Mathématiques Orsay

Département de Mathématiques d'Orsay

MEU359 - PROJET MODÉLISATION, CALCUL SCIENTIFIQUE, PROBA-STATS: TRAFIC ROUTIER

Bechelor's Project

submitted to the Faculty of Mathematics
of the University of Paris Saclay (Campus Orsay) by

YANICK CHRISTIAN TCHENKO

Matriculation number : 22214770

24 avril 2023

Examiner :	Paul Melotti
Co-Examiner :	Olivier Henard, Maud Le-Bouedec, Benoit Gaudeul
Major :	Mathématiques Applications et Numérique (M. Sc.)
Semester :	6

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

ACRONYMS

IDM Intellgent Driver Models

1 INTRODUCTION

1.1 Motivation & Problème

La collecte de données constitue une limitation majeure de la science des données appliquée, par exemple, à la conduite autonome [BDSDMoo]. Cela est généralement dû au fait que le traitement des données brutes provenant des capteurs est insuffisant ou très laborieux. Une approche généralement utilisée comme alternative valable est l'utilisation de données synthétiques [BDSDMoo]. Il s'agit de données provenant de simulations de trafic qui sont suffisamment proches des données réelles. Une modélisation précise du trafic peut donc avoir un impact positif considérable sur la qualité des données synthétiques. Il s'agit d'une technique qui s'appuie fortement sur des fondements théoriques tels que la théorie des réseaux et certaines théories physiques telles que le modèle d'onde cinématique pour rendre la collecte de données viable. La quantité d'intérêt modélisée et mesurée est le flux de trafic, c'est-à-dire le flux d'unités mobiles par temps et par capacité du moyen de transport. L'intérêt croissant pour cette approche suscite donc la curiosité des chercheurs, qui affinent de plus en plus les modèles de trafic existants et donc la qualité des données synthétiques en développant des systèmes de trafic de plus en plus précis et cohérents.

1.2 Structure du rapport

Dans ce travail, nous proposons d'explorer une approche de la modélisation microscopique du trafic routier dans une situation très simplifiée. Dans le chapitre 2 et le chapitre 3, nous donnerons brièvement quelques détails sur la méthodologie adoptée dans l'implémentation, en commençant par des approches générales du problème de la modélisation du trafic routier jusqu'à des approches spécifiques à la présente étude. Nous présenterons ensuite nos résultats dans le chapitre 4 où nous conclurons également sur les aspects d'amélioration de notre mise en œuvre.

2

MÉTHODOLOGIE EMPLOYÉE

2.1 Concept General de Modelisation

Afin d'analyser et d'améliorer les systèmes de circulation, il est essentiel de commencer par créer un modèle mathématique représentant le système de circulation. Ce modèle doit être fidèle à la réalité en représentant de manière précise le flux de trafic en fonction de divers paramètres d'entrée tels que la géométrie du réseau routier, le nombre de véhicules par minute, la vitesse des véhicules, etc.

Les modèles de système de circulation sont généralement classés en trois catégories, en fonction du niveau auquel ils opèrent : (1) Les modèles microscopiques représentent chaque véhicule séparément et tentent de reproduire son comportement. (2) Les modèles macroscopiques décrivent le mouvement des véhicules dans son ensemble en termes de densité de trafic (véhicules par km) et de flux de trafic (véhicules par minute). Ils sont généralement analogues aux écoulements de fluides. (3) Les modèles mésoscopiques sont des modèles hybrides qui combinent les caractéristiques des modèles microscopiques et macroscopiques ; ils modélisent le flux comme des groupes de véhicules.

2.2 Modèle Microscopique Naïve de Trafic

Un modèle de circulation microscopique est utilisé pour décrire le comportement des véhicules individuels, ce qui signifie qu'il fonctionne comme un système multi-agent où chaque véhicule fonctionne indépendamment en fonction des entrées environnementales (Voir figure 2.1).

Par exemple, considérons deux véhicules étiquetés i et k dans ce type de modélisation, où le véhicule i suit le véhicule k. Leurs positions sur la route sont notées x_i et x_k, tandis que leurs vitesses sont v_i et v_k, et leurs longueurs sont l_i et l_k. La distance entre les deux

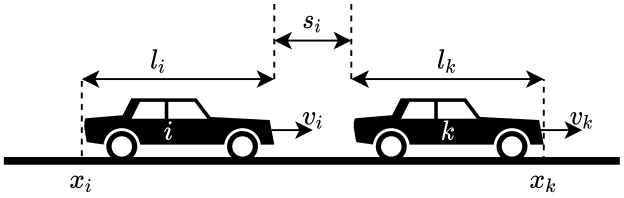

FIGURE 2.1 – Modèle microscopique

véhicules, appelée s_i, est la distance pare-chocs à pare-chocs et est calculée à l'aide des formules suivantes :

$$s_i = x_i - x_k - l_i \qquad \Delta v_i = v_i - v_k \tag{2.1}$$

Où Δv_i est la différence de vitesse entre le i-ème véhicule et le k-ème véhicule devant lui.

2.3 Modèle Microscopique Intelligent de Trafic

Treiber, Hennecke et Helbing [THH00] ont présenté le modèle de trafic intelligent (engl. IDM) qui explique comment l'accélération du véhicule i est déterminée par ses propres variables ainsi que celles du véhicule k. L'équation dynamique du modèle est la suivante :

$$\dot{v}_i = \frac{dv_i}{dt} = a_i\left[1 - (\frac{v_i}{v_{i,0}})^{\delta} - (\frac{s^*(v_i, \Delta v_i)}{s_i})^2\right] \tag{2.2}$$

avec

$$s^*(v_i, \Delta v_i) = s_{i,0} + v_i T_i + \frac{v_i \Delta v_i}{2\sqrt{a_i b_i}}$$

Ces équations impliquent plusieurs variables. $s_{0,i}$ qui représente la distance minimale souhaitée entre les véhicules i et k, $v_{i,0}$ qui est la vitesse maximale souhaitée par le véhicule i, et δ qui détermine la douceur de l'accélération. T_i, a_i et b_i représentent respectivement le temps de réaction, l'accélération maximale et la décélération confortable du véhicule i. La distance réelle souhaitée entre le véhicule i et k est représentée par s^*. De plus, $d = v_i T_i$ représente la distance de sécurité liée au temps de réaction, c'est-à-dire la distance parcourue par le véhicule avant que un arrêt. L'expression $\frac{v_i \Delta v_i}{2\sqrt{a_i b_i}}$ exprime la distance de sécurité basée sur la différence de vitesse. Elle représente la distance que le véhicule parcourra

pour ralentir sans heurter le véhicule devant et sans freiner trop fort.

Dans le cadre de ce travail, nous nous intéressons au IDM que nous proposons de mettre en œuvre après une clarification de son fonctionnement dans le paragraphe suivant.

2.4 Fonctionnement et Propriétés de l'IDM

Pour tout véhicule mobile, l'hypothèse est valable que ce dernier suit un mouvement décrit par une variation de sa vitesse en fonction du temps. Ainsi, cette variation sera nulle si le mouvement du véhicule est uniforme, négative resp. positive si le véhicule ralentit resp. accélère. Nous avons l'équation 2.2 désormais réduite comme suit :

$$\dot{v}_i = \frac{dv_i}{dt} = a_h + a_{int} \tag{2.3}$$

où nous pourrions décrire les valeurs a_h et a_{int} comme étant les accélérations propres au véhicule et celles causées par des facteurs extrinsèques. Ainsi, l'expression de l'accélération dans les équations 2.2 et 2.3 est une interpolation de la tendance à l'accélération avec a_h sur une route libre et la tendance à freiner avec la décélération a_{int} lorsque le véhicule se rapproche trop du véhicule qui le précède.

2.4.1 *Accélération Interpolée*

Nous définissons a_h et a_{int} comme suit :

— **Accélération homogène**
L'accélération homogène ou $a_{homogeneous}$ ou a_h est l'accélération sur une route libre. Elle décrit l'évolution de la vitesse d'un véhicule en un temps donné sur une route libre. Elle est maximale lorsque la vitesse est minimale et minimale lorsque la vitesse est maximale ; il n'y a plus d'accélération possible à une vitesse maximale. Elle est calculée suivant la formule

$$a_h = a_i\big(1 - (\frac{v_i}{v_{i,0}})^{\delta}\big)$$

Une représentation graphique de cette fonction correspondrait aux courbes observées à la figure 2.2. Pour des valeurs de δ comprises entre 1 et 5 et pour une vitesse

maximale fixée à $120Km/h$ on observe les phénomènes décrits ci-dessus sur la variation de l'accélération homogène.

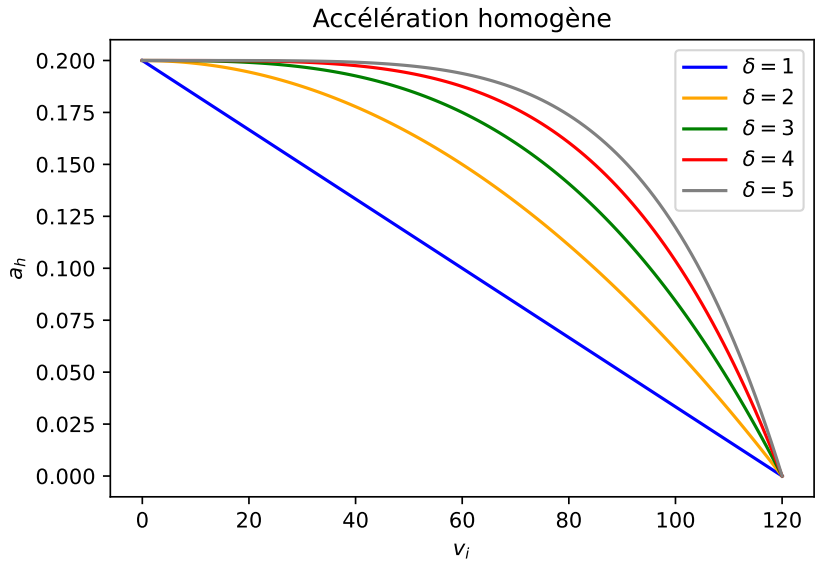

FIGURE 2.2 – Accélération homogène. Estimation pour une vitesse maximale de $120Km/h$ et différentes valeurs de δ. Dans tous les cas, l'accélération converge vers 0 au fur et à mesure que la vitesse évolue.

— **Accélération biaisée**

Comme indiqué précédemment, l'accélération biaisée décrit l'accélération due à des facteurs externes au véhicule, par exemple en cas d'interaction avec d'autres véhicules. Il s'agit de la décélération a_{int} due à la position du véhicule suivi très proche du pare-chocs du véhicule considéré. a_{int} est calculé selon la formule :

$$\begin{aligned} a_{int} &= -a_i\left(\frac{s^*(v_i, \Delta v_i)}{s_i}\right)^2 \\ &= -a_i\left(\frac{s_{i,0} + v_i T_i}{s_i} + \frac{v_i \Delta v_i}{2s_i\sqrt{a_i b_i}}\right)^2 \end{aligned}$$

L'évolution de la décélération et donc de la vitesse peut être observée dans la figure 2.3. Pour une distance minimale $s_{i,0} = 1m$ du véhicule i par rapport au véhicule k de la figure 2.1 et une vitesse de $10km/h$ lorsque le véhicule i est à 100m du véhicule k, on peut voir sur le graphique une décélération progressive à adopter sur $10min$ sans heurter le véhicule k.

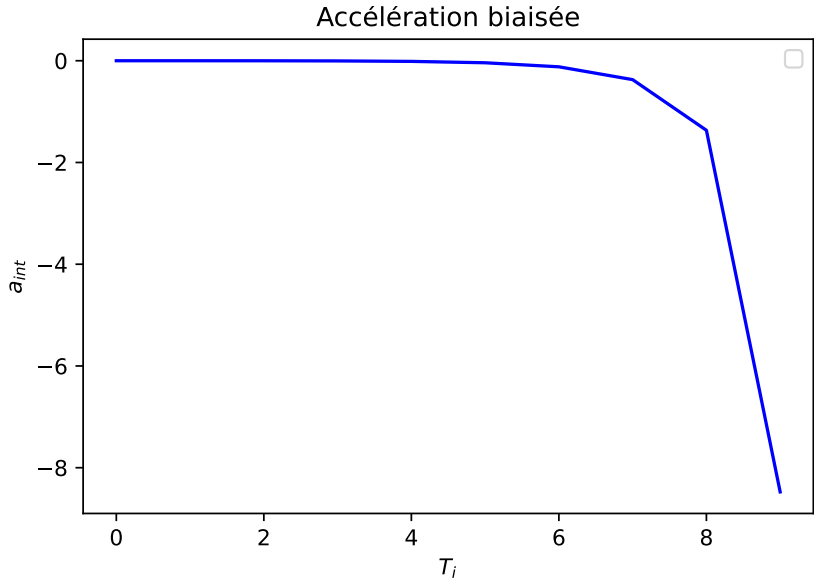

FIGURE 2.3 – Accélération biaisée : Variation négative de la vitesse (Freinage) en fonction du temps en cas d'interaction.

Sur la base de la distance s_i du véhicule suivant, on peut écrire de manière générale pour le modèle d'accélération du véhicule i la formulation suivante :

$$\dot{v}_i = \begin{cases} a_h = a_i(1 - (\frac{v_i}{v_{i,0}})^\delta), \ si \ s_i \to \infty \\ a_{int} = -a_i(\frac{s^*(v_i, \Delta v_i)}{s_i})^2, \ sinon \end{cases} \tag{2.4}$$

2.4.2 *Modélisation de la Dynamique du Véhicule*

La modélisation de la dynamique du véhicule en fonction de l'accélération interpolée entre a_h et a_{int} mentionnée ci-dessus est effectuée selon les principes des séries de Taylors [RZQ99]. Nous avons pour une fonction donnée f, infiniment différentiable sur un espace

vectoriel donné E avec $x \in E$, la formulation suivante pour le développement de la série de Taylor :

$$\begin{aligned} f(x) &= f(\alpha) + \frac{df}{dt}(\alpha)(x-\alpha) + \frac{d^2f}{dt^2}(\alpha)\frac{(x-\alpha)^2}{2} + ... \\ &= f(\alpha) + \sum_{k=1}^{\infty} \frac{d^k f(\alpha)}{dt^k}\frac{(x-\alpha)^k}{k!} \\ &= \sum_{k=0}^{\infty} \frac{d^k f(a)}{dt^k}\frac{(x-\alpha)^k}{k!} \end{aligned} \tag{2.5}$$

En posant $\alpha = \Delta x$ et $x = x + \Delta x$, l'équation 2.5 devient :

$$f(x+\Delta x) = \sum_{k=0}^{\infty} \frac{d^k f(\Delta x)}{dt^k}\frac{(x)^k}{k!} \tag{2.6}$$

La reformulation de cette équation en remplaçant f par x pour la description de la position du véhicule en fonction du temps donne

$$\begin{aligned} x(t+\Delta t) &= \sum_{k=0}^{\infty} \frac{d^k x(\Delta t)}{dt^k}\frac{(t)^k}{k!} \\ &= x(\Delta t) + \frac{dx}{dt}(t)\Delta t + \frac{d^2x}{dt^2}(t)\frac{\Delta t^2}{2} + ... \end{aligned} \tag{2.7}$$

$$\begin{aligned} &\implies \frac{d(x(t+\Delta t))}{dt} = \frac{dx(\Delta t)}{dt} + \frac{d^2x}{dt^2}(t)\Delta t + \frac{d^3x}{dt^3}(t)\frac{\Delta t^2}{2} + ... \\ &\implies v(t+\Delta t) \simeq v(\Delta t) + a(t)\Delta t \end{aligned} \tag{2.8}$$

Les équations 2.7 et 2.8 sont celles utilisées à chaque itération pour mettre à jour respectivement la position et la vitesse du véhicule en fonction du temps. L'estimation de l'accélération du véhicule est déduite suite à une refomulation de l'équation 2.8. On obtient ainsi $a(t) = -\frac{\Delta v(t)}{\Delta t}$. En initialisant la position, la vitesse et l'accélération à 0 et en fixant l'accélération, la vitesse et la décélération à des valeurs maximales de $20km/h$, $1.44ms^{-2}$ et $4.61ms^{-2}$ respectivement, nous obtenons une description de la dynamique d'un véhicule sous la forme d'un problème de Cauchy pour le système d'équations différentielles ordinaires. L'équation de vitesse étant linéaire et du premier ordre, le système admet une

solution unique pour la vitesse (et donc pour la position du véhicule) résultant d'une accélération donnée d'une position à une autre.

3

SETUP POUR L'EXPÉRIMENTATION

3.1 Modélisation du réseau routier et des feux de circulation

Après avoir défini le modèle (microscopique) de la dynamique du véhicule, nous en avons besoin lors de la modélisation du réseau routier où celui-ci doit s'appliquer. Pour cela, nous adopterons l'approche de Tan et al. [Tan+21] en faisant usage des graphes de connaissance. Nous définissons ici un graphe direct de la forme $G = (V, E)$ avec V l'ensemble des sommets (ou nœuds), c'est-à-dire les intersections, et E l'ensemble des arêtes qui représentent les routes.

Chaque véhicule défini une trajectoire composée de plusieurs routes et nous appliquerons l'IDM aux véhicules empruntant la même route. Lorsqu'un véhicule atteint la fin de la route, nous le retirons de cette route et l'ajoutons à la route suivante. Dans la simulation, nous ne conserverons pas un ensemble de nœuds, mais chaque route sera explicitement définie par les valeurs de ses nœuds de départ et d'arrivée.

Les feux de circulation sont placés à chaque nœud $n \in V$ et définissent une contrainte fondamentale d'adaptation de la dynamique du véhicule. En cas de feux rouges, nous revenons au cas de dynamique homogène décrit précédemment. Au feu rouge, deux cas de figure sont modélisés pour matérialiser la décélération : (1) la zone de ralentissement caractérisée par une distance de ralentissement d et un facteur de ralentissement k. Il s'agit d'une zone dans laquelle les véhicules ralentissent leur vitesse maximale en utilisant le facteur de ralentissement. Notons $v_{i,0}^{(k)} = kv_{i,0}$ la vitesse maximale du véhicule dans la zone de décélération. (2) En utilisant la force d'amortissement relative à l'accélération décrite dans équation 3.1, nous modélisons la dynamique du véhicule dans la zone d'arrêt. Elle est caractérisée par une distance d'arrêt et décrit la zone devant le feu de circulation dans laquelle les véhicules s'arrêtent.

$$\dot{v} = \frac{dv_i}{dt} = -b_i \frac{v_i}{v_{i,0}} \tag{3.1}$$

3.2 Modèle stochastique de générateur de véhicule

Cette section traite des méthodes utilisées pour générer les véhicules à utiliser dans le réseau routier. Les options suivantes sont principalement distinguées : (1) Nous pouvons ajouter manuellement des véhicules au réseau en incrémentant continuellement la liste des véhicules. (2) Une alternative à cette approche consisterait à générer les véhicules de manière stochastique en utilisant des formules probabilistes prédéfinies. Cette dernière variante est plus pratique et c'est celle qui a été adoptée dans le présent rapport.

Deux contraintes majeures conditionnent l'utilisation de générateurs stochastiques : (a) le flux de trafic à générer, qui décrit la quantité moyenne de véhicules à ajouter au trafic chaque minute. Nous spécifions également (b) la configuration de la liste L de véhicules qui est constituée des tuples de configuration et de la probabilité de chaque véhicule. On note cette liste $L = \{(\rho_1, V_1), (\rho_2, V_2), ...\}$ attribuant à tout véhicule donné V_i une probabilité ρ_i d'occurrence.

4

RÉSULTATS CONCLUSION

4.1 Résultats

Un objet clé pour l'élaboration des résultats est le Launcher (voir Listing 4.1). Il s'agit du lanceur de programme, un algorithme écrit comme objet de démarrage pour la simulation. Toutes les classes nécessaires sont importées. Un objet de classe de simulation est créé dans lequel les véhicules générés sont insérés. Les trajectoires des différents véhicules sont définies au préalable. L'exécution et l'affichage des attributs de la classe de fenêtre sont lancés. Tous les éléments de conception importants qui doivent être affichés sont configurés dans cette classe.

Listing 4.1 – Launcher

```
from window_ import *
from simulation_ import *
from vehicle_generator import VehicleGenerator

# Creer un object de type simulation
sim = Simulation()

vg = VehicleGenerator({
    'vehicles': [
        (10, {'path': [0, 8, 6], 'v': 16.6}),
        (10, {'path': [0, 12, 5], 'v': 16.6}),
        (10, {'path': [0, 16, 7], 'v': 16.6}) ,

        (10, {'path': [3,11 , 5], 'v': 16.6}),
        (10, {'path': [3, 15, 4], 'v': 16.6}),
        (10, {'path': [3, 19, 6], 'v': 16.6})
        ]
    })

sim.add_vehicle_generator(vg)
```

```
# Lancer la simultion
win = Window(sim)
win.run()
win.show()
```

Après exécution du programme, on observe une dynamique similaire à celle visible sur figure 4.1. Pour le trafic modélisé par des feux de signalisation, les véhicules générés aléatoirement sont alignés les uns après les autres en respectant les configurations générales (vitesse, distance, décélération, accélération) définies au préalable en fonction des zones. Dans la zone d'interaction (trafic biaisé), la distance est de $2m$ entre deux véhicules avec une décélération fixée à $1.40m/s^2$. On observe donc les véhicules sur la voie de gauche s'accumuler progressivement devant le feu rouge sans entrer en collision.

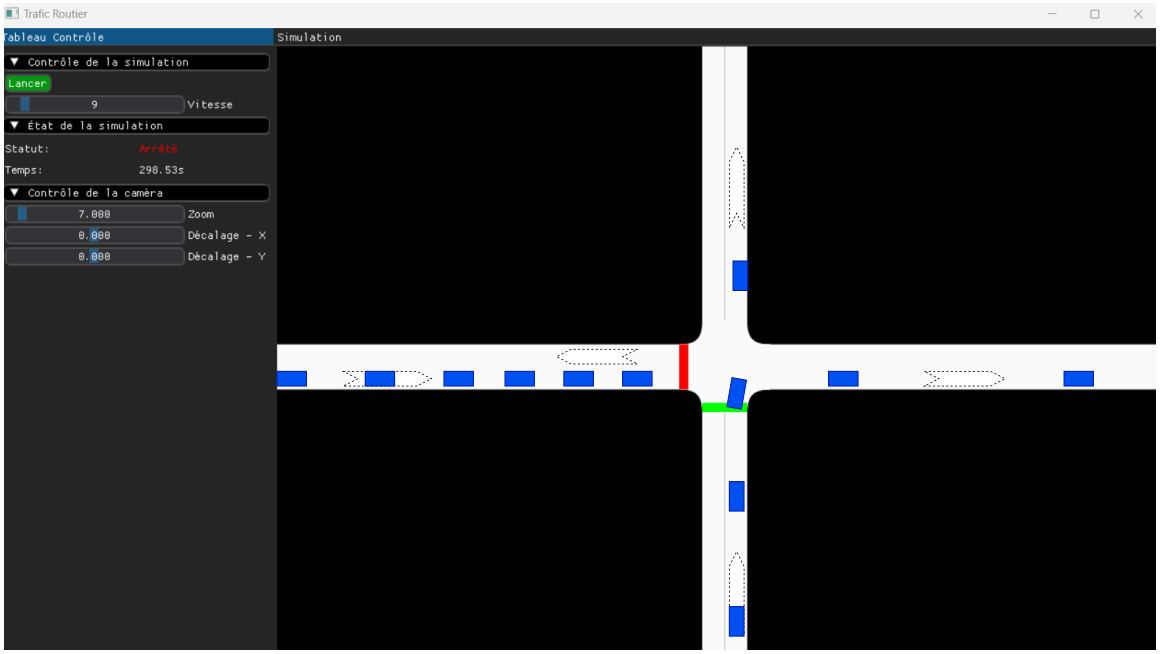

FIGURE 4.1 – Exemple de trafic final sur un simple carrefour.

Sur la voie verticale du schéma et lorsque le feu est vert, les véhicules se déplacent comme dans le cas d'une autoroute. Ici, un mouvement reste ininterrompu pour un mouvement de virage (virage, tout droit) jusqu'à ce que le feu passe au rouge. Dans ce dernier cas, les véhicules sur cette voie adoptent le même comportement (interaction) que les véhicules sur la voie horizontale jusqu'alors. Le feu de cette dernière passe alors au vert et laisse place à un flux de circulation homogène (accélération route libre).

4.2 Conclusion

Dans le présent projet, il était question d'expérimenter le trafic routier par le biais d'une modélisation microscopique. Grâce à l'élaboration d'équations différentielles décrivant la dynamique d'un véhicule du point de vue de sa vitesse, de sa position et de son accélération, nous avons établi un système qui réajuste la dynamique d'un ensemble de véhicules générés stochastiquement de manière progressive et adaptée. Le trafic a été modélisé sur la base d'un schéma de circulation à deux axes et deux feux, où une seule intersection a été prise en compte. Même si notre approche semble bien fonctionner dans ce cas, elle reste perfectible lorsqu'il s'agit de réseaux routiers plus complexes où d'autres facteurs tels que la densité des véhicules et les intersections à plus de quatre axes influenceraient considérablement la modélisation mathématique du système.

ANNEXE

Listing A.1 – Simulation Class

```
# Classe de simulation. Elle cr e un environnement de simulation (segments de
    route, v hicules, feux de signalisation, etc.) et lance la gestion (update) du
     trafic. Elle utilise la classe de g om trie pour un segment et la classe
    pour la g n ration d'objets de v hicules.

from vehicle_generator import VehicleGenerator
from geometry.segment import Segment
from vehicle import Vehicle


class Simulation:
    def __init__(self):
        self.segments = []
        self.signals = []
        self.signalsred = []
        self.vehicles = {}
        self.vehicle_generator = []

        self.t = 0.0
        self.frame_count = 0
        self.dt = 1/60
        self.roads = []

    def add_vehicle(self, veh):
        self.vehicles[veh.id] = veh
        if len(veh.path) > 0:
            self.segments[veh.path[0]].add_vehicle(veh)

    def add_segment(self, seg):
        self.segments.append(seg)
```

```
    def add_signal(self, seg):
        self.signals.append(seg)

    def add_signal_red(self, seg):
        self.signalsred.append(seg)


    def add_vehicle_generator(self, gen):
        self.vehicle_generator.append(gen)


    def create_vehicle(self, **kwargs):
        veh = Vehicle(kwargs)
        self.add_vehicle(veh)

    def create_segment(self, *args):
        seg = Segment(args)
        self.add_segment(seg)


    def create_signal(self, *args):
        seg = Segment(args)
        self.add_signal(seg)


    def create_signal_red(self, *args):
        seg = Segment(args)
        self.add_signal_red(seg)


    def create_quadratic_bezier_curve(self, start, control, end):
        cur = QuadraticCurve(start, control, end)
        self.add_segment(cur)

    def create_cubic_bezier_curve(self, start, control_1, control_2, end):
        cur = CubicCurve(start, control_1, control_2, end)
        self.add_segment(cur)

    def create_vehicle_generator(self, **kwargs):
```

```
        gen = VehicleGenerator(kwargs)
        self.add_vehicle_generator(gen)


    def run(self, steps):
        for _ in range(steps):
            self.update()

    def create_simpletrafic(self, sim, lane_space, intersection_size, length):
        # Carrefour
        ## Premier cas: Fragment d'entr e
        sim.create_segment((lane_space/2, length+intersection_size/2), (lane_space
            /2, intersection_size/2))
        sim.create_signal((-lane_space/2, 2*intersection_size/5), (-lane_space/2,
            intersection_size/2))
        sim.create_signal((lane_space/2, 2*intersection_size/5), (lane_space/2,
            intersection_size/2))

        sim.create_segment((length+intersection_size/2, -lane_space/2), (
            intersection_size/2, -lane_space/2))

        sim.create_segment((-lane_space/2, -length-intersection_size/2), (-
            lane_space/2, -intersection_size/2))
        sim.create_segment((-length-intersection_size/2, lane_space/2), (-
            intersection_size/2, lane_space/2))

        # sim.create_signal_red((lane_space/2, -2*intersection_size/5), (lane_space
            /2, -intersection_size/2))
        # sim.create_signal_red((-lane_space/2, -2*intersection_size/5), (-
            lane_space/2, -intersection_size/2))

        sim.create_signal_red((-2*intersection_size/5, lane_space/2), (-
            intersection_size/2, lane_space/2))
        sim.create_signal_red((-2*intersection_size/5, -lane_space/2), (-
            intersection_size/2, -lane_space/2))

        ## Second  cas: Fragment de sortie
        sim.create_segment((-lane_space/2, intersection_size/2), (-lane_space/2,
            length+intersection_size/2))
```

```
        sim.create_segment((intersection_size/2, lane_space/2), (length+
            intersection_size/2, lane_space/2))
        sim.create_segment((lane_space/2, -intersection_size/2), (lane_space/2, -
            length-intersection_size/2))
        sim.create_segment((-intersection_size/2, -lane_space/2), (-length-
            intersection_size/2, -lane_space/2))


        # Axes d'intersection
        sim.create_segment((lane_space/2, intersection_size/2), (lane_space/2, -
            intersection_size/2))
        sim.create_segment((intersection_size/2, -lane_space/2), (-
            intersection_size/2, -lane_space/2))
        sim.create_segment((-lane_space/2, -intersection_size/2), (-lane_space/2,
            intersection_size/2))
        sim.create_segment((-intersection_size/2, lane_space/2), (intersection_size
            /2, lane_space/2))

        # Virages
        sim.create_quadratic_bezier_curve((lane_space/2, intersection_size/2), (
            lane_space/2, lane_space/2), (intersection_size/2, lane_space/2))
        sim.create_quadratic_bezier_curve((intersection_size/2, -lane_space/2), (
            lane_space/2, -lane_space/2), (lane_space/2, -intersection_size/2))
        sim.create_quadratic_bezier_curve((-lane_space/2, -intersection_size/2), (-
            lane_space/2, -lane_space/2), (-intersection_size/2, -lane_space/2))
        sim.create_quadratic_bezier_curve((-intersection_size/2, lane_space/2), (-
            lane_space/2, lane_space/2), (-lane_space/2, intersection_size/2))

        sim.create_quadratic_bezier_curve((lane_space/2, intersection_size/2), (
            lane_space/2, -lane_space/2), (-intersection_size/2, -lane_space/2))
        sim.create_quadratic_bezier_curve((intersection_size/2, -lane_space/2), (-
            lane_space/2, -lane_space/2), (-lane_space/2, intersection_size/2))
        sim.create_quadratic_bezier_curve((-lane_space/2, -intersection_size/2), (-
            lane_space/2, lane_space/2), (intersection_size/2, lane_space/2))
        sim.create_quadratic_bezier_curve((-intersection_size/2, lane_space/2), (
            lane_space/2, lane_space/2), (lane_space/2, -intersection_size/2))

        sim.create_quadratic_bezier_curve((-lane_space/2, intersection_size/2), (-
            lane_space/2, lane_space/2), (-intersection_size/2, -lane_space/2))
```

```
        sim.create_quadratic_bezier_curve((intersection_size/2, lane_space/2), (
            lane_space/2, lane_space/2), (lane_space/2, intersection_size/2))
        sim.create_quadratic_bezier_curve((-lane_space/2, -intersection_size/2), (-
            lane_space/2, -lane_space/2), (intersection_size/2, -lane_space/2))
        sim.create_quadratic_bezier_curve((-intersection_size/2, lane_space/2), (
            lane_space/2, lane_space/2), (-lane_space/2, -intersection_size/2))

        return sim

    def update(self):
        # Actualiser vehicles
        for segment in self.segments:
            if len(segment.vehicles) != 0:
                self.vehicles[segment.vehicles[0]].update(None, self.dt)
            for i in range(1, len(segment.vehicles)):
                self.vehicles[segment.vehicles[i]].update(self.vehicles[segment.
                    vehicles[i-1]], self.dt)

        # Verifier la presence de vehicules hors limites sur les routes
        for segment in self.segments:
            if len(segment.vehicles) == 0: continue

            vehicle_id = segment.vehicles[0]
            vehicle = self.vehicles[vehicle_id]
            # Si le premier v hicule sort des limites de la route
            if vehicle.x >= segment.get_length():
                # Si le v hicule a une prochaine route
                if vehicle.current_road_index + 1 < len(vehicle.path):
                    vehicle.current_road_index += 1
                    next_road_index = vehicle.path[vehicle.current_road_index]
                    self.segments[next_road_index].vehicles.append(vehicle_id)
                vehicle.x = 0
                segment.vehicles.popleft()

        # Mise a jour des generateurs de vehicules
        for gen in self.vehicle_generator:
            gen.update(self)
        self.t += self.dt
        self.frame_count += 1
```

Listing A.2 – Windows Class

```
import dearpygui.dearpygui as dpg


class Window:
    def __init__(self, simulation):
        self.simulation = simulation

        self.zoom = 7
        self.offset = (0, 0)
        self.speed = 1

        self.is_running = False

        self.is_dragging = False
        self.old_offset = (0, 0)
        self.zoom_speed = 1

        self.setup()
        self.setup_themes()
        self.create_windows()
        self.create_handlers()
        self.resize_windows()
        self.signal_state = True

    def setup(self):
        dpg.create_context()
        dpg.create_viewport(title="Trafic Routier", width=1280, height=720)
        dpg.setup_dearpygui()

    def setup_themes(self):
        with dpg.theme() as global_theme:

            with dpg.theme_component(dpg.mvAll):
                dpg.add_theme_style(dpg.mvStyleVar_FrameRounding, 5, category=dpg.
                    mvThemeCat_Core)
                dpg.add_theme_style(dpg.mvStyleVar_FrameBorderSize, 1, category=dpg
                    .mvThemeCat_Core)
```

```
            dpg.add_theme_style(dpg.mvStyleVar_WindowBorderSize, 0, category=
                dpg.mvThemeCat_Core)
            # dpg.add_theme_style(dpg.mvStyleVar_ItemSpacing, (8, 6), category=
                dpg.mvThemeCat_Core)
            dpg.add_theme_color(dpg.mvThemeCol_Button, (0, 0, 0)) #(90, 90, 95)
            dpg.add_theme_color(dpg.mvThemeCol_Header, (0, 0, 0)) #(0, 91, 140)
        with dpg.theme_component(dpg.mvInputInt):
            dpg.add_theme_color(dpg.mvThemeCol_FrameBg, (0, 0, 0), category=dpg
                .mvThemeCat_Core)

    dpg.bind_theme(global_theme)


    with dpg.theme(tag="RunButtonTheme"):
        with dpg.theme_component(dpg.mvButton):
            dpg.add_theme_color(dpg.mvThemeCol_Button, (5, 150, 18))
            dpg.add_theme_color(dpg.mvThemeCol_ButtonHovered, (12, 207, 23))
            dpg.add_theme_color(dpg.mvThemeCol_ButtonActive, (2, 120, 10))

    with dpg.theme(tag="StopButtonTheme"):
        with dpg.theme_component(dpg.mvButton):
            dpg.add_theme_color(dpg.mvThemeCol_Button, (150, 5, 18))
            dpg.add_theme_color(dpg.mvThemeCol_ButtonHovered, (207, 12, 23))
            dpg.add_theme_color(dpg.mvThemeCol_ButtonActive, (120, 2, 10))


def create_windows(self):
    dpg.add_window(
        tag="MainWindow",
        label="Simulation",
        no_close=True,
        no_collapse=True,
        no_resize=True,
        no_move=True
    )

    dpg.add_draw_node(tag="OverlayCanvas", parent="MainWindow")
    dpg.add_draw_node(tag="Canvas", parent="MainWindow")

    with dpg.window(
```

```
            tag="ControlsWindow",
            label="Tableau Contrle",
            no_close=True,
            no_collapse=True,
            no_resize=True,
            no_move=True
        ):
            with dpg.collapsing_header(label="Contrle de la simulation",
                default_open=True):

                with dpg.group(horizontal=True):
                    dpg.add_button(label="Lancer", tag="RunStopButton", callback=
                        self.toggle)
                    #dpg.add_button(label="Next frame", callback=self.simulation.
                        update)

                dpg.add_slider_int(tag="SpeedInput", label="Vitesse", min_value=1,
                    max_value=100,default_value=1, callback=self.set_speed)

            with dpg.collapsing_header(label="tat de la simulation", default_open
                =True):

                with dpg.table(header_row=False):
                    dpg.add_table_column()
                    dpg.add_table_column()

                    with dpg.table_row():
                        dpg.add_text("Statut:")
                        dpg.add_text("_", tag="StatusText")

                    with dpg.table_row():
                        dpg.add_text("Temps:")
                        dpg.add_text("_s", tag="TimeStatus")

                    #with dpg.table_row():
                        #dpg.add_text("Cadre:")
                        #dpg.add_text("_", tag="FrameStatus")
```

```
        with dpg.collapsing_header(label="Contrle de la camra",
            default_open=True):

            dpg.add_slider_float(tag="ZoomSlider", label="Zoom", min_value=0.1,
                 max_value=100, default_value=self.zoom,callback=self.
                set_offset_zoom)
            with dpg.group():
                dpg.add_slider_float(tag="OffsetXSlider", label="Dcalage - X"
                    , min_value=-100, max_value=100, default_value=self.offset
                    [0], callback=self.set_offset_zoom)
                dpg.add_slider_float(tag="OffsetYSlider", label="Dcalage - Y"
                    , min_value=-100, max_value=100, default_value=self.offset
                    [1], callback=self.set_offset_zoom)

def resize_windows(self):
    width = dpg.get_viewport_width()
    height = dpg.get_viewport_height()

    dpg.set_item_width("ControlsWindow", 300)
    dpg.set_item_height("ControlsWindow", height-38)
    dpg.set_item_pos("ControlsWindow", (0, 0))

    dpg.set_item_width("MainWindow", width-315)
    dpg.set_item_height("MainWindow", height-38)
    dpg.set_item_pos("MainWindow", (300, 0))

def create_handlers(self):
    with dpg.handler_registry():
        dpg.add_mouse_down_handler(callback=self.mouse_down)
        dpg.add_mouse_drag_handler(callback=self.mouse_drag)
        dpg.add_mouse_release_handler(callback=self.mouse_release)
        dpg.add_mouse_wheel_handler(callback=self.mouse_wheel)
    dpg.set_viewport_resize_callback(self.resize_windows)

def update_panels(self):

    if self.is_running:
        dpg.set_value("StatusText", "En cours")
        dpg.configure_item("StatusText", color=(0, 255, 0))
    else:
```

```
            dpg.set_value("StatusText", "Arrêt")
            dpg.configure_item("StatusText", color=(255, 0, 0))

        dpg.set_value("TimeStatus", f"{self.simulation.t:.2f}s")




    def mouse_down(self):
        if not self.is_dragging:
            if dpg.is_item_hovered("MainWindow"):
                self.is_dragging = True
                self.old_offset = self.offset

    def mouse_drag(self, sender, app_data):
        if self.is_dragging:
            self.offset = (
                self.old_offset[0] + app_data[1]/self.zoom,
                self.old_offset[1] + app_data[2]/self.zoom
            )

    def mouse_release(self):
        self.is_dragging = False

    def mouse_wheel(self, sender, app_data):
        if dpg.is_item_hovered("MainWindow"):
            self.zoom_speed = 1 + 0.01*app_data

    def update_inertial_zoom(self, clip=0.005):
        if self.zoom_speed != 1:
            self.zoom *= self.zoom_speed
            self.zoom_speed = 1 + (self.zoom_speed - 1) / 1.05
        if abs(self.zoom_speed - 1) < clip:
            self.zoom_speed = 1

    def update_offset_zoom_slider(self):
        dpg.set_value("ZoomSlider", self.zoom)
        dpg.set_value("OffsetXSlider", self.offset[0])
        dpg.set_value("OffsetYSlider", self.offset[1])
```

```
    def set_offset_zoom(self):
        self.zoom = dpg.get_value("ZoomSlider")
        self.offset = (dpg.get_value("OffsetXSlider"), dpg.get_value("OffsetYSlider
            "))

    def set_speed(self):
        self.speed = dpg.get_value("SpeedInput")


    def to_screen(self, x, y):
        return (
            self.canvas_width/2 + (x + self.offset[0] ) * self.zoom,
            self.canvas_height/2 + (y + self.offset[1]) * self.zoom
        )

    def to_world(self, x, y):
        return (
            (x - self.canvas_width/2) / self.zoom - self.offset[0],
            (y - self.canvas_height/2) / self.zoom - self.offset[1]
        )

    @property
    def canvas_width(self):
        return dpg.get_item_width("MainWindow")

    @property
    def canvas_height(self):
        return dpg.get_item_height("MainWindow")


    def draw_bg(self, color=(0, 0, 0)):
        dpg.draw_rectangle(
            (-10, -10),
            (self.canvas_width+10, self.canvas_height+10),
            thickness=0,
            fill=color,
            parent="OverlayCanvas"
        )

    def draw_axes(self, opacity=80):
```

```
        x_center, y_center = self.to_screen(0, 0)

        dpg.draw_line(
            (-10, y_center),
            (self.canvas_width+10, y_center),
            thickness=2,
            color=(0, 0, 0, opacity),
            parent="OverlayCanvas"
        )
        dpg.draw_line(
            (x_center, -10),
            (x_center, self.canvas_height+10),
            thickness=2,
            color=(0, 0, 0, opacity),
            parent="OverlayCanvas"
        )

    def draw_grid(self, unit=10, opacity=50):
        x_start, y_start = self.to_world(0, 0)
        x_end, y_end = self.to_world(self.canvas_width, self.canvas_height)

        n_x = int(x_start / unit)
        n_y = int(y_start / unit)
        m_x = int(x_end / unit)+1
        m_y = int(y_end / unit)+1

        for i in range(n_x, m_x):
            dpg.draw_line(
                self.to_screen(unit*i, y_start - 10/self.zoom),
                self.to_screen(unit*i, y_end + 10/self.zoom),
                thickness=1,
                color=(0, 0, 0, opacity),
                parent="OverlayCanvas"
            )

        for i in range(n_y, m_y):
            dpg.draw_line(
                self.to_screen(x_start - 10/self.zoom, unit*i),
                self.to_screen(x_end + 10/self.zoom, unit*i),
                thickness=1,
```

```
            color=(0, 0, 0, opacity),
            parent="OverlayCanvas"
        )

    def draw_segments(self, color=(250, 250, 250)):
        for segment in self.simulation.segments:
            dpg.draw_polyline(segment.points, color=color, thickness=3.5*self.zoom,
                 parent="Canvas")

    def draw_signal(self,  color=(0, 255, 0)):
        for sig in self.simulation.signals:
            dpg.draw_polyline(sig.points, color=color, thickness=3.5*self.zoom,
                parent="Canvas")

    def draw_signal_red(self,  color=(255, 0, 0)):
        for sigr in self.simulation.signalsred:
            dpg.draw_polyline(sigr.points, color=color, thickness=3.5*self.zoom,
                parent="Canvas")



    def draw_vehicles(self):
        for segment in self.simulation.segments:
            for vehicle_id in segment.vehicles:
                vehicle = self.simulation.vehicles[vehicle_id]
                progress = vehicle.x / segment.get_length()

                position = segment.get_point(progress)
                heading = segment.get_heading(progress)

                node = dpg.add_draw_node(parent="Canvas")
                dpg.draw_line(
                    (0, 0),
                    (vehicle.l, 0),
                    thickness=1.76*self.zoom,
                    color=(0, 0, 255),
                    parent=node
                )

                translate = dpg.create_translation_matrix(position)
```

```
            rotate = dpg.create_rotation_matrix(heading, [0, 0, 1])
            dpg.apply_transform(node, translate*rotate)


    def apply_transformation(self):
        screen_center = dpg.create_translation_matrix([self.canvas_width/2, self.
            canvas_height/2, -0.01])
        translate = dpg.create_translation_matrix(self.offset)
        scale = dpg.create_scale_matrix([self.zoom, self.zoom])
        dpg.apply_transform("Canvas", screen_center*scale*translate)


    def render_loop(self):
        self.update_inertial_zoom()
        self.update_offset_zoom_slider()

        dpg.delete_item("OverlayCanvas", children_only=True)
        dpg.delete_item("Canvas", children_only=True)

        self.draw_bg()
        self.draw_axes()
        self.draw_grid(unit=10)
        self.draw_grid(unit=50)
        self.draw_segments()
        self.draw_vehicles()
        self.draw_signal(color=(0, 255, 0))
        self.draw_signal_red(color=(255, 0, 0))


        self.apply_transformation()

        self.update_panels()

        # Actualiser la simulation
        if self.is_running:
            self.simulation.run(self.speed)

    def show(self):
        dpg.show_viewport()
        while dpg.is_dearpygui_running():
```

```
            self.render_loop()
            dpg.render_dearpygui_frame()
        dpg.destroy_context()

    def run(self):
        self.is_running = True
        dpg.set_item_label("RunStopButton", "Arrêter")
        dpg.bind_item_theme("RunStopButton", "StopButtonTheme")

    def stop(self):
        self.is_running = False
        dpg.set_item_label("RunStopButton", "Lancer")
        dpg.bind_item_theme("RunStopButton", "RunButtonTheme")

    def toggle(self):
        if self.is_running: self.stop()
        else: self.run()
```

Listing A.3 – Road Geometry - Segment Class

```
# Modeliser des segments de routes

from scipy.spatial import distance
from scipy.interpolate import interp1d
from collections import deque
from numpy import arctan2, unwrap, linspace

class Segment:
    def __init__(self, points):
        self.points = points
        self.vehicles = deque()

        self.set_functions()


    def set_functions(self):
        # Point
        self.get_point = interp1d(linspace(0, 1, len(self.points)), self.points,
            axis=0)
```

```
        headings = unwrap([arctan2(
            self.points[i+1][1] - self.points[i][1],
            self.points[i+1][0] - self.points[i][0]
        ) for i in range(len(self.points)-1)])
        if len(headings) == 1:
            self.get_heading = lambda x: headings[0]
        else:
            self.get_heading = interp1d(linspace(0, 1, len(self.points)-1),
                headings, axis=0)

    def get_length(self):
        length = 0
        for i in range(len(self.points) -1):
            length += distance.euclidean(self.points[i], self.points[i+1])
        return length

    def add_vehicle(self, veh):
        self.vehicles.append(veh.id)

    def remove_vehicle(self, veh):
        self.vehicles.remove(veh.id)
```

Listing A.4 – Vehicle generator Class

```
from vehicle import Vehicle
from numpy.random import randint

# Generer un vehicule de maniere stochastique

class VehicleGenerator:
    def __init__(self, config={}):

        self.set_default_config()


        for attr, val in config.items():
            setattr(self, attr, val)

        self.init_properties()
```

```
    def set_default_config(self):

        self.vehicle_rate = 10
        self.vehicles = [
            (1, {})
        ]
        self.last_added_time = 0

    def init_properties(self):
        self.upcoming_vehicle = self.generate_vehicle()

    def generate_vehicle(self):
        # Renvoie un vehicule aleatoire de self.vehicles avec des proportions
            aleatoires
        total = sum(pair[0] for pair in self.vehicles)
        r = randint(1, total+1)
        for (weight, config) in self.vehicles:
            r -= weight
            if r <= 0:
                return Vehicle(config)

    def update(self, simulation):
        if simulation.t - self.last_added_time >= 60 / self.vehicle_rate:
            print('adding vehicle')
            segment = simulation.segments[self.upcoming_vehicle.path[0]]
            if len(segment.vehicles) == 0\
               or simulation.vehicles[segment.vehicles[-1]].x > self.
                   upcoming_vehicle.s0 + self.upcoming_vehicle.l:
                simulation.add_vehicle(self.upcoming_vehicle)
                self.last_added_time = simulation.t
            self.upcoming_vehicle = self.generate_vehicle()
```

Listing A.5 – Vehicle Class

```
import uuid
import numpy as np

class Vehicle:
    def __init__(self, config={}):
        self.set_default_config()
```

```
        for attr, val in config.items():
            setattr(self, attr, val)

        self.init_properties()

    def set_default_config(self):
        self.id = uuid.uuid4()

        self.l = 4
        self.s0 = 4
        self.T = 1
        self.v_max = 20
        self.a_max = 1.44
        self.b_max = 4.61

        self.path = []
        self.current_road_index = 0

        self.x = 0
        self.v = 0
        self.a = 0
        self.stopped = False

    def init_properties(self):
        self.sqrt_ab = np.sqrt(2*self.a_max*self.b_max)
        self._v_max = self.v_max

    def update(self, lead, dt):
        # Actualiser position et vitesse
        if self.v + self.a*dt < 0:
            self.x -= 1/2*self.v*self.v/self.a
            self.v = 0
        else:
            self.v += self.a*dt
            self.x += self.v*dt + self.a*dt*dt/2

        # Update l'acceleration
        alpha = 0
```

```
        if lead:
            delta_x = lead.x - self.x - lead.l
            delta_v = self.v - lead.v

            alpha = (self.s0 + max(0, self.T*self.v + delta_v*self.v/self.sqrt_ab))
                 / delta_x

        self.a = self.a_max * (1-(self.v/self.v_max)**4 - alpha**2)

        if self.stopped:
            self.a = -self.b_max*self.v/self.v_max
```

BIBLIOGRAPHIE

[BDSDM00] Tom Bellemans, Bart De Schutter et Bart De Moor. « On data acquisition, modeling and simulation of highway traffic ». In : *Proceedings of the 9th International Federation of Automatic Control (IFAC) Symposium on Transportation Systems*. Citeseer. 2000, p. 22-27.

[RZQ99] Yuhe Ren, Bo Zhang et Hong Qiao. « A simple Taylor-series expansion method for a class of second kind integral equations ». In : *Journal of Computational and Applied Mathematics* 110.1 (1999), p. 15-24.

[Tan+21] Jiyuan Tan, Qianqian Qiu, Weiwei Guo et Tingshuai Li. « Research on the construction of a knowledge graph and knowledge reasoning model in the field of urban traffic ». In : *Sustainability* 13.6 (2021), p. 3191.

[THH00] Martin Treiber, Ansgar Hennecke et Dirk Helbing. « Congested traffic states in empirical observations and microscopic simulations ». In : *Physical Review E* 62.2 (2000), p. 1805-1824. doi : 10.1103/physreve.62.1805. url : https://doi.org/10.1103%2Fphysreve.62.1805.