázku 1.2. Každý řidič má svoji preferovanou rychlost V_n. Jestliže rychlost v je u $n-1$ vozidla je vyšší než



■ sneslo by překreslit do TikZu ■

Obrázek 1.1: Trajektorie vozidel s konstantní rychlostí, zdroj: [13]

preferovaná rychlost u n-tého, tedy $v > \mathrm{V}_n$, znamená to, že n-té vozidlo pojede svoji preferovanou rychlostí (na obrázku 1.2 znázorněna čárkovanou čárou) a $n-1$ vozidlo mu ujede. Hodnota rychlosti v nemůže být záporná a vzdálenost mezi vozidly při nulové rychlosti by měla být níže než polopřímka lineární závislosti, viz černá tečka na obrázku 1.2 při rychlosti $v = 0$.

Nyní předpokládejme, že se $n-1$ vozidlo nějakou dobu t pohybuje konstantní rychlostí v[1] a potom náhle změní rychlost na hodnotu v'. Trajektorie vozidel n a $n-1$ mohou poté vypadat jako na obrázku 1.3. Z obrázku lze také vypozorovat jak časovou τ_n, tak prostorovou d_n mezeru mezi vozidly n a $n-1$. Z čárkovaného obdelníku poté dostáváme vztah vzdálenost mezi vozidly před změnou rychlosti s_n a po změně rychlosti s'_n,

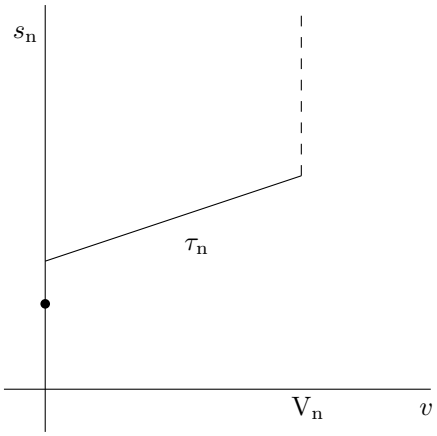
$$s_n = d_n + v\tau_n, \qquad s'_n = d_n + v'\tau_n.$$

Z toho vyplývá, že pokud leží v a v' na polopřímce z obrázku 1.2, sklon této přímky je právě τ_n a hodnota s_n při rychlosti $v = 0$ je d_n.

Ze vztahu mezi v a s_n, jak je zobrazeno na obrázku 1.2, plyne nezávislost mezer d_n a τ_n na rychlostech v, resp. v'. Pokud se tedy změní rychlost z hodnoty v' na v'', d_n a τ_n zůstanou při této změně rychlosti stejné. Lineární trajektorie vozidla $x_n(t)$ potom bude jednoduše posun v čase τ_n a místě d_n,

$$x_n(t + \tau_n) = x_{n-1}(t) - d_n. \tag{1.1}$$

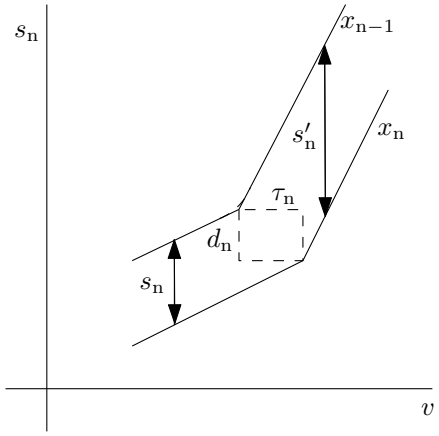
[1] hodnota rychlosti osciluje okolo hodnoty v



sneslo by překreslit do TikZu

Obrázek 1.2: Vztah mezi rychlostí v a vzdáleností mezi vozidly s_n, zdroj: [13]

U Newellova modelu platí, že n-té vozidlo bude (přibližně) kopírovat trajektorii n-1 vozidla dle vztahu (1.1) při vhodných hodnotách d_n a τ_n. Tím, jak se přesně dokáže n-té vozidlo dodržovat vztah (1.1), se Newellův model nezabývá, pouze předpokládá, že řidič je schopen se tímto vztahem řídit. To není tak těžké, protože pokud se změní rychlost vozidla $n-1$, n-tý řidič nemusí zareagovat okamžitě, ale může počkat dokud se mezera s_n nezvýší (neklesne) na hodnotu, která odpovídá nové rychlosti vozidla n-1 (viz obrázek 1.2).



Obrázek 1.3: Lineární aproximace při změně rychlosti vozidel, zdroj: [13]

Při pozorování se došlo k závěru, že každý řidič nezkoumá každého jiného řidiče na komunikaci, ale pouze jen vhodnou „makroskopickou“ část.

Hodnoty τ_n a d_n jsou přirozené, to vychází ze vztahu (1.1), kde postupným iterováním

dostaneme

$$x_n(t + \tau_n + \tau_{n-1} + \cdots + \tau_1) = x_0(t) - d_n - d_{n-1} - \cdots - d_1. \tag{1.2}$$

Hodnoty τ_n a d_n se značně liší mezi jednotlivými vozidly v závislosti na typu řidiče. První skupina řidičů raději jede blíže vozidlu před sebou $(n-1)$ až na minimální bezpečnou vzdálenost, zatímco druhá skupina naopak preferuje delší vzdálenost pro klidnou reakci na nenadálou událost. Je rozumné, aby se hodnoty τ_n a d_n lišily, a to tak, že každému jednotlivému vozidlu budou hodnoty vygenerovány z nějakého pravděpodobnostního rozdělení, jehož variační koeficient se bude blížit jedné. Při generování rychlostí vozidel by naopak měl být variační koeficient zanedbatelný.

Položíme-li

$$\overline{\tau} = \frac{1}{n}\sum_{k=1}^{n} \tau_k, \qquad \overline{d} = \frac{1}{n}\sum_{k=1}^{n} d_k, \tag{1.3}$$

kde $\overline{\tau}$ a $\overline{d}$ jsou průměrné posuny v čase resp. místě, potom průměrnou vlnovou rychlost spočteme jako podíl $\overline{d}/\overline{\tau}$.

Tento model se dá ovšem popsat nejen mikroskopicky (rychlost, mezera mezi vozidly), ale i makroskopicky (hustota k, intenzita q). Stacionární stav nastane ve chvíli, kdy se všechna vozidla budou pohybovat konstantní rychlostí s rozdílnými posuny (časovými i prostorovými).

Jestliže

$$s_n = d_n + v\tau_n,$$

a rychlost vozidel je konstantní, tak platí

$$\overline{s} = \overline{d} + v\overline{\tau}.$$

Hustotu k lze potom vypočítat jako převrácenou hodnotu průměrné mezery mezi vozidly $k = 1/\overline{s}$ a rychlost jako podíl intenzity a hustoty $v = q/k$. Tudíž platí

$$q = \frac{1}{\overline{\tau}} - \frac{\overline{d}}{\overline{\tau}}k, \tag{1.4}$$

za předpokladu, že rychlost v je menší než preferovaná rychlost jakéhokoliv vozidla V_k.

Rovnice (1.4) propojuje Newellův model s klasickými makroskopickými modely. Problém nastává ve chvíli, kdy průměrná rychlost v je vyšší, než některá s preferovaných rychlostí jednotlivých vozidel. Vozidla se v tomto modelu nemohou předjíždět a proto vznikne kongesce za vozidlem s malou preferovanou rychlostí. V následujících řádcích předpokládáme, že aktuální rychlost vozidla bude menší než preferovaná rychlost V_n.

Uvažujme, že se vozidlo n pohybuje přesně podle rovnice (1.1) a vozidlo za ním $(n-1)$ jede plynule za ním.

Rovnici (1.1) můžeme přepsat do tvaru

$$x_n(t+\tau_n) = x_n(t) + \tau_n v_n(t+T_n). \tag{1.5}$$

Rovnice (1.5) lze pro rovnoměrný pohyb zjednodušit na tvar

$$x_n(t+\tau_n) = x_n(t) + \tau_n v_n(t).$$

Tvar rovnice (1.5) plyne z matematické analýzy[2], přičemž hodnota T_n se nachází někde mezi nulou a τ_n. Pokud je funkce hladká bude přibližně

$$T_n = \frac{\tau_n}{2}. \tag{1.6}$$

Rovnici (1.5) lze přepsat na přibližný tvar

$$x_n(t+\tau_n) = x_n(t) + \tau_n v_n(t) + \tau_n T_n a_n(t), \tag{1.7}$$

který uvažuje zrychlení vozidla a_n[3]. Kombinací rovnic (1.7) a (1.1) dostáváme vztah pro rychlost

$$v_n(t+\tau_n) = \frac{1}{\tau_n}\left[x_{n-1}(t) - x_n(t)\right] - \frac{d_n}{\tau_n}. \tag{1.8}$$

Po zderivování rovnice (1.8) dostáváme vztah pro zrychlení vozidla

$$a_n(t+\tau_n) = \frac{1}{\tau_n}\left[v_{n-1}(t) - v_n(t)\right]. \tag{1.9}$$

Z rovnic (1.8) a (1.9) je vidět zřejmá závislost na vozidle, které jede před aktuálním, členem $v_{n-1}(t)$, resp. $x_{n-1}(t)$. Řidič volí svoji rychlost na základě odstupu od předchozího vozidla, resp. zrychlení na základě rychlostí.

Konečný vztah pro Newellův model je tedy

$$a_n(t) = \frac{\frac{1}{\tau_n}\left[v_{n-1}(t) - v_n(t)\right] - \frac{d_n}{\tau_n} - v_n(t)}{T_n}, \tag{1.10}$$

kde vztah mezi τ_n a T_n vychází z rovnice (1.6).

U Newellova modelu není vůbec definované čelní vozidlo a jeho vlastnosti. Je např. možné nastavit mu konstantní rychlost a vlastnosti následujících vozidel vypočítat poté rovnice (1.10).

[2] věta o střední hodnotě

[3] Předpokládejme vozidlo v klidu, které se rozjíždí. Na začátku zvolíme $\tau_n = 5s, t = 0s, T_n = \tau_n/2 = 2,5s$, potom platí

$$x_n(5) = x_n(0) + 5v_n(0+2,5).$$

Pro výpočet dráhy bereme hodnotu rychlosti v čase 2,5s. Nemůžeme vzít hodnotu v nule, vozidlo by v tomto případě stálo na místě. Přibližná hodnota bude tedy opravdu v polovině hodnoty τ_n.

1.2 Gippsův model

Další model, kterým se zde budeme zabývat, se jmenuje Gippsův. Tento model byl poprvé popsán v roce 1981 a odvolává se i na zmíněný Newellův model. I v dnešní době se tento model využívá pro simulaci dopravyy. Hlavním zdrojem pro tuto podkapitolu je článek P.G. Gippse[6]. Většina modelů, které vznikly před Gippsovým, jsou různé variace na rovnici

$$a_n(t+T_n) = l_n \frac{[v_{n-1}(t) - v_n(t)]^k}{[x_{n-1}(t) - x_n(t)]^m}, \tag{1.11}$$

kde jednotlivé proměnné mají stejný význam jako při popisu Newellova modelu 1.1. Proměnné l_n, k a m jsou parametry, které je nutno empiricky odhadnout. Tyto modely se nazývají Modely General Motors a jsou blíže rozebrány v [17].

Ačkoliv podávají tyto modely dobré výsledky v mnoha situacích, je žádoucí, aby během přepočtů poloh, rychlostí a zrychlení nějakým způsobem byl zahrnut reakční čas řidiče T_n. To si vyžaduje značné množství historických dat, pokud model bude použit v simulačním programu. Navíc, parametry l_n, k a m nemají přímou souvislost s vlastnostmi vozidla či řidiče.

Gipps se proto rozhodl, že vytvoří model, který bude splňovat následujicí podmínky:

- model by měl napodobovat chování opravdového provozu,
- parametry modelu by měly odpovídat vlastnostem řidiče a vozidla, které jsou jasné; to znamená, že pro nastavení parametrů není potřeba model kalibrovat,
- model by se měl chovat správně i v případě, že doba mezi přepočítáváním polohy a rychlosti je stejná, jako reakční čas řidiče.

1.2.1 Popis modelu

Následující model je odvozen z omezení jak řidiče, tak i vozidla a pomocí těchto omezení se dopočítávají bezpečná rychlost podle předcházejícího vozidla. Předpokládá se přitom, že řidič volí svoji rychlost tak, aby zajistil bezpečné zastavení nebo aby dokázal náhle rychle zastavit při nenadálé události.

První podmínka, kterou aplikujeme na vozidlo n, souvisí s rychlostí vozidla. Ta nepřekročí jeho preferovanou rychlost V_n a jeho zrychlení na volné komunikaci[4] se zvyšuje stejně tak jako rychlost dokud se také zvyšuje kroutící moment motoru vozidla a poté začne klesat na nulovou hodnotu a té nabude ve chvíli, kdy se vozidlo bude pohybovat svoji preferovanou rychlostí V_n

$$v_n(t+T_n) = v_n(t) + 2{,}5\, a_n\, T_n \left(\frac{1 - v_n(t)}{V_n}\right) \sqrt{\left(0{,}025 + \frac{v_n(t)}{V_n}\right)}. \tag{1.12}$$

[4] tj. komunikace, na které se nevyskytuje žádné další vozidlo

Koeficienty v nerovnici (1.12) byly stanoveny na základě z měření na hlavních dopravních komunikacích při průměrném provozu. Využití nerovnice (1.12) pro tento model je považované za přijatelné až do chvíle, kdy se vozidlo přiblíží vozidlu jedoucím před ním. Od té chvíle důraz na tuto podmínku klesá, naopak největší důležitost je ve chvíli, kdy vozidlo nemá před sebou žádné vozidlo či vozidlo před ním se nachází velmi daleko.

Další omezení, které je nutno zmínit, se týká brždění. Jestliže $n-1$ (první) vozidlo zahájí brždění v čase t, zpomalí a následně zastaví v místě x^*_{n-1}, které dostáváme rovnicí

$$x^*_{n-1} = x_{n-1}(t) - \frac{v_{n-1}(t)^2}{2b_{n-1}}, \tag{1.13}$$

kde člen b_{n-1} označuje nejvyšší kritické zpomalení ■ **decelarace?** ■, kterou je řidič vozidla $n-1$ schopen vykonat. Její hodnota je vždy záporná.

Vozidlo n jedoucí za ním nebude reagovat na toto zpomalení ihned, ale až v čase $t+T_n$ a tudíž zastaví až v místě x^*_n, daném rovnicí

$$x^*_n = x_n(t) + [v_n(t) + v_n(t+T_n)]\,\frac{T_n}{2} - \frac{v_n(t+T_n)^2}{2b_n}. \tag{1.14}$$

Pro vlastní bezpečnost musí řidič vozidla n zajistit, že bude splněna podmínka

$$x^*_{n-1} - s_{n-1} < x^*_n, \tag{1.15}$$

kde s_{n-1} je délka vozidla $n-1$, sečtena s bezpečnou vzdáleností následujícího (n-tého) vozidla. Pokud podmínka (1.15) neplatí, n-té vozidlo nemá žádný prostor pro případnou řidičovu chybu. Z toho důvodu obsahuje Gippsův model ještě další podmínku: bezpečný odstup s další časovou rezervou, θ. Řidič začne reagovat na vozidlo před sebou až v čase $t+T_n$. Od této chvíle tedy máme reakční čas T_n a bezpečný reakční čas $T_n+\theta$, který budeme ve výpočtech dále využívat. S využitím rovnic (1.13) a (1.14) můžeme podmínku (1.15) přepsat do nerovnice

$$\begin{aligned} x_{n-1}(t) - \frac{v_{n-1}(t)^2}{2b_{n-1}} - s_{n-1} \geq x_n(t) + \\ + [v_n(t) + v_n(t+T_n)]\,\frac{T_n}{2} + v_n(t+T_n)\theta - \frac{v_n(t+T_n)^2}{2b_n}. \end{aligned} \tag{1.16}$$

Bez parametru θ by řidič vozidla byl nucen brzdit až ve chvíli, kdy opravdu musí, to znamená na maximální výkon brzd. Do té chvíle by se stále pohyboval svoji preferovanou rychlostí. Parametr θ slouží k dřívějšímu a přitom ne tak prudkému brždění.

Při skutečném pozorování dopravy jsme schopni změřit všechny parametry vozidla n v rovnici (1.16) kromě členu b_{n-1}. Tento člen nahradíme odhadnutou hodnotou $\hat{b}$, upravíme tuto rovnici a dostáváme

$$\begin{aligned} -\frac{v_n(t+T_n)^2}{2b_n} + v_n(t+T_n)\left(\frac{T_n}{2}+\theta\right) - \\ - [x_{n-1}(t) - s_{n-1} - x_n(t)] + v_n(t)\frac{T_n}{2} + \frac{v_{n-1}(t)^2}{2\hat{b}} \leq 0. \end{aligned} \tag{1.17}$$

Relativní hodnoty T_n a θ jsou důležité v určování chování vozidel. Stejně jako v případě Newellova modelu 1.1, (1.6), pokud je hodnota θ rovna právě jedné polovině T_n, vozidlo jedoucí bezpečnou rychlostí bude schopno udržovat tento stav nekonečně dlouhou dobu. Díky tomu můžeme rovnici (1.17) přepsat do tvaru

$$-\frac{v_n(t+T_n)^2}{2b_n}+v_n(t+T_n)T_n-[x_{n-1}(t)-s_{n-1}-x_n(t)]+v_n(t)\frac{T_n}{2}+\frac{v_{n-1}(t)^2}{2\hat{b}}\leq 0. \quad (1.18)$$

Z toho plyne

$$v_n(t+T_n)\leq b_nT_n+\sqrt{\left(b_n^2T_n^2-b_n\left(2\left[x_{n-1}(t)-s_{n-1}-x_n(t)\right]-v_n(t)T_n-\frac{v_{n-1}(t)^2}{2\hat{b}}\right)\right)}. \quad (1.19)$$

Z nerovnice (1.18) vychází, že bezpečné rychlosti v_n budou ležet mezi dvěma kořeny této nerovnice, ale pokud bude mít spodní kořen zápornou hodnotu můžeme ho ignorovat, protože nás zajímají pouze kladné hodnoty rychlostí. Podcenění plynulého brždění řidiče vozidla n nastane ve chvíli, kdy

$$v_n(t+T_n)<v_n(t)+b_nT_n.$$

V tuto chvíli musí vozidlo začít brzdit rychleji, než by sám řidič chtěl. Řidič tedy volí rychlost na základě preferované decelarace, ale může brzdit i rychleji pokud by to bylo nutné.

Nerovnice (1.12) pro vozidlo n na volné komunikaci a (1.19) pro vozidlo n před kterým se nachází vozidlo $n-1$ představují dvě hlavní podmínky pro rychlost v_n v čase $t+T_n$ a jestliže řidič přizpůsobí rychlost parametrům vozidla a tak, aby jízda byla bezpečná, dostáváme **konečný vztah pro Gippsův model:**

$$v_n(t+T_n)=\min\left(v_n(t)+2,5a_nT_n\left(1-\frac{v_n(t)}{V_n}\right)\sqrt{\left(0,025+\frac{v_n(t)}{V_n}\right)},\right.$$
$$\left.b_nT_n+\sqrt{\left(b_n^2T_n^2-b_n\left(2\left[x_{n-1}(t)-s_{n-1}-x_n(t)\right]-v_n(t)T_n-\frac{v_{n-1}(t)^2}{2\hat{b}}\right)\right)}\right) \quad (1.20)$$

Pokud platí pro téměř všechna vozidla nižší hodnota v rovnici (1.20), vychází se z nerovnice (1.19), vozidla jedou blízko sebe, z čehož vyplývá vysoká hustota provozu. Pokud platí opak, tz. nerovnice (1.12), dopravní proud je volný.

1.2.2 Parametry Gippsova modelu

Všechny proměnné a parametry Gippsova modelu potom jsou:

- t - aktuální čas

- T_n - reakční čas, stejný pro všechna vozidla,
- $v_n(t)$ - rychlost vozidla n v čase t,
- a_n - maximální hodnota zrychlení, kterým je řidič ochoten zrychlovat - nikoliv zrychlení vozidla v čase n!,
- V_n - preferovaná rychlost vozidla n,
- b_n - maximální decelerace vozidla, kterým je řidič ochoten brzdit,
- x_n - poloha vozidla n
- s_{n-1} - délka vozidla $n-1$ sečtena s bezpečnou vzdáleností následujícího (n) vozidla
- $\hat{b}$ - odhadnutá hodnota decelerace b_{n-1}

1.3 Kraussův model

Pro simulaci v této práci budeme využívat software SUMO[5] a součástí tohoto programu je předpřipravený Kraussův model vytvořený v rámci disertační práce stejnojmeného autora. V popisu tohoto modelu budeme vycházet z této práce[11].

Tento model se zaměřuje přimárně na všeobecné vlastnosti dopravního proudu, zkoumání chování řidiče zde není primární. Myšlenka autora je taková, že existují obecné vlastnosti dopravního proudu, ze kterých vyplývá chování jednotlivých účastníků silničního provozu a není to chování jednotlivce, které hlavně ovlivňují vlastnosti dopravního proudu.

1.3.1 Obecný přístup

Pokud se dopravní proud modeluje mikroskopicky, musí být brány v úvahu dva typy pohybu vozidla. Prvním typem pohybu je vozidlo jedoucí po vozovce osamoceno, zatímco ve druhém případě vozidlo interaguje s ostatními vozidly. Na základě těchto pohybů předpokládáme následující podmínky. První podmínka má přímo souvislost s rychlostí vozidla. Ta je omezena nějakou maximální rychlostí $v_{\max}$,

$$v \leq v_{\max}. \tag{1.21}$$

Za maximální rychlost může být například zvolena preferovaná rychlost vozidla.

Hlavní důvod, proč mezi sebou vozidla interagují, je fakt, že řidiči nemají v úmyslu srážet se s ostatními vozidly. Další podmínkou tedy bude tzv. „nekoliznost“ systému. Budeme předpokládat, že řidič vozidla zvolí tedy takovou rychlost vozidla, která nebude vyšší než maximální bezpečná rychlost v_{safe},

$$v \leq v_{\text{safe}}. \tag{1.22}$$

[5] Simulation of Urban MObility

Z maximální bezpečné rychlosti poté plyne interakce mezi vozidly. Její stanovení bude uvedeno dále.

Je možné formulovat modely na základě podmínek (1.21) a (1.22). Nicméně je vcelku vhodné předpokládat také omezující podmínky pro zrychlení,

$$\begin{aligned} -b \leq \frac{dv}{dt} \leq a, \\ a, b > 0. \end{aligned} \tag{1.23}$$

Později bude ukázáno, že podmínka (1.23) je nezbytnou složkou, pokud chceme modelovat dopravní proud správně.

Předchozí podmínky můžeme shrnout do následující nerovnice

$$v(t + \Delta t) \leq \min(v_{\max}, v_{\text{safe}}, v(t) + a\Delta t), \tag{1.24}$$

kde se maximální bezpečná rychlost v_{safe} musí splňovat podmínku

$$v(t + \Delta t) \geq v(t) - b\Delta t. \tag{1.25}$$

Celkové povědomí o tom, jak se jednotlivá vozidla pohybují a interagují mezi sebou závisí na výpočtu maximální bezpečné rychlosti v_{safe}. Jakmile je stanovena hodnota v_{safe}, nerovnice (1.24) představuje aktualizované schéma pro simulace dopravního proudu, pokud jsou splněny všechny podmínky zmíněné výše.

1.3.2 Interakce mezi vozidly

Pro vytvoření sofistikovaného dopravního modelu je nutné ukázat, jak vozidla mezi sebou interagují. Podívejme se na dvě vozidla na komunikaci, první vozidlo se nachází v bodě x_1 a jede rychlostí v_1, vozidlo, které ho následuje je v bodě x_f a pohybuje se rychlostí v_f. Jestliže délka prvního vozidla bude l, tak mezera mezi vozidly se vypočte vztahem

$$g = x_1 - x_f - l. \tag{1.26}$$

Jak již bylo zmíněno, hlavní důvod, proč mezi sebou vozidla interagují, je fakt, že řidiči nemají v úmyslu srážet se s ostatními vozidly. Z toho plyne, že mezera mezi vozidly musí být nezáporná. Narozdíl od jiných modelovaných přístupů, zde nezačneme s předpokladem jak může být zrychlení vozidla vypočteno z vozidla jedoucí před ním, protože nekoliznost stejně není splněna automaticky a někdy je velice složité ji dokázat.

Místo toho začneme zavedením toho, že se v spojitých modelech spolu interagující vozidla nesrazí tehdy, pokud je mezera mezi vozidly g větší než nějaká preferovaná mezera mezi vozidly g_{des} a splňuje tuto dynamickou nerovnici

$$\frac{dg}{dt} \geq \frac{g_{\text{des}} - g}{\tau_{\text{des}}}. \tag{1.27}$$

Preferovaný bezpečný časový odstup mezi vozidly τ_{des} a preferovaná mezera g_{des} by mohly funkcemi odstupu mezi vozidly či jejich rychlostmi. Fakt, že v tomto modelu

nemohou nastat kolize je zřejmá, protože pokud položíme g rovné nule časová derivace bude vždy nezáporná a preferovaná mezera g_{des} také.

Předpokládejme případ dvou vozidel jedoucími rychlost v_1 resp. v_f s odstupem g. Druhé vozidlo jede svoji bezpečnou rychlostí a řidič tohoto vozidla může zastavit za jakýkoliv okolností tak, aby nedošlo ke kolizi s prvním vozidlem. To znamená, že pokud řidič má reakční čas τ a brzdná vzdálenost vozidel jedoucích rychlostí v je dána funkcí $f(v)$, je situace bezpečná právě tehdy když

$$f(v_f) + v_f\tau \leq f(v_1) + g. \tag{1.28}$$

Hodnota funkce $f(v)$ nemusí být nutně minimální možná brzdná vzdálenost, ale může to být jiná funkce závisející na způsobu řízení řidiče a pohodlné jízdy. To samé platí pro reakční čas τ.

Úprava nerovnice 1.28 vede k bezpečnostnímu pravidlu pro rychlost druhého vozidla. Nicméně my toto pravidlo neodvozujeme ze dvou důvodů. První z nich je ten, že po úpravě bychom potřebovali znát přesný předpis funkce $f(v)$, který jak uvidíme není možné zjistit. Zadruhé je zbytečné a výpočetně náročné používat typ řízení, které potřebuje předvídat nenádalou událost a vypočítávat decelaraci až k úplnému zastavení každý jednotlivý krok. Proto zavedeme Taylorův rozvoj brzdné vzdálenosti $f(v)$ okolo průměrné rychlosti obou vozidel,

$$\overline{v} = \frac{v_1 + v_f}{2}, \tag{1.29}$$

který budeme v dalším odvození dále používat. Sudé členy tohoto rozvoje se vyruší a zanedbáním členů vyšších řádů dostáváme

$$f'(\overline{v})v_f + v_f\tau \leq f'(\overline{v})v_1 + g. \tag{1.30}$$

Smysl derivace $f(\overline{v})$ vysvětlíme jednoduše. Pokud se podívame na decelaraci vozidla b z rychlosti v na nulu (nemusí být nutně konstantní), tedy $\dot{v} = -b(v)$, kdy $b(v) > 0$, obrdžíme vztah

$$f'(v) = -\frac{d}{dv}\int_v^0 \frac{v'}{b(v')}dv' = \frac{v}{b(v)} \tag{1.31}$$

Nyní můžeme přepsat podmínku bezpečnosti do tvaru

$$v_1 - v_f \geq \frac{v_1\tau - g}{\frac{\overline{v}}{b(\overline{v})} + \tau}, \tag{1.32}$$

kde

$$v_1 - v_f = \frac{dg}{dt}. \tag{1.33}$$

Nerovnice (1.32) se v podstatě rovná nerovnici (1.27), kde preferovaný odstup $g_{\text{des}} = v_1 t$ a preferovaný bezpečný časový odstup mezi vozidly $\tau_{\text{des}} = \tau_b + \tau$, kde $\tau_b = \overline{v}/b(\overline{v})$ je stanoveno na základě decelerací, které řidič využije.

1.3.3 Diskretizace modelu

Nerovnice v části 1.3.1 je potřeba převést na diskrétní rovnice pro dráhy, rychlosti a zrychlení pro možnost simulace na počítači.

Vhodná cesta pro vytvoření rovnic dynamického systému z bezpečnostní podmínky (1.27) je převést spojitou rychlost v_f ve výrazu $\dot{g}(t) = v_1(t) - v_f(t)$ jako rychlost diskrétní s diferencí či integračním krokem $t + \Delta t$. Potom rovnici (1.27) přepíšeme do tvaru

$$v_f(t + \Delta t) \leq v_1(t) + \frac{g(t) - g_{\text{des}}}{\tau_{\text{des}}}. \tag{1.34}$$

Rovnice pro polohu vozidla se potom na stejném principu převede na

$$x(t + \Delta t) = x(t) + v(t + \Delta t)\Delta t. \tag{1.35}$$

Víme, že při $\Delta t \to 0$ a $g_{\text{des}} \geq 0$ rovnice (1.34) a nerovnice (1.35) garantují bezpečnost. Pro konečný čas Δt musí být tato garance opět dokázána.

Mezera $g(t)$ mezi dvěma vozidly je diskretizována vztahem

$$g(t + \Delta t) = g(t) + \Delta t(v_1(t + \Delta t) - v_f(t + \Delta t)). \tag{1.36}$$

Přidáním rovnice (1.34) dostáváme

$$\xi(t + \Delta t) = \xi(t) + \left(1 - \frac{\Delta t}{\tau_b + \tau}\right) + \Delta t \frac{g_{\text{des}} - v_1(t)\Delta t}{\tau_b + \tau}, \tag{1.37}$$

kde

$$\xi(t) = g(t) - v_1(t)\Delta t.$$

Bezpečnost ($g \geq 0$) je tedy garantována tehdy, když platí:

$$\begin{aligned} &\xi(t = 0) \geq 0, \\ &\Delta t \leq \tau, \\ &g_{\text{des}} \geq v_1 \Delta t. \end{aligned} \tag{1.38}$$

To je výsledek, který se ovšem dal předpokládat. Záleží pouze na tom, zda je i po provedení iteračního kroku stále splněna podmínka bezpečnosti, tz. kontrolovat jestli je náš iterační krok menší než reakční čas řidiče τ.

1.3.4 Rovnice modelu

Předpokládáme tedy, nezávisle na náhodném kolísání, že každé vozidlo se pohybuje nejvyšší rychlostí na základě podmínek, které jsou stanoveny níže. **Kraussův model tedy stanovíme následujícími rovnicemi:**

$$\begin{aligned} &v_{\text{safe}}(t) = v_1(t) + \frac{g(t) - g_{\text{des}}}{\tau_b + \tau}, \\ &v_{\text{des}} = \min[v_{\max}, v(t) + a(t)\Delta t, v_{\text{safe}}(t)], \\ &v(t + \Delta t) = \max[0, v_{\text{des}} - \eta]. \end{aligned} \tag{1.39}$$

Preferovaná mezera g_{des} může být zvolena různě. Většinou se volí $g_{\text{des}} = v_1\tau$, kde τ je reakční čas řidiče. Hodnota τ_b je definována jako podíl $\overline{v}/b$, kde $\overline{v}$ plyne ze vztahu (1.29). Nově zde také zavádíme proměnnou η, která je náhodnou výchylkou od běžného řízení. Volba integračního kroku Δt a preferované mezery g_{des} je závislá na podmínkách (1.38).

1.4 Intelligent Driver Model

Intelligent Driver Model (IDM) je poměrně nový matematický model z roku 2000, který byl vymyšlen v Německu. Snaží se vylepšit některé nedostatky předcházejících modelů, ale i přispět novými myšlenkami v této oblasti.

V této sekci budeme primárně vycházet z práce [17] a [16], ve které je Intelligent Driver Model (IDM) do značné míry rozebrán. Využijeme hlavně podkapitolu 5.1 z [17], kterou doplníme o poznatky, které jsou uplatněny z kapitoly 1.

1.4.1 Matematický popis

Pro výpočet zrychlení v IDM je použit vztah ■ **bacha na to, jak značíte derivace podle času, mělo by to být všude stejně** ■

$$\dot{v}_n = a_{\text{free}} + a_{\text{int}}, \tag{1.40}$$

kde se aktuální zrychlení účastníka vypočítá jako součet zrychlení při volném proudu a_{free} a zrychlení, ve kterém je zahrnut i účastník jedoucí před ním a_{int}. Zde je vidět určitá paralela s Gippsovým modelem (kapitola 1.2, kde se ovšem vybírá pouze jedna z těchto dvou hodnot. Navíc i rovnice se od IDM odlišují. Pro volný proud je tedy zrychlení $a_{\text{int}} = 0$.

Zrychlení při volném proudu je definováno vztahem

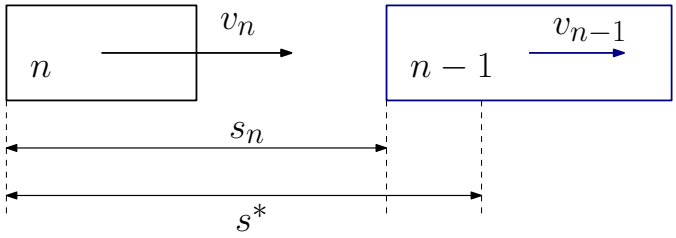
$$a_{\text{free}} = a\left[1 - \left(\frac{v_n}{v_{n\text{free}}}\right)^{\delta}\right], \tag{1.41}$$

kde a je maximální zrychlení, δ je exponent zrychlení, v_n je aktuální rychlost vozidla a $v_{n\text{free}}$ je rychlost, které chce vozidlo dosáhnout.

Pokud v rovnici (1.41) bude zlomek v_n/v_{free} roven jedné, auto se již dostalo na svou preferovanou rychlost v_{free} a celkové zrychlení a_{free} bude rovno nule. Naopak pokud bude v_n rovno nule, znamená to, že a_{free} bude přímo rovno a, a auto pojede svým maximálním zrychlením. Pokud v_n bude menší než v_{free}, pak platí že, čím vyšší hodnotu bude mít nastavenou parametr δ, tím se dosáhne vyšší hodnota a_{free}. Pokud v_n bude větší než v_{free}, bude platit pravý opak.

Auto jedoucí při volném proudu je ale nereálná představa, proto se do rovnice (1.40) přidává interakční člen, který přímo závisí na parametrech vozidla jedoucí před naším účastníkem. Tento člen je roven

$$a_{\text{int}} = -a\left(\frac{s_n^*}{s_n}\right)^2, \tag{1.42}$$



Obrázek 1.4: Proměnné jednotlivých aut

kde a je maximální zrychlení, $s_n = x_{n-1} - x_n$ je rozdíl poloh vozidel a s_n^*

$$s_n^* = s_0 + v_n T + \frac{v_n(v_n - v_{n-1})}{2\sqrt{ab}} \tag{1.43}$$

je preferovaný odstup. Hodnota s_0 se vypočte jako součet minimální bezpečné vzdálenosti mezi vozidly a délkou následujícího vozidla, T je bezpečnostní časový odstup mezi vozidly a b je konstanta brzdění. Situace je schématicky znázorněna na obrázku 1.4.

Člen úměrný $v_n - v_{n-1}$ je kvadratický v proměnné v_n a obsahuje součin $v_n v_{n-1}$. Tyto nelineární členy popisují interakci, v tomto případě dvou vozidel. Pokud tedy bude vzdálenost s_n mezi auty vysoká, potom zlomek s_n^*/s_n bude mít velmi malou hodnotu a interakční člen a_{int} lze zanedbat, takže vozidlo se pohybuje jako ve volném proudu.

Pokud rychlost auta v_n bude vyšší než rychlost auta v_{n-1}, pak interakční člen s_n^* bude kladný a požadovaný odstup se zvětší, takže zrychlení auta $\dot{v}_n$ se musí snížit.

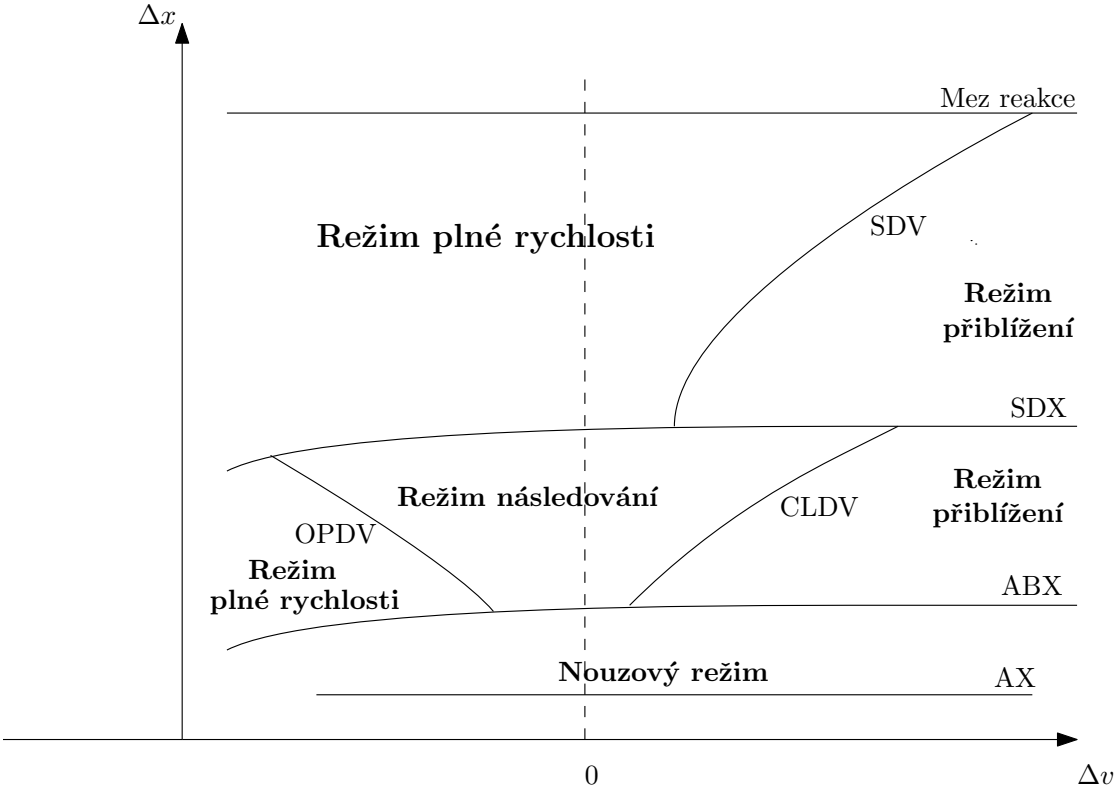
1.5 Wiedemannův model

V programu SUMO je také připraven Wiedemannův model, jehož první verze vznikla již v roce 1974 jako disertační práce tehdejšího studenta R. Wiedemanna. Tento model byl posléze několikrát vylepšován, v roce 1992 o tomto modelu vyšel aktualizovaný článek. Tyto práce ovšem bohužel nejsou k dispozici a SUMO (viz dále) navíc ani neobsahuje přímo Wiedemannův model, nýbrž jeho poupravenou verzi. V této sekci budeme hlavně vycházet z článku [7]. V něm najdeme poměrně přesný popis Wiedemannova modelu[6], který je odlišný s verzí modelu v SUMO pouze v drobnostech. Na ně bude upozorněno v kapitole ■ **doplnit číslo až bude** ■ o simulacích.

Tento model je odlišný oproti předcházejícím. Pracuje totiž v tzv. režimech. Nejprve je nutné zjistit, v jakém režimu se vozidlo vlastně nachází a podle toho se určuje výpočet akcelerace či decelarace, která je pro každý režim jiná.

Klíčové pro pochopení Wiedemannova modelu je obrázek 1.5. Osa x zobrazuje rozdíl rychlostí vozidel Δv (aktuálního a následujícího) a na ose y se nachází vzdálenost mezi těmito vozidly Δx. Znamená to, že pokud rozdíl rychlostí a zároveň rozdíl poloh vozidel dosahuje určitých hodnot, vždy náleží jednomu konkrétnímu režimu, podle kterého je

[6] Tento článek se přímo odkazuje na článek z roku 1992



Obrázek 1.5: Režimy Wiedemannova modelu

vozidlo dále zpracováváno. K tomu abychom mohli určit příslušný režim vozidla, musíme si určit prahové hodnoty křivek AX, ABX, SDX, CLDV, OPDV a SDV.

Preferovaný odstup mezi stacionárními[7] vozidly značíme AX. Tento práh sestává s délky čelního vozidla sečtenou s preferovanou vzdáleností mezi vozidly. Je definován vztahem

$$AX = L_{n-1} + AXadd + RND1_n \cdot AXmult, \tag{1.44}$$

kde $AXadd$ a $AXmult$ jsou parametry modelu a $RND1_n$ značí parametr závislý na řidiči generovaný z normálního rozdělení.

ABX je preferovaná maximální vzdálenost mezi vozidly při malých rozdílech. Ten se vypočítá následovně:

$$ABX = AX + BX, \tag{1.45}$$

kde

$$BX = (BXadd + BXmult \cdot RND1_n) \cdot \sqrt{v}, \tag{1.46}$$

[7] nezávislé na čase

kde $BXadd$ a $BXmult$ jsou opět parametry modelu. Rychlost v určujeme v závislosti na základě následujícího vozidla,

$$v = \begin{cases} v_{n-1} & \text{pro} \quad v_n > v_{n-1}, \\ v_n & \text{pro} \quad v_n \leq v_{n-1}. \end{cases} \tag{1.47}$$

Maximální vzdálenost při které bere vozidlo na zřetel vozidlo před sebou se značí SDX. SDX se pohybuje mezi 1,5 až 2,5 násoobkem hodnoty ABX a je definována následovně

$$SDX = AX + EX \cdot BX, \tag{1.48}$$

kde EX stejně jako u ostatní členů je roven

$$EX = EXadd + EXmult \cdot (NRND - RND2_n). \tag{1.49}$$

Parametry $EXadd$ a $EXmult$ jsou opět parametry pro zpřesnění výpočtů, hodnota $NRND$ je generována z normálního rozdělení a $RND2_n$ značí parametr závislý na řidiči generovaný také z normálního rozdělení.

Další důležitou křivku určující práh, kdy řidič zjistí, že se přibližuje pomalejšímu vozidlu před sebou, značíme SDV. Vypočítá se vztahem:

$$SDV = \left(\frac{\Delta x - L_{n-1} - AX}{CX} \right)^2, \tag{1.50}$$

kde CX zavedeme jako

$$CX = CXconst \cdot (CXadd + CXmult \cdot (RND1_n + RND2_n)), \tag{1.51}$$

Parametry $CXconst$, $CXadd$ a $CXmult$ jsou zde opět pro zpřesnění výpočtů při určování režimu.

Pokud se jedná o klesající rozdíl rychlostí CLDV, v [7] je uvedeno, že v článku z roku 1992 od Wiedemanna a Reitera je stejná jako křivka SDV.

Při zvyšujícím se rozdílu rychlostí, OPDV, řidič zjišťuje, že jede rychlostí nižší než vozidlo před ním. Práh OPDV se vypočítá jako součin SDV a parametrů modelu $OPDVadd$, $OPDVmult$ a $NRND$, což je náhodné číslo z normálního rozdělení,

$$\begin{aligned} OPDV = CLDV \cdot (-OPDVadd - OPDVmult \cdot NRND) = \\ SDV \cdot (-OPDVadd - OPDVmult \cdot NRND). \end{aligned} \tag{1.52}$$

Na základě těchto prahů stanovujeme příslušný režim Wiedemannova modelu. Tyto režimy jsou taktéž zobrazeny na obrázku 1.5.

Režim následování: Hodnota zrychlení vozidla je vždy počítána každou iteraci znovu od nuly. Důvodem toho je nepřesné zacházení s plynovým pedálem ve vozidle. Pokud vozidlo překoná práh SDV či ABX, jeho hodnota akcelerace bude rovna $-b_{null}$. Naopak při dosáhnutí prahů $OPDV$ či SDX bereme kladnou hodnotu b_{null}. Akcelerace či decelerace je definována jako

$$b_{null} = BNULLmult \cdot (RND4_n + NRND), \tag{1.53}$$

kde $BNULLmult$ je parametr modelu, hodnota $NRND$ je generována z normálního rozdělení a $RND4_n$ značí parametr závislý na řidiči generovaný také z normálního rozdělení.

Režim plné rychlosti: Vozidlo je umístěno nad všemi prahy v obrázku 1.5 a jede nezávisle na okolní dopravě. Vozidlo jede s maximálním zrychlením tak, aby dosáhlo své preferované rychlosti. Ve chvíli, kdy je dosažena preferovaná rychlost. Nepřesné zacházení s plynem ve vozidle je modelováno přiřazením akcelerace b_{null} či $-b_{null}$. Maximální zrychlení je definováno jako

$$b_{\max} = BMAXmult \cdot (v_{\max} - v \cdot FaktorV), \tag{1.54}$$

kde $FaktorV$ zavedeme jako

$$FaktorV = \frac{v_{\max}}{v_{des} + FAKTORVmult \cdot (v_{\max} - v_{des})}, \tag{1.55}$$

kde $v_{\max}$ je maximální rychlost vozidla a FAKTORVmult parametr modelu.

Režim přiblížení: Pokud přesáhne vozidlo práh SDV, řidič zjišťuje, že se přibližuje pomalejšímu vozidlu. Řidič tedy začne zpomalovat, aby zabránil kolizi. Decelerace se vypočítá vztahem

$$b_n = \frac{1}{2}\frac{(\Delta v)^2}{ABX - (\Delta x - L_{n-1})} + b_{n-1}, \tag{1.56}$$

kde b_{n-1} je decelerace následujícího vozidla.

Nouzový režim Jestliže vzdálenost čela aktuálního vozidla s nárazníkem vozidla před aktuálním je menší než ABX, decelerace se vypočítá následovně

$$b_n = \frac{1}{2}\frac{(\Delta v)^2}{AX - (\Delta x - L_{n-1})} + b_{n-1} + b_{\min}\frac{ABX - (\Delta x - L_{n-1})}{BX}, \tag{1.57}$$

kde $b_{\min}$ je maximální decelarace vozidla a zavedeme ji jako

$$b_{\min} = -BMINadd - BMINmult \cdot RND3_n + BMINmult \cdot v_n, \tag{1.58}$$

kde $BMINadd$ a $BMINmult$ jsou parametry modelu a $RND3_n$ značí parametr závislý na řidiči generovaný z normálního rozdělení.

1.6 Shrnutí

Časově spojité mikroskopické modely bez předjíždění jsou v zásadě definovány funkcí zrychlení. V nejstarších modelech může zrychlení $\dot{v}(t+T_r)$ vozidla α zapsáno jako

$$\dot{v}(t+T_r) = \frac{-\lambda v_\alpha^m \Delta v_\alpha}{s_\alpha^l}, \tag{1.59}$$

kde λ, l a m jsou parametry modelu, s je odstup mezi vozidly a Δv_α rozdíl rychlostí.

Kromě toho, může zrychlení může také záviset na vlastní rychlosti vozidla v_α a klesá se vzdáleností s_α k vozidlu před ním,

$$s_\alpha = x_{\alpha-1} - x_\alpha - l_\alpha, \tag{1.60}$$

kde l_α je délka vozidla α.

V závislosti na rovnici (1.59), kdy zrychlení závisí na vedoucím vozidle, tyto modely není možné použít při velmi nízkých hustotách provozu. Pokud není určeno vedoucí vozidlo ($s_\alpha \to \infty$), zrychlení buďto není možné stanovit (pro hodnotu $l = 0$) nebo je rovné nule ($l > 0$) bez ohledu na rychlost vozidla. Dá se předpokládat, že v tuto chvíli se vozidla budou snažit dosáhnout své preferované rychlosti. Chování vozidel v hustém provozu je ovšem poněkud nerealistické. Především odstup mezi vozidly s týkající se vedoucího vozidla nemusí nutně směřovat k rovnovážnému stavu. Ani malé mezery nepřimějí vozidlo k brždění pokud je rozdíl rychlostí Δv_α roven nule.

Tyto problémy byly vyřešeny v Newellově modelu (viz. odstavec 1.1), kde je z rovnice (1.10) vidět, že zrychlení není závislé pouze na rozdílů rychlostí. Newellův model je nekolizní, ale okamžitá závislost na rychlosti na hustotě provozu vede k velmi vysokým hodnotám zrychlení.[15]

Kromě Newellova modelu a dalších modelů pro základní výzkum existují také vysoce komplexní modely jako Wiedemannův (viz. odstavec 1.5). Ten se snaží napodobovat dopravu jak nejlépe to jde ovšem za cenu mnoha parametrů modelu.

Další model, který obsahuje realistické brzdné reakce řidičů, je Gippsův model 1.2 (brzdný koeficient b_n) a Kraussův model 1.3 (brzdný koeficient b). Navzdory své jednoduchosti, tyto modely ukazují realistické chování řidiče a jsou nekolizní. Toto chování se bohužel ztrácí ve chvíli, kdy modelujeme dopravu deterministicky[8]. Tím pádem není možné simulovat nestabilní stavy dopravy.

[8] např. když vozidla vjíždí do určité oblasti každých 5 sekund

2. Simulační a podpůrný software

učesat, je to zmatené

Po prostudování modelů byla otázka dalšího postupu. Bylo možné pokračovat v aplikaci, kterou autor vytvořil pro svoji bakalářskou práci. další variantou bylo využít softwaru AIMSUN, který by byl také vhodný pro naprogramování vlastních modelů. Nicméně zvítězil open-source simulační software SUMO. Tento produkt se neustále vyvíjí již od roku 2001 a jeho podpora neustále pokračuje. **tato věta je opravdu bla bla bla** Nejprve se autor pokusil začít pracovat v AIMSUNu, nicméně to skončilo neúspěšně, protože pro naprogramování dalších modelů do tohoto software byl potřeba balíček, který nebyl zdarma.

Další variantou bylo pokračovat v programovacím jazyku JAVA na vytvořené aplikaci pro bakalářskou práci, nicméně po prozkoumání dalších variant se autor rozhodl, že bude výhodnější pokračovat s již vyvíjeným softwarem SUMO.

V této kapitole rozebereme simulační software SUMO[1] a další, ať již naprogramované autorem či volně dostupné nástroje.

2.1 SUMO

SUMO je open-source nástroj pro dopravní simulace. Ve výzkumu se dá SUMO využít v mnoha oblastech např. vyhledání nejkratší cesty[2], možnost zkoušení signálních plánů nebo simulace komunikace mezi vozidly. SUMO bylo a je využíváno na mnoha zajímavých projektech[3].

Od roku 2001 se SUMO postupně rozšiřovalo a nyní je z něj již soubor několika programů. Umožňuje na základě dopravních měření, matic vzdáleností využívat různé routovací algoritmy, simulování křižovatek. Obsahuje také rozhraní TraCI pro online simulaci.

Při vývoji SUMO byly hlavní dva důvody proč by měl být opensource. První z nich je snaha podporovat komunitu, která se zabývá simulacemi, volně dostupným nástrojem ve kterém již budou zahrnuty základní algoritmy. Existují některé další volně dostupné simulační softwary, na kterých ovšem nepracovali studenti a přestaly být dříve či později podporovány. Druhým důvodem je neexistující porovnatelnost implementovatelnost modelů a algoritmů.

Tento druhý důvod a také možnost vlastní implementace modelů vedly autora k volbě SUMO. K tomu, aby bylo možné vlastní modely vůbec implementovat, bylo nejprve nutné vlastní SUMO zkompilovat, to znamená, že nestačí pouze verze SUMO, které obsahuje spustitelné soubory, ale přímo zdrojové kódy. Více o tom jak SUMO zkompilovat se čtenář dozví v příloze A.

[1] Simulation of Urban MObility

[2] či podle jiného kritéria

[3] např. předpověď dopravních proudů v době MS v Německu v roce 2006 v Kolíně nad Rýnem, využití komunitou V2X pro zjišťování dráhy vozidla atd.

2.1.1 Jak to funguje

Nyní již víme, čeho všeho je SUMO přibližně schopno. K tomu abychom spuštěna simulace je zapotřebí mít připraveny minimálně tři typy souborů, a to

- konfigurační soubor,
- soubor s podkladem (dopravní síť),
- soubor s vozidly.
- další soubory

2.1.2 Konfigurační soubor

V konfiguračním souboru je možné nastavit všechny možnosti, které obsahuje SUMO jako parametry příkazové řádky. To se může hodit obzvlášť ve chvíli, kdy se SUMO volá s mnoha parametry. Tehdy ho stačí zavolat pouze s konfiguračním souborem.

2.1.3 Soubor s podkladem

Pro diplomovou práci dostal autor data konkrétně průměrné intenzity a rychlosti z mýtných bran Pražského okruhu. Bylo proto vhodné mít jako podklad pro simulaci síť v konkrétních úsecích mezi mýtnými branami. Pro mapu oblasti bylo využito volně dostupných map ze serveru OpenStreetMap, kde se konkrétní úsek stáhnul ve formátu OSM a vymazaly se všechny prvky nesouvisející s Pražským okruhem. Posléze byl použit jeden z nástrojů SUMO a to konkrétně netconvert, který převádí data z předem definovaného formátu do SUMO formátu. Více k tomuto tématu v příloze B.

SUMO také umožňuje využít vlastní nástroj na generování sítí, JTRrouter či DFrouter.

2.1.4 Vozidla pro simulaci

Vozidla mají v SUMO několik parametrů, které mimo jiné závisí také na volbě mikroskopického modelu. Mezi základní patří

- id vozidla,
- typ vozidla,
- dráha, po které se bude vozidlo pohybovat,
- čas vjezdu do oblasti,
- místo vjezdu do oblasti,
- rychlost při vjezdu do oblasti,
- jízdní pruh při vjezdu do oblasti.

Můžeme na začátku definovat typ vozidla, což znamená jeho parametry jako např. délka vozidla, zrychlení, max. rychlost atp. Poté u každého vozidla nastavíme tento typ a vozidlo automaticky dostane všechny přiřazené parametry.

Dráha, po které se bude vozidlo pohybovat, souvisí s odstavcem 2.1.3, kde přímo tyto jízdní úseky vytváříme. Tento parametr může obsahovat více hodnot, v případě Pražského okruhu minimálně část mezi dvě mýtnými branami.

2.1.5 Další soubory

Další soubor, který vytvoříme, se týká informací o indukčních smyčkách. Definujeme zde její polohu a také informace, které má ukládat. Ty se dají nastavit stejně jako z dat, které máme k dispozici, tz. počet vozidel za minutu a průměrná rychlost vozidel za minutu.

2.2 Příprava a vyhodnocení simulace - program

Aby bylo možné dostat se k jádru celé práce, je nutná příprava dat pro simulaci a také jejich následné vyhodnocení. K tomuto účelu byla vytvořena konzolová aplikace jménem *AllInOne.jar*. Pro její správnou funkčnost je nutné mít nainstalované rozhraní pro Java aplikace (JRE), která umožňuje 4 základní operace,

1. převedení formátu dat (MATLAB) z Pražského okruhu na formát podporovaný softwarem SUMO (XML),
2. vytvoření souboru vozidel pro simulaci (viz podkapitola 2.1.4),
3. vytvoření výstupů ze simulace (viz dále)

Tato aplikace byla vytvořena v jazyku JAVA z důvodu nalezení balíčku pro možnost práce s daty ze softwaru MATLAB. Dále se autor rozhodl, že vytvoří raději jednu přehlednou aplikaci než 5 malých.

2.2.1 Třída AllInOne

V hlavní třídě se nachází konstruktor, který na základě vstupních parametrů zavolá příslušnou třídu, případně vytiskne chybovou hlášku s problémem. Dále je zde funkce na zjištění, zda zadaný parametr je číslo.

- *public AllInOne(String args[])* - konstruktor, větvení na základě vstupního pole *args*
- *public static boolean isInteger(String s)* - funkce na kontrolu, zda je vstupní řetězec *s* číslo

2.2.2 Převedení formátu dat - třída MatlabToXML

Pro SUMO je možné vytvořit si jakýkoliv soubor vozidel. SUMO obsahuje přímo manuál, jak tento soubor vytvořit. Není ovšem vhodné vytvářet ho ručně, nýbrž použít nějaký nástroj pro XML.

Třída, která toto umí v Javě, je nazvána *MatlabToXML*. V konstruktoru se vytvoří soubor, jehož název je jeden ze vstupních parametrů a poté ve funkci *createLoopDocument* se přímo vytváří jeho obsah.

- *public MatlabToXML(String fileNameMatlab, String fileNameCars)* - první vstupní parametr určuje soubor ve formátu .mat, který má předem definovanou strukturu, která je shodná s mýtnými branami z Pražského okruhu. Výstupní soubory se potom budou jmenovat na základě vstupní proměnné *fileNameCars*, kde dále bude následovat potržítko a číslo pruhu[4], ve kterém budou informace o rychlostech a intenzitách pouze pro ten daný pruh.
- *public Document createLoopDocument(int lane)* - vstupní parametr *lane* určuje pruh, pro který se právě obsah souboru vytváří.

Výstupní soubor bude shodný s výstupním souborem po simulaci SUMO, takže bude možné přesně jednotlivé hodnoty z mýtných bran porovnat s nasimulovanými.

Příkaz, kterým lze program spustit aby vytvořil tento dokument, je následující

```
1 java -jar AllInOne.jar -m sokp-0187-20121031.mat det0187.xml
```

Ze souboru *sokp-0187-20121031.mat* vytvoří v závislosti na počtu pruhů pod touto mýtnou branou stejný počet souborů, tedy *det0187_0.xml*, *det0187_1.xml* atd.

Tyto soubory lze následně použít jako vstup pro vytvoření souboru s vozidly pro SUMO a nebo pro vytvoření grafů nebo histogramů pro porovnání výsledků se simulací.

2.2.3 Vytvoření souboru vozidel pro simulaci - třída XMLCreator

Další funkčností, kterou aplikace disponuje, je vytvoření vozidel pro simulaci. Tento soubor je plně kompatibilní se vstupními soubory pro SUMO. Vstupním souborem pro generování vozidel bude opět soubor s příponou .mat, výstupním soubourem potom jeden soubor obsahující za sebou seřazená vozidla dle vjezdu do oblasti s rychlostí vygenerovanou z Gaussova rozdělení. Další parametry jsou předem dané či generované v aplikaci a určují se na základě volby modelu, který lze nastavit z příkazové řádky.

Jediný problém nastává ve chvíli, kdy musíme jednotlivým vozidlům nastavit jejich cestu v simulaci. Pokud všechna vozidla pojedou po stejné dráze, stačí, když se každému vozidlu přiřadí cesta, která je shodná s názvy hran podkladu (viz. podkapitola 2.1.3 a B.1 poslední odstavec).

[4] Např. pro vstupní parametr detektorkm0201.xml, budou výstupní soubory detektorkm0201_0.xml a detektorkm0201_1.xml

Pro naší simulaci cestu pro vozidlo předem připravíme tak, aby všechna vozidla projela úsekem, který chceme simulovat. V konzolové aplikaci v tomto směru není nutné nic dalšího nastavovat.

Třída XMLCreator obsahuje následující konstruktor a metody

- *public XMLCreator(String fileNameMatlab, String fileNameCars, String model, String type)* - konstruktor, který přijímá název souboru, ze kterých budeme vozidla generovat, název výstupního souboru a způsob generování vozidel,
- *public Document createRouteEqualDocument(String model)* - naplnění souboru při konstantní časové vzdálenosti vozidel,
- *public Document createRouteExpDocument(String model)* - naplnění souboru při generování vozidel z exponenciálního rozdělení,
- *public Element createVType(Element element, int actualCar, String model, int iTime, int lane)* - vytvoření parametrů pro konkrétní vozidlo, v závislosti na modelu
- *public Element createVehicle(Element element, int actualCar, double time, int iTime, int lane, String type)* - vytvoření jednoho exempláře vozidla s příslušnými parametry, *actualCar* je pořadí vozidla od začátku simulace, *lane* pruh do kterého bude vozidlo vygenerováno, *time* je čas ve kterém vozidlo bude vpuštěno do simulace.

Lze přidat konkrétní model, třetí parametr příkazu, podle kterého chceme aby SUMO simulovalo provoz. Tento parametr je nepovinný a pokud ho nezadáme, program předpokládá volbu Kraussova modelu. Zatím jsou k dispozici 4 modely a k nim příslušné parametry:

- Gippsův model 1.2
  - desiredSpeed (V_n) - preferovaná rychlost daného vozidla, rychlost je generovaná z normálního rozdělení se střední hodnotou 130 a směrodatnou odchylkou **Doplnit až bude hotovo**

```
1       java -jar AllInOne.jar -c sokp-0201-20121001.mat vozidla.rou.xml Gipps
```

- Kraussův model 1.3 - nemá žádné speciální parametry kromě základních

```
1       java -jar AllInOne.jar -c sokp-0201-20121001.mat vozidla.rou.xml Krauss
```

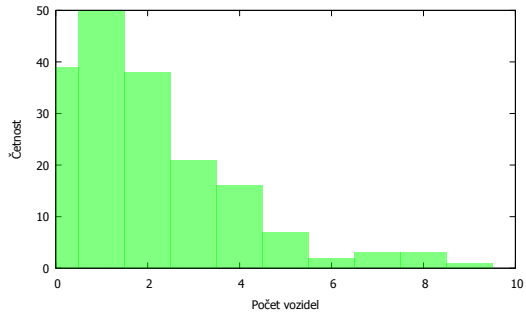
- IDM - Intelligent Driver Model 1.4
  - delta (δ) - exponent akcelerace (rovnice (1.41)) - hodnota 4
  - stepping - počet iterací při výpočtu rychlosti - hodnota 1

```
java -jar AllInOne.jar -c sokp-0201-20121001.mat vozidla.rou.xml IDM
```

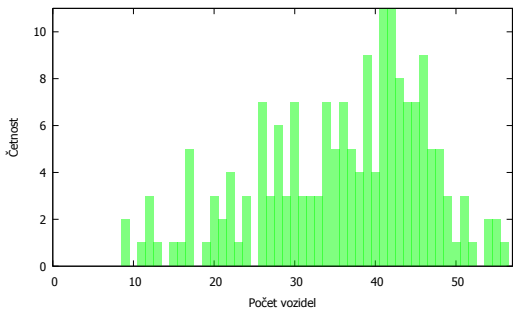
- Wiedemannův model ■ **Doplnit až bude hotovo teoreticky** ■

Výsledný soubor bude mít název vozidla.rou.xml - .rou je povinná druhá přípona, která pokud není uvedena, SUMO nespustí simulaci.

Výchozí generování vozidel je deterministické. Pokud nám v jedné minutě projede např. 10 vozidel, tak vozidla budou mít vstupní čas rovnoměrně rozdělen po 1/10. Tato varianta se zde nachází proto, aby bylo možné zkontrolovat správné generování vozidel, či případné chování jednotlivých modelů.



Obrázek 2.1: Četnosti pro čas od 2:00-5:00



Obrázek 2.2: Četnosti pro čas od 6:00-9:00

Další varianta využívá znalost, že pokud rozdělení počtu vozidel za nějaký čas je Poissonovo, potom odstupy mezi vozidly mají exponenciální rozdělení. Z pohledu na obrázek 2.1 a 2.2[5] lze předpokládat, že se opravdu jedná o Poissonovo rozdělení.

Pro variantu generování souboru vozidel deterministicky si můžeme nechat poslední parametr prázdný, výsledný příkaz tedy bude:

```
java -jar AllInOne.jar -c sokp-0201-20121001.mat vozidla.rou.xml Gipps
```

Pokud chceme, aby se nám vozidla generovala z exponenciálního rozdělení, kde se každou minutu bude měnit jeho parametr λ (v závislosti na počtu vozidel za minutu), zvolíme možnost spuštění s parametrem exp:

```
java -jar AllInOne.jar -c sokp-0201-20121001.mat vozidla.rou.xml Krauss exp
```

Je možné vynechat jeden či dva poslední parametry. V tu chvilí budou vozidla do simulace pouštěna s deterministickým časovým odstupem a budou se pohybovat na základě Kraussova modelu. Je možné také vynechat jeden z těchto dvou parametrů, tz.

```
java -jar AllInOne.jar -c sokp-0201-20121001.mat vozidla.rou.xml Gipps
```

generuje vozidla, která se budou pohybovat dle Gippsova modelu a budou mít deterministický časový odstup. Je možné vynechat i model, tz.

[5] vytvořeny také tímto programem

```
java -jar AllInOne.jar -c sokp-0201-20121001.mat vozidla.rou.xml exp
```

generuje vozidla z exponenciálního rozdělení. Pohyb bude na základě Kraussova modelu.

Výstupní soubor lze rovnou použít jako vstupní soubor vozidel pro simulaci v SUMO.

2.2.4 Výstupy ze simulací

Po simulaci SUMO vytvoří několik souborů. Hlavní soubor XML obsahuje informace o vozidlech, které se nacházejí v simulaci každou sekundu. Pro naše výstupy budou nejdůležitější soubory, které vzniknou pomocí dalšího souboru (viz 2.1.5). Tento soubor se bude zpracovávat stejně jako soubor vzniklý pomocí nástroje uvedeného v 2.2.2.

Pro porovnání reálných a nasimulovaných dat program nabízí tři možnosti vyhodnocení:

- časová řada počtu vozidel každou minutu,
- počet vozidel za časový úsek, jehož délku je možné nastavit,
- histogram četností vozidel. Zde je možnost nastavení časového intervalu, ve kterém nás četnosti zajímají.

2.2.5 Časová řada - třída XMLReader

Jednou z možností, jak zobrazit výstupy ze souborů, je pomocí časové řady. Program vyhodnotí všechny vstupní soubory a předpřipraví soubor *gnuplot.txt* pomocí kterého je potom možné vykreslit tuto řadu. Program vytvoří tento soubor pro každý pruh zvlášť a poté souhrný soubor, který obsahuje každý pruh a součet vozidel v jednom grafu.

Třída XMLReader obsahuje následující konstruktor a metody

- *public XMLReader(String fileNameInput, String fileNameOutput) throws DocumentException* - první parametr je vstupní soubor ve formátu XML (buďto vytvořený programem z Matlab souboru nebo ze simulace SUMO), druhý parametr je potom název souboru, který po spuštění gnuplot vygeneruje. Po spuštění našeho programu je tedy nutné spustit ještě program gnuplot.
- *public static void timeSeries(String fileNameInput,String fileNameOutput,int numberOfFiles) throws DocumentException* - stejné parametry jako u konstruktoru + navíc počet souborů, které je nutné zpracovat
- *private static Calendar setStart(Calendar cal)* - nastavení časové řady od půlnoci do půlnoci
- *public static Document parse(String fileName) throws DocumentException* - rozdělení XML dokumentu na jednotlivé části

Je potřeba si uvědomit, že zadáváme jeden vstupní soubor, ale vstup je vlastně několik souborů zároveň. Např. příkaz

```
1 java -jar AllInOne.jar -x detektor.xml graf.pdf
```

prochází aktuální složku a kontroluje všechny XML soubory mající tvar *detektor_0.xml*, *detektor_1.xml* atd. Poté vytvoří pomocné soubory, které obsahují sadu příkazů pro vykreslení grafu gnuplotem. Pomocné textové soubory jsou vždy pro každý pruh dva, a to *pocetvozidelpruh_1.txt* a *gnuplot_1.txt* pro 1. pruh atd. Hlavní soubor, kde jsou obsaženy všechny pruhy, má název *gnuplot.txt* a ke svému spuštění potřebuje *pocetvozidelpruh.txt*.

2.2.6 Počet vozidel za daný čas - třída XMLReaderHist

Další variantou je vykreslení počtu vozidel za čas. Je možné vykreslit počet vozidel za minutu, kde výsledek bude stejný jako časová řada s tím rozdílem, že grafem bude histogram. Program tudíž umí nastavit i jinou časovou délku úseku - např. počet vozidel za hodinu.

Třída XMLReaderHist obsahuje následující konstruktor a metody

- *public XMLReaderHist(String fileNameInput, String fileNameOutput, String w) throws DocumentException* - stejná situace jako u XMLReaderu (viz 2.2.5), poslední parametr je potom délka úseku, za který chceme počítat vozidla
- *public static void hist(String fileNameInput, String fileNameOutput, int numberOfFiles, String w) throws DocumentException* - úplně stejná funkce jako u XMLReaderu doplněna o délku časového úseku

Všechny ostatní funkce, které třída obsahuje, již byly vysvětleny v předcházejících třídách.

Pokud chceme histogram počtu vozidel během celého dne, necháme poslední parametr volání prázdný, tedy

```
1 java -jar AllInOne.jar -xh detektor.xml histogram.pdf
```

Další možností je nastavit si délku časového úseku. Tato délka musí být dělitelná počtem minut dne (1440) a udává se v minutách. Jinak program argument ignoruje a vytvoří histogram, jako kdyby žádný parametr nebyl zadán. Příkaz

```
1 java -jar AllInOne.jar -xh detektor.xml histogram.pdf 60
```

tedy vytvoří histogram, kde se sčítají vozidla za hodinu.

2.2.7 Histogram - třída Histogram

Poslední možností analýzy výstupu je histogram četností počtu vozidel.[6] Tento histogram můžeme udělat jak pro celý den, tak i pro určitou časovou oblast (např. ranní špička).

[6] Obrázky 2.1 a 2.2 vznikly právě díky tomuto modulu.

Pokud nejsou zadány parametry od a do, histogram se vytvoří pro celý den. Je možné také vytvořit histogram přes půlnoc - tz. od 22:00 do 3:00.

Třída Histogram obsahuje následující konstruktor a metody:

- *public Histogram(String fileNameInput, String fileNameOutput, String from, String to) throws DocumentException* - opět stejné parametry jako v předchozích třídách, tz. název vstupního a výstupního souboru, který bude vytvořen po spuštění gnuploutu, a dále časové hodnoty od kdy a do kdy chceme histogram vykreslit,
- *public static void hist(String fileNameInput, String fileNameOutput, int numberOfFiles, String from, String to) throws DocumentException* - vytvoření souborů pro gnuplot, všechny parametry jsou stejné jako u předchozích tříd.

Pokud chceme vytvořit histogram počtu vozidel za celý den, příkaz vypadá následovně

```
1 java -jar AllInOne.jar -h detektor.xml histogram.pdf
```

Pokud nás zajímá pouze určitý časový úsek, např. ranní špička, na konec příkazu zadáme příslušné meze

```
1 java -jar AllInOne.jar -h detektor.xml histogram.pdf 2:00 5:00
```

pro vytvoření histogramů počtu vozidel od 2 do 5 ráno.

2.3 Shrnutí práce s programem

Program byl vytvořen v jazyce JAVA jako podpůrný prostředek pro vytvoření jak vstupů, tak výstupů pro program SUMO při daném typu vstupních dat z Pražského okruhu. Program se spouští s příkazové řádky příkazem

```
1 java -jar AllInOne.jar
```

a dále zadáním příslušných parametrů.

Program se dělí na hlavních 5 částí a pro každou z nich je uveden příklad pro spuštění:

- vytvoření vstupního souboru vozidel pro SUMO
  - pravidelné rozložení - např. IDM model

    ```
    1 java -jar AllInOne.jar -c sokp-0201-20121001.mat vozidla.rou.xml IDM
    ```

  - z exponenciálního rozdělení - Kraussův model (není nutno uvádět)

    ```
    1 java -jar AllInOne.jar -c sokp-0201-20121001.mat vozidla.rou.xml exp
    ```

  - z exponenciálního rozdělení - Wiedemannův model

    ```
    1 java -jar AllInOne.jar -c sokp-0201-20121001.mat vozidla.rou.xml
        Wiedemann exp
    ```

- převedení formátu dat z Matlabu do formátu SUMO

```
java -jar AllInOne.jar -m sokp-0187-20121031.mat det0187.xml
```

- vytvoření grafu časové řady

```
java -jar AllInOne.jar -x detektor.xml graf.pdf
```

- vytvoření histogramu počtu vozidel za časový úsek - v tomto případě 1 hodina

```
java -jar AllInOne.jar -xh detektor.xml histogram.pdf 60
```

- vytvoření histogramu četností vozidel - v tomto případě od 2 do 5 hodin ráno

```
java -jar AllInOne.jar -xh detektor.xml histogram.pdf 2:00 5:00
```

3. Implementace nového modelu

V simulačním nástroji SUMO jsou již některé z modelů uvedených v kapitole 1 implementovány. Např. Newellův model zde zatím implementován není.

V tuto chvíli bych zde uvedl obecné informace, které s implementací nového modelu souvisí bez ohledu na to, o jaký model se konkrétně jedná.

Než začneme, je nutné mít připravené soubory pro nový model, pro Gippsův model jsou to soubory *MSCFModel_Gipps.cpp* a *MSCFModel_Gipps.h*. Postup, jak nové soubory vytvořit a implementovat do SUMO je uveden v příloze C.

3.1 MSCFModel.h

V souboru *MSCFModel.h* (ne *MSCFModel_Gipps.h*) se nacházejí definice základních funkcí, které posléze v přidaném modelu implementujeme. Některé z nich použijeme u našeho implementovaného modelu. Ty, které nepoužijeme, zůstanou stejné a při použití našeho modelu během simualce se budou volat funkce implementované v *MSCF.cpp*, případně *MSCF.h*.

Při procházení zdrojových kódů je na první pohled vidět, že většina obecných funkcí je primárně vytvořena z Kraussova modelu. Zbytek funkcí počítá nějakou obecnou vlastnost, která přímo nesouvisí s modelem. Tyto funkce nemají u své definice klíčové slovo virtual, které značí, že tuto funkci je možno nahradit při vlastní implementaci nového modelu. Dále jsou zde proměnné, které při vytvoření nového modelu automaticky přístupné. **Tyto proměnné se používají i při jiných výpočtech jako je např. vjezd vozidla do oblasti. Je proto nutné dbát na to, abychom v implementaci nového modelu neuložili špatnou hodnotu (např. decelerace).**

Funkce obsahují většinou následující vstupní proměnné:

- *const MSVehicle* const veh* - ukazatel na aktuální vozidlo, přes něj je možné zjistit detailnější informace o vozidle
- *SUMOReal speed* - rychlost aktuálního vozidla
- *SUMOReal gap2pred* - vzdálenost mezi aktuálním a vedoucím vozidlem
- *SUMOReal predSpeed* nebo *SUMOReal vL* - rychlost vedoucího vozidla
- *SUMOReal predMaxDecel* nebo *SUMOReal vL* - maximální decelerace vedoucího vozidla

Ostatní vlastnosti vedoucího vozidla nejsme schopni jednoduše zjistit, nicméně rychlost a vzdálenost by při implementaci nových modelů měly plně postačit.

Seznam funkcí, které je možné přepsat, je následující:

- *virtual SUMOReal freeSpeed(const MSVehicle* const veh, SUMOReal speed, SUMOReal seen, SUMOReal maxSpeed, const bool onInsertion = false) const;* - výpočet

rychlosti vozidla, které jede jako první, tz. jeho poloha je nejdále od začátku simulace a nejede před ním žádné vozidlo

- *virtual SUMOReal followSpeed(const MSVehicle* const veh, SUMOReal speed, SUMOReal gap2pred, SUMOReal predSpeed, SUMOReal predMaxDecel) const* - výpočet rychlosti vozidla na základě vlastností vedoucího vozidla
- *virtual SUMOReal insertationFollowSpeed(const MSVehicle* const veh, SUMOReal speed, SUMOReal gap2pred, SUMOReal predSpeed, SUMOReal predMaxDecel) const* - výpočet rychlosti vozidla při vjezdu do oblasti na základě vozidla, které se nachází před ním. **SUMO nenechá vjet vozidlo do oblasti, pokud není zajištěna bezpečná vzdálenost pro zabrždění vozidla.**
- *virtual SUMOReal stopSpeed(const MSVehicle* const veh, const SUMOReal speed, SUMOReal gap2pred)* - výpočet rychlosti ve chvíli, kdy se před vozidlem nachází statický objekt[1]
- *virtual SUMOReal interactionGap(const MSVehicle* const veh, SUMOReal vL) const;* - výpočet mezery, kdy ovlivňuje vedoucí vozidlo aktuální

Můžeme také využít další podpůrné funkce, které přímo nezávisí na modelu. Dají se použít při např. při kontrole bezpečné rychlosti. Tyto funkce nelze primárně přepsat v novém modelu. Jsou to:

- *SUMOReal maximumSafeFollowSpeed(SUMOReal gap, SUMOReal predSpeed, SUMOReal predMaxDecel) const;* - výpočet maximální bezpečné rychlosti pro aktuální vozidlo na základě vedoucího
- *SUMOReal maximumSafeStopSpeed(SUMOReal gap) const;* - výpočet bezpečné rychlosti pro zastavení uvnitř mezery

3.2 Implementace Gippsova modelu

K tomu abychom mohli přidat do SUMO Gippsův model, je nejprve nezbytné vytvořit soubory pro nový model - viz příloha C. Pokud vše funguje jak má, spustíme kompilaci SUMO a pokud vše proběhne bez problémů můžeme postoupit dále. Jinak je ovšem chyba v nedodržení postupu v příloze a není vhodné postupovat dále.

Ve chvíli, kdy se nám SUMO zkompilovalo a máme k dispozici soubory *MSCFModel_Gipps.cpp* a *MSCFModel_Gipps.h*, můžeme začít programovat Gippsův model. V příloze C v sekci C.2 je k dispozici návod na vytvoření vlastních parametrů, my ho použijeme pouze pro preferovanou rychlost vozidla *desiredSpeed* a tím pádem při tvorbě konstruktoru by nám měla být k dispozici hodnota proměnné *desiredSpeed*, kterou můžeme přidat jako parametr při vytváření souboru vozidel.

[1] objekt, který má nulovou rychlost

Prvně zakomponujeme do souboru *MSCFModel_Gipps.h* všechny naše nové proměnné, které budou k dispozici našemu modelu. Uložíme je do sekce protected, aby k nim měl přístup pouze Gippsův model, případně jeho potomek.

```
protected:
        SUMOReal myDesiredSpeed;
        SUMOReal myGippsDecel;
```

Kromě *myDesiredSpeed* jsme zde přidali ještě jednu proměnnou a to z důvodu zpřehlednění (viz dále).

Nyní již vytvoříme konstruktor, který bude naplní proměnné zděděné od obecného modelu a zároveň naše nově přidané proměnné.

```
MSCFModel_Gipps::MSCFModel_Gipps(const MSVehicleType* vtype, SUMOReal accel,
    SUMOReal decel,
        SUMOReal headwayTime, SUMOReal desiredSpeed)
    : MSCFModel(vtype, accel, decel, headwayTime), myDesiredSpeed(desiredSpeed)
        {
                myGippsDecel = -decel;
}
```

Je také nutné v hlavičkovém souboru *MSCFModel_Gipps.h* upravit definici stávajícího konstruktoru na

```
 MSCFModel_Gipps(const MSVehicleType* vtype, SUMOReal accel, SUMOReal decel,
     SUMOReal headwayTime, SUMOReal desiredSpeed);
}
```

MSCFModel_Gipps.h Tento kód zavolá konstruktor obecného modelu, uloží nám hodnoty zadané při generování vozidel do simulace do jednotlivých proměnných. My si potom přidáme ještě hodnotu *desiredSpeed* do proměnné *myDesiredSpeed* a do proměnné *myGippsDecel* uložíme zápornou hodnotu decelerace.

Není možné uložit do myDecel hodnotu -decel, protože na hodnotu myDecel se ptá SUMO v různých situacích a byla by v něm uložena záporná hodnota namísto kladné, které SUMO očekává.

V tuto chvíli bychom měli mít k dispozici následující proměnné v celém těle Gippsova modelu. Uvedeme zde také proměnnou odpovídající parametrům Gippsova modelu - viz 1.2.2:

- myAccel - akcelerace a_n,
- myGippsDecel - decelerace b_n,
- myDesiredSpeed - preferovaná rychlost V_n,
- myHeadwayTime - reakční čas T_n.

Ostatní proměnné se mění v čase a vždy jsou vstupními parametry dané funkce.

Gippsův model pracuje tak, že vybírá minimální hodnotu z dvou možností: z rychlosti ve volném proudu či rychlosti při interakci z vedoucím vozidlem - viz rovnice 1.20. My si

obě části této rovnice naprogramujeme, každá část rovnice bude jedna funkce. V souboru *MSCFModel_Gipps.h* je obě zadefinujeme,

```
1        private:
2       virtual SUMOReal vFree(SUMOReal speed) const;
3                       virtual SUMOReal vInteraction(SUMOReal speed, SUMOReal
                            predSpeed, SUMOReal gap2pred) const;
4                       virtual SUMOReal chooseBrakeConst(SUMOReal decel) const;
```

parametry těchto funkcí přímo odpovídají tomu, co pro výpočet daného členu potřebujeme. K funkci *chooseBrakeConst* se dostaneme za chvíli. Funkce *vFree* potom vypadá následovně:

```
1  SUMOReal MSCFModel_Gipps::vFree(SUMOReal speed) const {
2      SUMOReal actualSpeed = speed;
3          SUMOReal result = actualSpeed + 2.5*myHeadwayTime*myAccel*(1-actualSpeed
               /myDesiredSpeed)*sqrt(0.025 + actualSpeed/myDesiredSpeed);
4      return result;
5  }
```

Funkce přesně kopíruje vzorec 1.12, který je u Gippsova modelu vzorec pro rychlost při volném proudu. Pokud je preferovaná rychlost V_n rovna aktuální rychlosti $v_n(t)$, celý člen za proměnnou actualSpeed se vyruší a dostáváme, že rychlost v následujícím kroku bude přímo stejná jako aktuální.

Interakční člen nám dokumentuje funkce *vInteraction* obsahující výpočet rychlosti na základě vedoucího vozidla. Její kód je následující:

```
1  SUMOReal MSCFModel_Gipps::vInteraction(SUMOReal speed, SUMOReal predSpeed,
       SUMOReal gap2pred) const{
2           SUMOReal wholeGap = (gap2pred+myType->getMinGap())*2;
3           SUMOReal speeds = speed*myHeadwayTime + ((predSpeed*predSpeed)/(2*
                chooseBrakeConst(myGippsDecel)));
4           SUMOReal sqrtPart = sqrt(myGippsDecel*myGippsDecel*myHeadwayTime*
                myHeadwayTime - myGippsDecel*(wholeGap - speeds));
5           SUMOReal rV = myGippsDecel*myHeadwayTime + sqrtPart;
6           if (rV < 0)
7                   rV = 0;
8           return rV;
9   }
```

Proměnná *wholeGap* odpovídá části $[x_{n-1}(t) - s_{n-1} - x_n(t)]$. Tu bylo potřeba upravit pro účely SUMO. Nemáme totiž možnost zjistit parametr s_{n-1}, jenž závísí na délce vedoucího vozidla a k této informaci se jednoduše nedostaneme. Proto jsme upravili tento člen jako součet vzdálenosti mezi vozidly (předek aktuálního vozidla a zadek vedoucího vozidla) a k ní jsme přičetli minimální bezpečnostní odstup. Tím bychom měli dostat stejný výsledek jako je uvedeno v Gippsově modelu.

Další zajímavostí je vypočet členu $\hat{b}$. Při kalibraci Gippsova modelu vyšla nejlépe varianta brzdná konstanta vedoucího vozidla bude rovna:

$$\hat{b} = \min\left(-3.0, \frac{b_n - 3.0}{2}\right) m/sec^{-2}. \tag{3.1}$$

Tato rovnice přepsaná do funkce se rovná přesně obsahu funkce *chooseBrakeConst* definovanou v části private, viz výše. Její kód je poměrně jednoduchý:

```
SUMOReal MSCFModel_Gipps::chooseBrakeConst(SUMOReal decel) const{
        return MIN2(-3.0, (- decel - 3.0)/2);
}
```

Funkce *MIN2* je vlastní funkce SUMO, která hledá minimum ze dvou hodnot.

Pro větší přehlednost jsme kód funkce *vInteraction* rozdělili na více částí. Podstatná je proměnná *myGippsDecel*, kterou jsme na začátku nastavili jako zápornou a proto zde figuruje s kladným znaménkem.

V tuto chvíli máme připravené obě funkce pro výpočet rychlostí pro vozidla vjíždějící do simulace.

V další fázi je potřeba tyto funkce využít pro výpočty. Všechny následující funkce budeme pouze nahrazovat již stávajícími, které obsahuje *MSCFModel*. Vždycky je otázkou, zda je nutné danou funkci přepisovat a nebo nechat její stávající verzi.

Začneme funkcí *freeSpeed*. Tato funkce nám vypočítává rychlost při volném proudu, tj. ve chvíli kdy před aktuálním vozidlem nejede žádné další. Mohli bychom tuto funkci nechat stejnou, tak jak je implementována v SUMO, ovšem daleko lepší bude tuto funkci přepsat a to z toho důvodu, že Gippsův model přímo uvažuje o rychlosti při volném proudu. Definice přepisovaných funkcí necháme vždy stejné jako v *MSCFModel.h*, pro *freeSpeed* by definice[2] vypadala takto:

```
virtual SUMOReal freeSpeed(const MSVehicle* const /* veh */, SUMOReal /* speed
    */, SUMOReal seen, SUMOReal maxSpeed, const bool onInsertion) const;
```

Implementace naší vlastní funkce *freeSpeed* bude následující

```
SUMOReal
MSCFModel_Gipps::freeSpeed(const MSVehicle* const veh , SUMOReal speed,
    SUMOReal seen, SUMOReal maxSpeed, const bool onInsertion) const {
        return vFree(speed);
}
```

Je vidět, že většina vstupních parametrů funkce nejsou vůbec využity, nicméně musejí zde zůstat z důvodu kompatibility s ostatními modely. Gippsův model při výpočtu jednoduše tyto parametry jednoduše neřeší.

Další funkcí, kterou je nutné přepsat, bude zcela jistě *followSpeed*. Ta počítá rychlost v závislosti na vedoucím vozidle. My již víme, že tato funkce by měla vracet stejný výsledek jako rovnice 1.20. Její implementace vypadá následovně:

```
SUMOReal
MSCFModel_Gipps::followSpeed(const MSVehicle* const veh, SUMOReal speed,
    SUMOReal gap, SUMOReal predSpeed, SUMOReal predMaxDecel) const {
        return MIN2(vFree(speed), vInteraction(speed, predSpeed, gap));
}
```

[2]Každá definice zde zmiňována bude umístěna do souboru MSCFModel_Gipps.h do části public. Tato definice je shodná s definicí uvedenou v souboru MSCFModel.h

Bylo vyzkoušeno změnit i funkci *insertionFollowSpeed*, nicméně zatím se zdá, že stávající verze implementována v SUMO je plně postačující.

Další funkcí k přepsání je *stopSpeed*. Tato funkce má stejné parametry jako *followSpeed* až na parametr *predSpeed*, který je v tuto chvíli nulový, protože tato funkce počítá rychlost, pokud se před ní nachazí statický objekt. Proto voláme stejné funkce jako u *followSpeed* s tím, že *predSpeed* má hodnotu nula.

```
1 SUMOReal
2 MSCFModel_Gipps::stopSpeed(const MSVehicle* const veh, const SUMOReal speed,
      SUMOReal gap) const {
3         return MIN2(vFree(speed), vInteraction(speed, 0, gap));
4 }
```

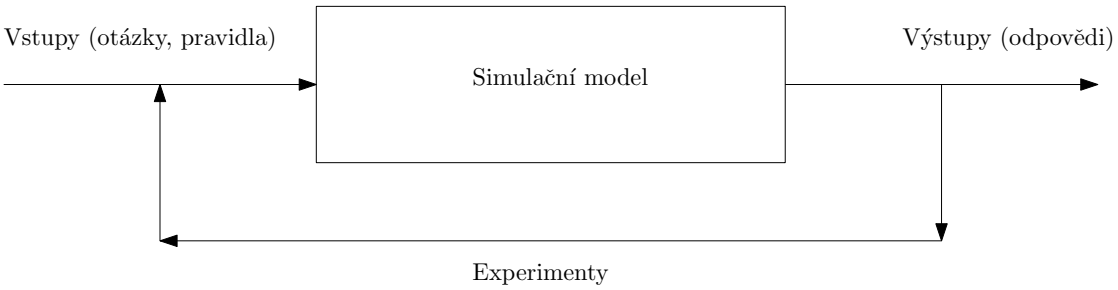
Tímto je nový model implementován.

4. Kalibrace modelů

4.1 Stručný úvod do simulačních modelů

Je známo, že simulace je vhodná možnost jak provést experimentální test v porovnání s odlišnými návrhy systému. Daleko lepší možností je pokus provést na modelu v počítači než na reálném systému a riskovat jeho poškození či zničení, navíc na některých reálných systémech nelze ani pokus provést. Výsledky tohoto experimentu na počítači mohou poskytnout mnoho užitečných informací, na jejichž základě je např. možné zvolit optimální variantu při rozhodování. Podle tohoto konceptu je tedy simulační model počítač, který je určen k řízení pokusů s modelem systému s cílem sestavení platných výsledků i pro reálný systém. Jinými slovy simulační model se používá k odpovědi na otázky typu „Co když?“.

Simulace je tedy experiment na modelu reálného systému.[14]. Za předpokladu, že vývoj modelu systému v čase vhodně napodobuje vývoj reálného systému, můžeme sbírat informace o modelu (např. proměnné) a posléze při použití statistickým metod či jiného vhodného matematického aparátu a následných dalších experimentů můžeme díky tomu navrhnout i budoucí chování systému. Tato myšlenka je znázorněna na obrázku 4.1.



Obrázek 4.1: Simulační model - zdroj:[1]

Spolehlivost procesu rozhodování záleží na schopnosti vytvořit takový simulační model, který reprezentuje chování reálného systému s dostatečnou přesností. Reálný systém potom může být nahrazen modelem a na modelu je možné provádět různé experimenty. Toto platí pro simulační analýzu obecně a zůstává platné i pro dopravní simulace. Proces určující, zda je simulační model dostatečně podobný reálnému systému, se nazývá **validace modelu** - iterativní proces zahrnující **kalibraci parametrů modelu** a porovnání chování modelu s chováním reálného systému (pokud je to možné). Nesoulady mezi systémem a modelem nutí systémového analytika k zjišťování dalších informací o systému. Tyto informace jsou potom použity k vylepšení modelu. Je nutné si uvědomit, že model by mohl být vylepšován neustále. Je proto nutné určit práh, kdy model již dává postačující výsledky, tj. kdy je jeho přesnost akceptována. Tento práh je právě dán validací. V tu chvíli již není nezbytné model dále vylepšovat. Validace simulačního modelu by měla být brána v potaz během celého procesu vytváření modelu.

4.2 Validace modelů

Hlavní zdroj pro informace k validaci modelů je použito [9]. V [2] je ještě více do podrobna rozebrána problematika validace a verifikace modelů.

Verifikace je v první řadě rozhodnutí zda simulační program vykonává přesně to, co bylo zamýšleno, to jest odstranění chyb z programu. Verifikace tedy kontroluje „přenesení“ simulačního modelu do správně fungujícího programu.

Verifikace obvykle znamená spuštění programu s implementovaným simulačním modelem s různým nastavením vstupních parametrů a kontrolou, zda jsou výsledky simulace přijatelné. V některých případech je možné některé výpočty provádět analyticky a posléze numerickou simulací výsledky porovnat a tím zjistit přijatelnost výstupů.

Validace modelu je hotova tehdy, pokud simulační model je přesná reprezentace zkoumaného systému. Pokud je model validní, mohou být rozhodnutí, která byla provedena na základě simulačního modelu stejná jako kdyby se prováděl experiment na reálném systému.

Model je **věrohodný** tehdy, když jeho výsledky jsou přijaty uživatelem a použity v rozhodovacím procesu.

Při validaci simulačního modelu by analytik neměl zapomenout, že

- simulační model je systém, který ovšem může být pouze aproximací reálného systému, bez ohledu na to, kolik práce bylo vloženo do vývoje modelu,
- simulační model by měl být vždy vyvíjen pro specifické účely,
- simulační model by mšl být validován vzhledem k perfomačním indikátorům souvisejících s účely modelu a mohl být použit pro rozhodování
- vývoj a validace modelu by měly probíhat zároveň během celé analýzy systému a vytváření simulačního modelu.

Law and Kelton [9] dále navrhli postup, jak vyvinout validní a věrohodné simulační modely. Dělí se na tři části:

- vytvoření modelu se „zjevnou“ validitou[1],
- testování předpokladů modelu empiricky,
- rozhodnutí, jak moc jsou výstupní data ze simulace reprezentativní.

Vytvoření modelu se zjevnou validitou Pokud lidé, kteří rozumí reálnému systému, potvrdí že simulační model vypadá racionálně, potom můžeme tvrdit že jde o model se zjevnou validitou.

Pro dosažení takového modelu je dle [9] nutné, aby

- člověk vytvářející nový model by měl spolupracovat s lidmi, kteří jsou důvěrně obeznámeni s reálným systémem,

[1] z anglického face validity

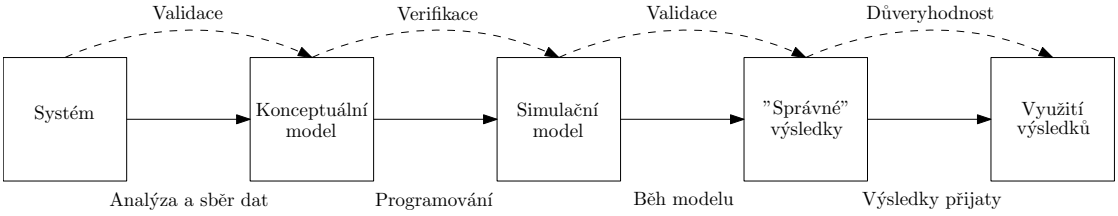
- analytik musí být důsledný při získávání všech požadovaných informací,
- pokud existují historická data, měla by být získána a použita při tvorbě modelu,
- musí se také dávat pozor, aby tato data byla správná a znázorňovala to, co je modelováno.

Testování předpokladů modelu empiricky Jestliže je použito pravděpodobností rozdělení jako vstupní parametr simulačního modelu na základě shody s reálnými daty, u výstupu musí být tyto okolnosti zohledněny.

Další vlastnost ovlivňující model je citlivost na počáteční podmínky. Pokud se výstupní data signifikatně změní pouze na základě nepatrné změny vstupních parametrů, model potom může působit chaoticky.

Tyto dvě podmínky musí být při vyhodnocení testování také zahrnuty.

Rozhodnutí, jak moc jsou výstupní data ze simulace reprezentativní Hlavní test, kterým se posuzuje validita simulačního modelu, je porovnání výstupních dat. Pokud se data ze simulačního modelu porovnají s výstupními daty a výsledek je příznivý, lze model označit za validní. Model ovšem vždy bude pouze aproximace systému a proto představa, že model a systém se rovnají nemůže být nikdy pravdivá.



Obrázek 4.2: Vztahy mezi validací, verifikací a důveryhodností - zdroj:[1]

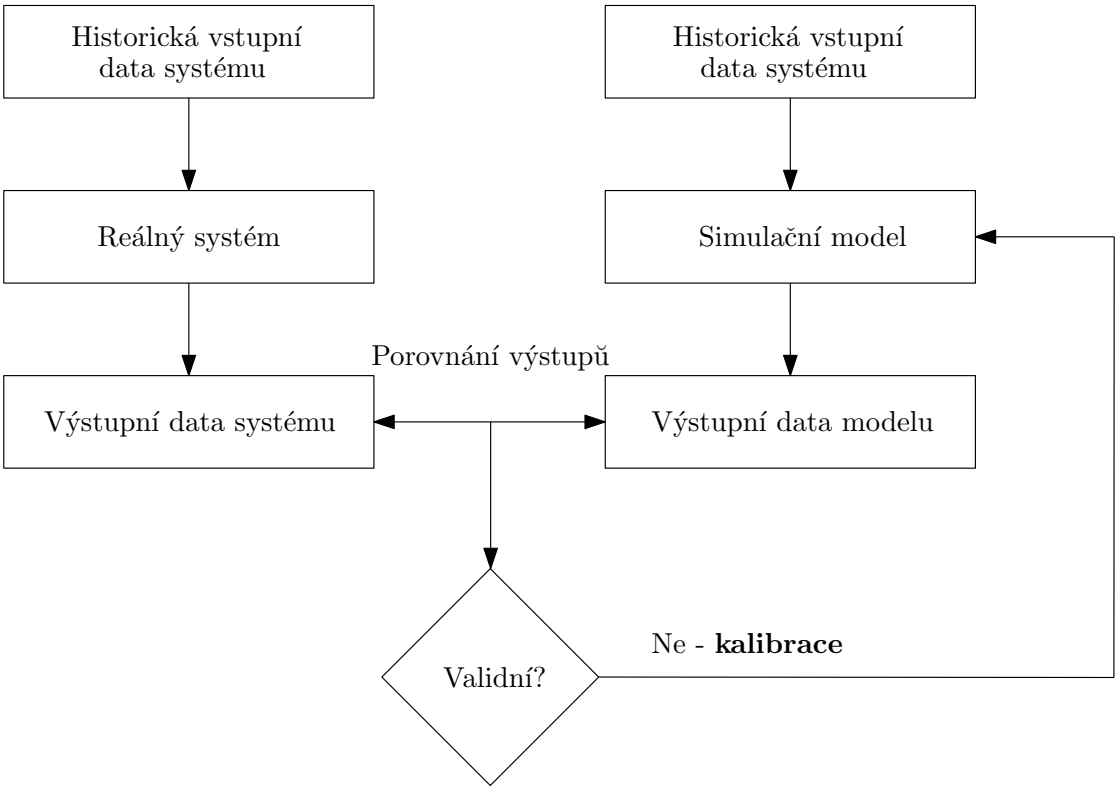
Shrnutí této části o validaci je názorně zobrazeno na obrázku 4.2. Obdelníky v tomto diagramu znázorňují stavy modelu a jejich vztah k procesu studia simulace. Přerušované čáry nad obdelníky ukazují, ve kterých situacích dochází k validaci, verifikaci a důveryhodnosti. Ačkoliv to není v obrázku 4.2 zahrnuto , je důležité nezapomínat, že celý proces modelování je nutné brát jako iterativní a že některé postupy se můžou opakovat a to několikrát.

Validace modelu tedy znamená proces testování, že model skutečně představuje uskutečnitelnou a použitelnou alternativu k experimentování se systémem. Jednou z části je kalibrace modelu, což znamená přizpůsobování parametrů modelu dokud výstupní data nepodobají datům pozorovaných na systému. Další otázkou potom je, zda je model nakalibrován pouze na určitou sadu vstupních dat či je nakalibrován všeobecně pro různé typy vstupních dat. Abychom tuto otázku náležitě zodpověděli, analytik musí použít dva nezávislé soubory dat, jednu pro kalibraci a druhou pro validaci. První soubor by měl být použit ke kalibraci modelu, druhý pro vyzkoušení nakalibrovaného modelu a poté k

validaci tohoto modelu. Výsledný soubor dat je posléze porovnán s druhým souborem dat ze systému.

Simulační model je validován na základě porovnání dat mezi pozorovanými výstupními daty systému a výstupními daty vytvořenými počítačovým modelem.

Tento proces je schématicky znázorněn na obrázku 4.3.



Obrázek 4.3: Diagram validace modelu na základě porovnání dat systému a modelu - zdroj:[1]

4.3 Kalibrace modelů

K tomu, jak správně nakalibrovat model neexistuje jednotný postup a většinou je to obecné know-how jednotlivých autorů kalibrace. Existuje několik článků o kalibraci např. [4], kde se hlavně dozvíme informace o kalibrace modelů General motors, které pracují na principu rovnice (1.11), více viz. [17, str.11-13].

Je zde popsána část kalibrace věnující se i nekolizním Car-Following modelům, jako je např. Gippsův model, nicméně je zde stručně uvedeno, že Gipps model nekalibroval, nýbrž použil reálné hodnoty z provozu a jeho model s nimi rovnou fungoval, tím pádem neměl důvod se kalibrací nějakým způsobem dál zabývat.

V článku [8] se autoři snaží nakalibrovat jednotlivé modely metodou pokus omyl. Řeší zde posléze pouze různé možnosti kontroly validity (viz dále). Kalibrace parametrů na principu optimalizace je složitá z toho důvodu, že účelová funkce nemá analytické řešení.

Další možnost, jak kalibrovat modely, byla vyzkoušena institutem pro výzkum dopravy v Berlíně [5], kde vyzkoušely Nelder-Mead[2] metodu, která se numericky snaží najít maximum nelineárních problému v několikarozměrném prostoru. Výhodou pro autory byla informace o rychlosti a poloze vozidla na testovacím okruhu každou 0.1s s tím, že nepřesnost určení rychlosti a polohy byla zanedbatelná.

Výsledkem z procházení informací o kalibraci docházíme k závěru, že ideální bude volit parametry také metodou pokus omyl, která je časově i výpočetně nenáročná. Podstatné bude, aby výsledky ze simulačních modelů odpovídaly výsledků naměřených mýtnými branami.

4.4 Statistické metody pro validaci modelů

Statistické metody pro validaci modelů jsou vysvětleny hlavně v [9], [1] a [10]. Jak jsme zmínili v minulé sekci, budeme potřebovat dvě sady dat, jednu ke kalibraci modelu a druhou k validaci modelu.

4.4.1 Statistický test střední hodnoty

V každém kroku validačního procesu by měl být prováděn experiment. Každý pokus je určen vstupními daty (které budou pro všechny pokusy stejné) a hodnotami parametrů modelu (ty se budou vždy lišit). Výstupem simulace bude soubor hodnot proměnných, v našem případě nejspíše intenzit, které zaznamenáváme každou vzorkovací periodu. Například, předpokládejme, že simulace má vzorkovací periodu 1 minuta - to znamená, že každou minutu přibývají hodnoty intenzit. Proměnná, která nás zajímá každou vzorkovací periodu, je intenzita w, výsledek simulačního modelu tedy bude soubor hodnot w_{ij}, kde indexem i značíme konkrétní detektor a j je čas zaznamenání. Index i probíhá od 1 do n,

$$i = 1, 2 \ldots \mathrm{n},$$

kde n je celkový počet detektorů a čas j probíhá od 1 do m,

$$j = 1, 2 \ldots \mathrm{m},$$

kde m je počet vzorkovacích period v simulaci.

Pokud v_{ij} označíme naměřenou intenzitu, v případě Pražského okruhu mýtnou branou, kde indexy i a j budou stejné jako v případě modelu, klasická statistická technika k validaci modelu bude porovnání hodnot v_{ij} a w_{ij} a zjištění zda jsou u sebe dostatečně blízko.

[2] též pojmenována Downhill simplexová metoda

Pro detektor i bude porovnání založeno na testování zda rozdíl

$$d_i = w_{ij} - v_{ij} \tag{4.1}$$

má střední hodnotu $\overline{d}_i$ významně vzdálenou od nuly či nikoliv. K tomu se využívá t-statistika

$$\bar{t}_{\mathrm{m}-1} = \frac{\overline{d}_i - \delta_i}{\frac{\overline{s}_d}{\sqrt{m}}}, \tag{4.2}$$

kde δ_i je předpokládaná hodnota $\overline{d}_i$ a $\overline{s}_d$ je směrodatná odchylka od $\overline{d}_i$.
Jako nulovou hypotézu označíme

$$H_0 : \delta_i = 0. \tag{4.3}$$

To nastane ve chvíli, kdy

$$\left|\bar{t}_{\mathrm{m}-1}\right| > t_{\mathrm{m}-1;\alpha/2}. \tag{4.4}$$

Pokud zamítneme hypotézu (4.3) na hladině α, soudíme, že model není dostatečně přesný v chování jako systém a model zamítneme. Musíme vyzkoušet jiné kombinace vstupních parametrů modelu.

Naopak pokud nezamítneme tuto hypotézu, tvrdíme, že chování simulace a systému jsou již natolik blízké, že simulaci považujeme za validní.

Tento proces musí být proveden pro každý z detektorů. Model je možné označit za validní ve chvíli, kdy hypotézu nezamítneme pro žádný z detektorů[3].

Pokud jde o tuto statistickou metodu validace, jsou zde další okolnosti, které je nutné vzít v úvahu [10] konkrétně při dopravních simulacích.

- Tento postup předpokládá stejná a nezávisle rozdělená pozorování[4], kdežto měření reálného systému a příslušné nasimulované výstupy nemusí tento předpoklad splňovat. Jednou z možností je vzít hodnoty z několika časových období a navíc zprůměrovat tyto hodnoty pro výpočet (4.1).
- Čím vyšší je vzorkovací perioda, tím nižší je kritická hodnota $t_{\mathrm{m}-1;\alpha/2}$. To naznačuje, že simulační model má větší šanci být zamítnut, tím jak roste vzorkovací perioda. T-statistika za těchto podmínek není vhodný nástroj pro validaci daného modelu.

[3] Záleží ovšem na dalších okolnostech, musí být vzato v úvahu, o jaká data se jedná či že model nikdy nemůže být stejný jako simulovaný systém.

[4] i.i.d - independent and identically distributed random variables

4.4.2 GEH

GEH byla vymyšlena v 70. letech v Anglii[5]. a používá se pro zjištění informace o tom, zda je model dobře nakalibrován. Bere v úvahu absolutní i relativní rozdíly hodnot intenzit. GEH formule vypadá takto:

$$GEH = \sqrt{\frac{2(m-c)^2}{m+c}}, \tag{4.5}$$

kde c je reálná intenzita provozu a m intenzita nasimulovaná modelem.

Rovnice 4.5 je stejná, jako rovnice pro chí-kvadrát test, ale toto není test v pravém slova smyslu. Je to empirický vzorec, o kterém bylo dokázáno, že se hodí pro analýzu dopravy. Ideálně funguje pro hodinové intenzity provozu, pro jiné časové úseky je doporučeno intenzity přepočítat.

Dle hodnoty GEH rozlišujeme tyto tři stavy:

- $GEH < 5$ - přijatelné, model je nakalibrován správně,
- $GEH > 5$ a $GEH < 10$ - možná chyba v kalibraci či v datech,
- $GEH > 10$ - vysoká pravděpodobnost chyby v kalibraci či v datech.

Na tomto základě je možné rozhodnout, zda výsledky, které dostaneme pomocí modelů jsou věrohodné či nikoliv.

[5] Jmenuje se dle iniciálů svého objevitele Geoffreyho E. Haverse

Literatura

[1] *Aimsun 7 Dynamic Simulators User's Manual*. TSS-Transport Simulation Systems, 2012.

[2] Balci, Osman. Verification, validation, and accreditation. *Proceedings of the 30th conference on Winter simulation*. IEEE Computer Society Press, 1998.

[3] Behrisch M, Bieker L, Erdmann J, Krajzewicz D, *SUMO - Simulation of Urban MObility*, Dokořán, Praha, 2008

[4] MARK, Brackstone a McDonald MIKE. Car-following: a historical review.*Transportation Research Part F 2 (1999)*. 2000.

[5] BROCKFELD, Elmar, Reinhart D. KÜHNE a Peter WAGNER. Calibration and validation of microscopic traffic flow models. *Transportation Research Record: Journal of the Transportation Research Board*. 2004.

[6] Gipps, P. G. (1981). A behavioural car-following model for computer simulation. *Transportation Research Part B: Methodological*, 15(2), 105-111.

[7] HIGGS, Bryan; ABBAS, M.; *MEDINA, Alejandra. Analysis of the Wiedemann Car Following Model over Different Speeds using Naturalistic Data*. Road Safety and Simulation, Indianapolis, Indiana, 2011.

[8] KANAGARAJ, Venkatesan a Gowri ASAITHAMBI. Evaluation of Different Vehicle Following Models under Mixed Traffic Conditions. *Procedia - Social and Behavioral Sciences 104*. 390 – 401. Dostupné z: http://ac.els-cdn.com/S1877042813045230/1-s2.0-S1877042813045230-main.pdf?_tid=fdf19d26-f011-11e4-97be-00000aacb35e&acdnat=1430492259_b7e70449c1b8d5805fd032cf131ac585

[9] KELTON, David a Averill LAW. *Simulation modeling and analysis*. New York: McGraw-Hill, 1991.

[10] Kleijnen, Jack PC. *Statistical validation of simulation models*. European Journal of Operational Research 87.1 (1995): 21-34.

[11] KRAUSS, Stefan. *Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics*. Köln, 1998. PhD. Thesis. Universität Köln.

[12] May A D, *Traffic flow fundamentals*, Prentice Hall, 1989

[13] Newell, G. F. (2002). A simplified car-following theory: a lower order model. *Transportation Research Part B: Methodological*, 36(3), 195-205.

[14] Pidd, M. *Computer Simulation in Management Science*. 3. vyd. Chichester: John Wiley & Sons.

[15] Treiber M, Hennecke A, Helbing D, *Congested traffic states in empirical observations and microscopic simulations*, Physical Review E, **62** (2), pp. 1805–1824, 2000

[16] Treiber M, *Microsimulation of road traffic flow* [online], dostupné z `http://www.traffic-simulation.de/`

[17] VANIŠ, Miroslav. *Matematické modelování vybraných problémů v dopravě v jazyce Java*. Praha, 2013. Bakalářská práce. FD ČVUT.

A. Příloha A: Kompilace simulačního softwaru SUMO

A.1 SUMO

SUMO je volně šiřitelný dopravní simulátor, který vznikl v roce 2001. SUMO umožňuje modelování intermodálních dopravních systémů. Ty mohou obsahovat dopravní prostředky, městskou hromadnou dopravu či chodce. Dále v SUMO existují nástroje pro vyhledávání cesty, visualizace, import či export map, počítání emisí a další. Nejdůležitější vlastností pro nás je možnost implmentování vlastního dopravního modelu. SUMO samo o sobě již obsahuje některé modely - Kraussův model 1.3, Intelligent Driver Model **??** a Wiedermanův model.

A.1.1 SUMO download

Na stránkách projektu SUMO[1] se stáhne aktuální verze SUMO na pravé straně. Uvažujeme pouze případ dopravního simulátoru SUMO pod Windows, v době vzniku byla na světě verze SUMO 0.22. Ideální je varianta stáhnutí tří zkomprimovaných souborů. Pro potřeby kompilace stačí pouze soubor SUMO 0.22 sources (sumo-src-0.22.0,14 MB). V souboru SUMO 0.22 manual, tutorial and examples (sumo-doc-0.22.0.zip, 35 MB) se nachází podrobná dokumentace ke všem naprogramovaným součástem SUMO a SUMO 0.22 for Windows (sumo-winbin-0.22.0.zip, 33 MB) obsahuje již zkompilované SUMO, které je možné rovnou použít. V tomto souboru se ve složce *docs/userdoc* nachází uživatelská příručka, která obsahuje i návod na zkompilování SUMO pod Windows. Tento návod ovšem v některých částech není dostatečně přesný.

A.2 Visual Studio

Pro zkompilování SUMO je zapotřebí mít nainstalované Visual Studio. To je možné získat přímo na stránkách Visual Studia[2] a po zaregistrování je možno získat Expres edici Visual Studia. Autor pracuje s Visual Studio 2010 Pro. Kompilace SUMO je tedy vyzkoušena na této verzi. **Pro správné fungování Visual Studia je nutné mít nainstalován Microsoft .NET Framework verze 4 ne vyšší!**

A.3 Ostatní závislosti

SUMO ke svému zkompilování potřebuje další knihovny. V první řadě je to **Xerces-C** (xerces-c-3.1.1-x86-windows-vc-10.0.zip, 23 MB)[3], které už je samo o sobě zkompilované.

[1] http://www.dlr.de/ts/sumo/en
[2] http://www.visualstudio.com
[3] http://mirror.hosting90.cz/apache//xerces/c/3/binaries/xerces-c-3.1.1-x86-windows-vc-10.0.zip

Podstatné je stáhnout verzi x86 a ne 64bitovou, aby s ní byly schopni další součásti pracovat.

Další závislostí jsou **knihovny FOX**, kterou si budeme muset ve Visual Studiu zkompilovat sami. V návodu u SUMO je uvedeno, že je vyzkoušena pouze verze 1.6.36[4]. Přesně pro tuto jedinou verzi se mi povedlo provést kompilaci v pořádku.

Poslední částí pro zdárnou kompilaci jsou FWTools. Verze doporučená SUMO ovšem nefunguje a ani nikde na internetu se ji nepodařilo nalézt. Funkční je ovšem verze 2.4.7[5].

Důrazně se doporučuje ani jednu z těchto částí neinstalovat do klasického adresáře Program Files, může to způsobit problémy.

Nutné je také mít nainstalovaný Python (pro něj neplatí předchozí odstavec), ovšem pozor rozhodně ne nejnovější verzi[6], nýbrž verzi 2.7[7]. U Pythonu není nutné žádné další nastavení.

A.3.1 Xerces-C

Nejprve rozbalíme obsah archivu a posléze překopírujeme ze složky *bin* obě dll knihovny[8] do složky *Windows/System32*. Přes veškerou snahu se nepodařilo XERCES naimportovat všem projektům. Tím pádem se musí celý obsah složky *include* překopírovat do složky, kde je nainstalované Visual Studio do složky *vc/include* [9]. Výsledkem tedy bude překopírovaná složka *Xercesc* ze složky *Xerces/include/* do složky */vc/include*. Pro kompilaci a následné spuštění zkompilovaných programů je nutno vytvořit systémovou proměnnou XERCES[10], jejíž hodnota bude nastavena na kořenový adresář Xerces. Poté vyhledáme v systémových proměnných proměnnou PATH a nakonec hodnoty této proměnné přidáme %XERCES%/bin/.

A.3.2 FOX

FOX klasicky rozbalíme z archivu a poté musíme Visual Studiem zkompilovat. Otevřeme v rozbaleném archivu složku *windows/vcpp* a v ní otevřeme Visual Studiem soubor *win32.dsw*. Dle verze Visual Studia se může objevit hlášení o nutnosti konvertování. Potvrdíme pro všechny (možno zaškrtnout) a necháme Visual Studio načíst FOX. Po načtení musíme vybrat možnost kompilace **Release** a zkompilujeme **pouze** projekt foxdll. Pak provedeme to samé s tím rozdílem, že vybereme možnost **Debug**. Při kompilaci můžou vzniknout chyby, ty nás ovšem netrápí, protože to důležité by se mělo vytvořit. V kořenovém adresáři se vytvoří složka *lib* ve které se nacházejí zkompilované knihovny FOXDLL-1.6.dll a FOXDLLD-1.6.dll, které také překopírujeme do složky *Windows/System32*. Do systémových proměnných je nutné přidat proměnnou z názvem FOX16 s ces-

[4] http://ftp.fox-toolkit.org/pub/fox-1.6.36.zip
[5] http://home.gdal.org/fwtools/FWTools247.exe
[6] 3 a výše
[7] https://www.python.org/ftp/python/2.7.8/python-2.7.8.msi, 16 MB
[8] xerces-c_3-1.dll a xerces-c_3_1D
[9] např. c:/Program Files/Microsoft Visual Studio 10.0/VC/include/
[10] Start - Počítač - Vlastnosti systému - Upřesnit nastavení systému (nabídka na pravé straně) - Proměnné prostředí... - Systémové proměnné - Nová...

tou do kořenového adresáře FOX[11] a následně na konec hodnoty systémové proměnné PATH přidat %FOX16%/lib.

A.3.3 FWTools

FWTools klasicky nainstalujeme (ideálně ne do složky Program Files). Vytvoříme stejně jako pro FOX systémovou proměnnou s názvem PROJ_GDAL, která bude opět odkazovat do kořenového adresáře FWTools. Poté vyhledáme v systémových proměnných proměnnou PATH a nakonec hodnoty této proměnné přidáme %PROJ_GDAL%/bin.

A.3.4 SUMO

V kořenovém adresáři SUMO ve složce **/build/msvc10** se nachází soubor s názvem *prj.sln*. Ten otevřeme Visual Studiem. Pokud je vše nastaveno správně, je možno spustit kompilaci[12] SUMO, to znamená zkompilovat všechno (všech 54 projektů). Pokud kompilujeme v režimu Release, dostaneme výstup, který se nachází v kořenovém adresáři ve složce *bin*. Tento výstup je v sumo-winbin-0.22.0.zip.

[11] např. C:/fox-1.6.36/
[12] Build → Build Solution

B. Příloha B: Vytvoření sítě pro simulaci v SUMO

B.1 OpenStreetMap

Pro vytvoření sítě bylo využito serveru OpenStreetMap, který obsahuje kromě pozemních komunikací spoustu dalších objektů. Nejdříve bylo na tomto serveru vybrána oblast, ve které se nacházejí mýtné brány. Posléze tato mapa byla vyexportována do souboru OSM. Ten ovšem obsahoval i další objekty, které nemají pro simulování žádnou relevanci. Tyto objekty bylo nutné odstranit.

K tomu se využil free software JOSM[1], kde se vybraly všechny prvky, které bylo nutné odstanit. Tyto prvky byly sice odstraněny, ovšem velikost souboru se ještě zvýšila, protože ke každému objektu byl pouze přirazen parametr skrytí.

Aby se prvky opravdu smazali byl využit software XMLStartlet[2], který přímo pracuje se souborem XML. Ve stejné složce, kde se nachází XMLStartlet se překopíruje soubor OSM s již odstraněnými prvky (*input.osm*) a s příkazové řádky se spustí příkaz

```
1 xmlstarlet ed -d "/osm/*[@action='delete']" < input > output.osm
```

kde *output.osm* je název výstupního souboru s přímo smazanými prvky.

B.2 netConvert

Pokud máme připravený soubor z odstavce B.1, zkopírujeme ho do složky se souborem *netconvert.exe*. Z příkazové řádky spustíme příkaz

```
1 netconvert --guess-ramps --remove-geometry --junctions.join --osm-files output.
    osm --output-file nodes.net.xml
```

Tento příkaz nám vytvoří soubor *nodes.net.xml*, který obsahuje již přesnou strukturu sítě, kterou bude možné otevřít v SUMO. Existuje mnoho parametrů netconvert, po několika zkouškách autor usoudil, že tyto parametry jsou nejblíže tomu, čeho se chce dosáhnout. Program *netconvert.exe* se pokusí uhádnout nájezdy a sjezdy s Pražského okruhu (*–guess-ramps*), které nejsou zřejmé ze souboru *output.osm*, s tím souvisí vytvoření propojení (*–junctions.join*) těchto nájezdů a sjezdů a vytvoření dalších uzlů (*–remove-geometry*). V tuto chvíli máme již použitelný soubor jako podklad pro simulaci.

SUMO pouští vozidla do oblasti tak, že vezme jeden z parametrů vozidla, který určuje vjezd do oblasti, v dalším parametru jsou informace o dalších hranách, které vozidlo projíždí. Problém výstupního souboru *nodes.net.xml*, se kterým jsou tyto parametry vozidla srovnávány, spočívá v tom, že hrany a uzly jsou 8 až 9ti místná čísla, která nemají na první pohled danou spojitost. Je tedy nutné je přejmenovat.

[1] https://josm.openstreetmap.de/
[2] http://xmlstar.sourceforge.net/

Nejlepší způsob je asi v použití SUMO-GUI, kde při najetí na prvek zjistíme jeho ID a potom v souboru *nodes.net.xml* nahradíme předem vymyšleným ID z nějakého vlastního systému. Autor zvolil číslování od 1 a v závislosti na směru bud písmeno a nebo b. Takže hrana má např. název 1a, 3b.

C. Příloha C: Vytvoření souborů pro nový model

C.1 Implementace nového modelu

Ideální volbou pro implementaci nového modelu je začít s modelem, který již existuje. Pokud se nacházíme v kořenovém adresáři simulačního nástroje SUMO, potom ve složce */src/microsim/cfmodels* nalezneme všechny dosud naprogramované modely.

Nejjednoduší způsob je zkopírování souborů s již existujícím modelem (např. *MSCFModel_KraussOrig2.h* a *MSCFModel_KraussOrig2.cpp* a jejich následné přejmenování na jméno našeho modelu (např. Gipps). Výsledkem tedy budou dva soubory *MSCFModel_Gipps.h* a *MSCFModel_Gipps.cpp*. Nyní otevřeme postupně oba soubory a nahradíme každý výskyt *MSCFModel_KraussOrig1* za *MSCFModel_Gipps*.

Velmi důležitá věc je potom přidat tyto soubory do projektu[1]. To se ve Visual Studiu provede volbou *Project* → *Add Existing Item* a vyberou se námi vytvořené soubory, tedy např. *MSCFModel_Gipps.h* a *MSCFModel_Gipps.cpp*.

Posléze otevřeme soubor */src/utils/xml/SUMOXMLDefinitions.h* a přidáme do seznamu parametrů[2].

V souboru */src/utisk/xml/SUMOXMLDefinitions.cpp* poté přidáme opět k sekci modelů Gippsův:

```
1 { "carFollowing-GIPPS", SUMO_TAG_CF_GIPPS},
```

Závěrečnou fází přidání modelu je otevření souboru */src/microsim/MSVehicleType.cpp*, kde se ve funkci *build* nachází dělení, podle toho jaký model je nadefinován v souboru .rou.xml. Na tomto základě je zvolen vybraný model. Do tohoto dělení je nutné vložit následující kód:

```
1 case SUMO_TAG_CF_GIPPS:
2         vtype->myCarFollowModel = new MSCFModel_Gipps(vtype, accel, decel, sigma
              , tau);
```

Pokud chceme zvolit náš model při generování vozidel, existuje více možností jak to udělat. Nejjednodušší je vložit mezi tagy v souboru s příponou .rou.xml <*vtype*> a </*vtype*> tag <*carFollowing-Gipps*>. V něm poté budou následovat jednotlivé parametry modelu.

[1] Postup, jak se otevírá a kompiluje projekt SUMO, se nachází v příloze A v sekci A.3.4

[2] ideálně vyhledat nějaký stávající model - pro Krausse SUMO_TAG_CF_KRAUSS_ORIG1 a za něj (ne místo něj!) přidat SUMO_TAG_CF_GIPPS

C.2 Implementace nových parametrů

Např. u Gippsova modelu 1.2 je nutné přidat parametr preferované rychlosti V_n. Proto je zde uveden stručný návod pro přidání jednoho parametru. To je nezávislé na modelu do té doby dokud při volání konstruktoru nepřipojíme tento parametr jako vstupní.

Pokud se nacházíme v kořenovém adresáři SUMO, potom ve složce */src/utils/xml/* otevřeme soubory *SUMOXMLDefinitions.h* a *SUMOXMLDefinitions.cpp*. V prvně jmenovaném souboru *SUMOXMLDefinitions.h* do části *enum SumoXMLAttr* přidáme název nového atributu ve tvaru[3]

```
1 SUMO_ATTR_DESIREDSPEED,
```

V souboru *SUMOXMLDefinitions.cpp* v sekci *StringBijection<int>::Entry SUMOXMLDefinitions::attrs[]* přiřadíme tyto parametry proměnným pro SUMO a to ve tvaru

```
1 { "desiredSpeed", SUMO_ATTR_DESIREDSPEED},
```

Další částí nutnou vykonat je přidání parametrů do souboru */src/utils/xml/SUMOVehicleParserHelper.cpp*, konkrétně do metody *getAllowedCFModelAttr()*. Pro Gippsův model bude kód následující:

```
1 std::set<SumoXMLAttr> gippsParams;
2 gippsParams.insert(SUMO_ATTR_ACCEL);
3 gippsParams.insert(SUMO_ATTR_DECEL);
4 gippsParams.insert(SUMO_ATTR_TAU);
5 gippsParams.insert(SUMO_ATTR_CF_DESIREDSPEED);
6 allowedCFModelAttrs[SUMO_TAG_CF_GIPPS] = gippsParams;
```

[3] Je vhodné najít sekci parametrů Car Following Modelů a přidat ji tam, nicméně při vložení např. hned na začátek to bude fungovat také. Ideální je vyhledat text „SUMO_ATTR_CF_“