UCLouvain

École polytechnique de Louvain

Synthèse d'image et modélisation physique pour une simulation de trafic urbain sous Unreal Engine

Auteurs : **Luca Mellini, Nicolas Rosar**
Promoteur : **Benoît Macq**
Lecteurs : **Christophe Craeye, Lucas El Raghibi, Jonathan Samelson**
Année académique 2021–2022
Master [60] en sciences informatiques

Avant-propos

Le fruit de ce travail résulte de la collaboration entre différents acteurs, ayant tous apporté une contribution notable.

Tout d'abord, nous souhaitons adresser nos remerciements à notre directeur de mémoire Benoît Macq. Grâce à lui, nous avons eu l'opportunité de travailler sur un sujet présentant une évolution à laquelle nous sommes ravis d'avoir pu contribuer.

Ensuite, nous remercions les encadrants de l'équipe en charge de Digital Twin pour leur disponibilité et les conseils avisés qu'ils nous ont apporté tout au long de ce travail. Nos remerciements s'adressent à Jonathan Samelson, Victorien Sonneville, Lucas El Raghibi et Nikita De Broux. Leurs critiques constructives nous ont sans cesse permis d'améliorer nos capacités de réflexion, ce qui a grandement contribué à la production de cette thèse.

Nous remercions également l'équipe en charge de Robotran pour leur collaboration ainsi que leur réactivité. Nos remerciements s'adressent tout particulièrement à Nicolas Docquier qui nous a servi de point de contact. Nos discussions enrichissantes ont pu apporter de la valeur à notre travail.

Merci à notre jury, composé de Christophe Craeye, Benoît Macq, Jonathan Samelson et Lucas El Raghibi pour la lecture de notre mémoire.

Nous tenons également à adresser nos remerciements à notre ami Egan Lecoq. Son savoir-faire et son professionnalisme nous a permis d'aborder certains concepts sous un angle meilleur.

Travailler sur une thèse requiert plus qu'une assistance académique. Ainsi, nous souhaitons remercier nos proches respectifs pour le soutien perpétuel qu'ils nous ont apporté tout au long de cette année.

Table des matières

Liste des notations et acronymes

PiLAB	Pixels and Interactions Lab
UE	Unreal Engine
3D	Trois Dimensions
BVS	Blueprint Visual Scripting
IA	Intelligence Artificielle
LFS	Large File Storage
VPN	Virtual Private Network
API	Application Programming Interface
LOD	Level Of Details
FPS	Frames Per Second
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
JSON	JavaScript Object Notation

Table des figures

1 Introduction

De nos jours, il est difficile de calibrer des systèmes de gestion de trafic à cause du besoin massif de données nécessaires pour entrainer ces systèmes. Dès que nous souhaitons récupérer des données dans une localisation peuplée, nous faisons face à un problème majeur qui est le respect de la vie privée. Peu de gens sont d'accord d'être filmés et ne veulent pas que leur image soit utilisée à des fins commerciales.

L'équipe de recherche PiLAB est consciente de cette problématique et s'est lancée dans l'étude de solutions innovantes et non invasives. C'est ainsi qu'elle a mis en place un projet visant à reproduire des scènes de trafic routier sous forme de simulation vidéo-ludique.

En effet, les moteurs de jeu actuels permettent de reproduire un degré de réalisme qui se rapproche fortement de la réalité. Grâce à ce genre d'outil, il est désormais possible de recréer n'importe quel environnement et situation. Par conséquent, les contraintes concernant la génération de données et le respect de la vie privée en sont amoindries.

Ainsi, le projet Digital Twin proposé par PiLAB servira de base au travail présenté dans ce document. Au cours de l'avancée de cette thèse, les objectifs et possibilités de développement ont fortement évolué. Cependant, une question principale persiste : « De quelle manière pouvons-nous améliorer une simulation afin de la rendre plus réaliste ? ». Parmi toutes les pistes envisageables, nous avons concentré notre travail sur la synthèse d'images et la modélisation physique d'une scène de trafic urbain.

Ainsi, notre thèse comporte deux principaux partis.

Dans un premier temps, le projet Digital Twin qui constitue le travail effectué lors du premier quadrimestre. Il nous a permis de nous familiariser avec l'environnement de développement proposé le moteur de jeu que nous avons utilisé tout au long de ce projet.

Dans un second temps, le projet d'intégration de la solution Robotran effectué lors du second quadrimestre. Il nous a permis d'apporter un enrichissement conséquent à notre travail d'un point de vue physique mécanique.

Bien que ces deux partis présentent des différences, étant donné la façon dont nous les avons abordés, ils permettent finalement d'obtenir un résultat homogène.

Ce résultat constitue une première étape ayant pour but de vérifier la faisabilité du projet dans sa globalité. De futures itérations seront nécessaires à son développement afin qu'il soit utilisable comme outil complémentaire à d'autres travaux.

2 Contexte général

Cette section permet de présenter les deux principaux partis de notre mémoire. Tout d'abord, nous y présentons une analyse descriptive de ceux-ci. Ensuite, Nous y traitons les enjeux qui les constituent, en expliquant notamment les attentes vis-à-vis des deux partis en question.

2.1 Digital Twin

2.1.1 Description

Digital Twin (dont le nom est tiré du concept de « jumeau numérique » [1]) est un projet de développement de système de surveillance d'un carrefour routier. Il a été débuté en décembre 2020 par François-Xavier Inglese, un ancien chercheur de l'équipe de recherche PiLAB (Pixels and Interactions Lab), dirigée par le professeur Benoît Macq et faisant partie de l'ICTEAM (Institut des Technologies de l'Information et de la Communication, de l'Électronique et des Mathématiques Appliquées) à l'UCLouvain (Université Catholique de Louvain).

Ce projet est développé avec l'outil Unreal Engine (UE) [2]. Il s'agit d'un moteur de jeu développé en 1988 par l'entreprise Epic Games. Au travers des années, différentes versions du logiciel ont émergé [3]. Malgré une nouvelle version apparue officiellement en 2022 et une possibilité de migration entre les versions, ce projet sera poursuivi avec la quatrième version du logiciel dans le cadre de notre travail.

Un moteur de jeu est un logiciel qui facilite le développement de jeux vidéo et de leur interface en fournissant des outils complets pour exécuter le jeu durant son développement, gérer les collisions des objets, intégrer ses propres modèles en trois dimensions (3D), etc. De nombreux aspects du développement de jeux vidéo sont donc pris en charge par UE afin de permettre aux développeurs de se consacrer au cœur de leur jeu.

Un autre concept très important appelé Blueprints Visual Scripting (BVS [4]) permet de programmer à l'aide d'une représentation visuelle.

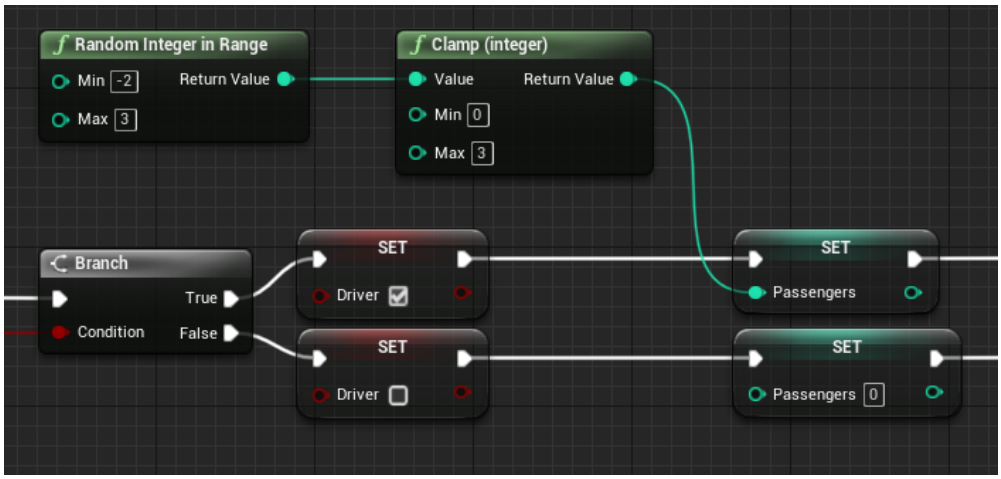


Figure 1 - Exemple de code réalisé en Visual Scripting

Le BVS est un système permettant de réaliser des scripts en utilisant des nœuds. Ceux-ci sont des blocs de code qui sont connectés les uns aux autres pour réaliser une séquence d'actions (calculs, conditions, comparaisons, ajout d'acteurs, etc.). Prenons comme exemple le morceau de code suivant :

```
if (condition) {
    // block of code to execute if condition is true
}
else {
    // block of code to execute if condition if false
}
```

Figure 2 - Exemple de code écrit

Celui-ci va simplement exécuter le code dans la première paire d'accolades si la condition est vraie ou bien le code dans la seconde paire si la condition est fausse.

Voici le même code réalisé avec le BVS :

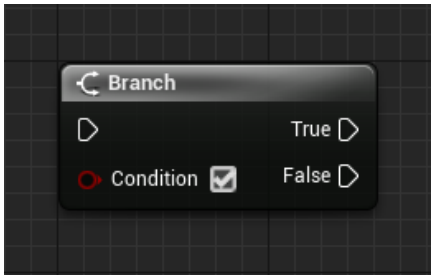


Figure 3 - Exemple de code sous forme de nœud en BVS

Ce système de programmation permet donc de s'abstraire de la programmation « écrite » plus complexe mais engendre un coût supplémentaire invisible aux yeux de l'utilisateur. Il est aussi question de limitations dans la marche de manœuvre comparé à du code en C++ pur. De nombreux autres concepts mériteraient d'être mentionnés concernant le BVS mais l'exemple précédent est suffisant pour la suite de notre analyse.

Tous ces concepts permettent donc à des personnes, qui ne possèderaient pas forcément de compétences poussées dans le développement, de créer des jeux vidéo aisément.

En plus de pouvoir créer des jeux vidéo, UE peut être utilisé à d'autres fins. Dans le cas de Digital Twin, nous allons utiliser celui-ci comme simulateur de trafic routier.

2.1.2 Problématique

Dans le contexte de l'automobile et du trafic routier, il existe une multitude de configurations de routes avec des vitesses imposées différentes, des conditions météo différentes, des acteurs différents selon le moment de la journée, etc. De plus, pour obtenir des données, il est difficilement concevable de mettre en scène des accidents de voiture mettant en péril la vie des conducteurs et usagés de la route. À cela s'ajoute la question de la vie privée concernant les personnes et leur droit à l'image.

Un autre point important concerne l'annotation des données [5]. Par exemple, pour une image de caméra de sécurité à un instant précis, il est nécessaire d'ajouter des informations afin d'identifier les différents acteurs présents. Ce processus requiert un temps conséquent s'il n'est pas automatisé. C'est pour répondre à cette problématique que la simulation entre en jeu. Toute situation peut en théorie être simulée de manière artificielle en éliminant les contraintes précédemment citées.

La finalité du projet consiste donc à fournir des images réalistes de scènes urbaines (accompagnées de ses métadonnées) par l'intermédiaire d'une simulation. Pour ce faire, nous avons exploré différents points sur lesquels il est important de se pencher pour rendre un projet UE le plus réaliste et optimisé possible. Ces points seront discutés plus en profondeur dans les prochaines sections de ce manuscrit.

Nous remarquons la portée de ce projet en s'attardant sur les différentes possibilités d'utilisation de ces données ou de la simulation elle-même :

- Données d'entrainement pour des Intelligences Artificielles (IA) de véhicule autonome ou de caméras de surveillance.
- Simulateur de conduite de véhicule
- Analyse de la dangerosité d'une configuration routière ou de sa configuration adéquate en vue de chantiers.
- Calibration de systèmes de gestion de trafic.

2.2 Robotran

2.2.1 Description

Robotran [6] est un logiciel libre permettant de simuler et d'analyser des systèmes multicorps [7] à l'aide d'équations générées symboliquement. Il permet de traiter des applications industrielles et de recherche concrètes. Parmi celles-ci, il y a l'analyse du confort et de la stabilité des véhicules ferroviaires, l'optimisation de la configuration des suspensions de véhicules pendulaires ou la simulation en temps réel de l'action d'un piano haptique. Robotran est également utilisé dans le cadre de l'apprentissage scolaire et de la recherche académique.

Cet outil est actuellement développé par la division MEED (Multibody Research Group of the Mechatronics, Electrical Energy and Dynamic Systems) qui fait partie de l'iMMC (Institut de mécanique, des matériaux et du génie civil) de l'EPL (École Polytechnique de Louvain).

Robotran est disponible dans plusieurs langages de programmation (Matlab, C/C++ et Python) et est utilisable sur plusieurs plateformes (Windows, MacOS et Linux). Ce logiciel de recherche possède une interface graphique permettant d'avoir une animation 3D de simulation en temps réel et peut également être interfacée avec d'autres environnements de développement.

2.2.2 Problématique

Comme expliqué précédemment, la finalité du projet est d'obtenir une simulation de trafic routier via un moteur de jeu afin de pouvoir en récolter des données réalistes. Pour ce faire, il est important que le comportement des véhicules soit le plus proche possible de celui retrouvé dans la réalité. Bien que les véhicules présents dans le projet possèdent déjà une conduite permettant de reproduire des scènes réalistes, nous avons pris conscience du fait qu'il est possible de les améliorer considérablement.

C'est pour répondre à ce besoin qu'une étroite collaboration avec l'équipe en charge de Robotran a pris forme.

Cette collaboration bilatérale permet, dans le cadre de ce travail, de nous munir d'une physique sur-mesure pour améliorer le comportement des véhicules de notre simulation. Cependant, elle pourra également être mise au profit de Robotran dans de futures collaborations. En effet, Robotran possède un simulateur de conduite permettant de visualiser leur travail d'un point de vue mécanique. Ce simulateur possède une scène graphique suffisante à leur activité mais n'est pas aussi réaliste qu'une scène développée par un véritable moteur de jeu.

Il pourrait donc être envisagé de leur faire bénéficier de notre expertise dans le développement de scène réaliste dans de futures itérations.

Il est important de souligner que notre travail d'intégration de Robotran au sein de Digital Twin relève du prototypage et permet dans un premier temps d'évaluer la faisabilité de cette intégration. Un aspect important sera donc l'exploration des différentes pistes qui pourraient être intéressantes pour de futurs travaux visant à approfondir cette collaboration avec Robotran.

3 Solution proposée

Ce mémoire comporte deux partis principaux.

Le premier, Digital Twin, consiste en un projet de développement d'une simulation de trafic routier avec le moteur de jeu UE. Il permettra de répondre à une demande concernant la récolte de données sur base d'une scène de jeu que nous avons essayé de rendre la plus réaliste et la plus optimisée possible. Ceci grâce à l'étude de certaines métriques propres à l'environnement UE.

Le second, Robotran, consiste en un logiciel libre permettant de simuler et d'analyser des systèmes multicorps. La collaboration avec l'équipe en charge de ce logiciel répondra à une demande concernant l'amélioration de la physique des véhicules présents dans une scène de simulation préalablement créée. Il sera également important d'explorer les différentes pistes pouvant amener à une intégration plus approfondie de Robotran au sein du prototype qu'il nous sera donné de présenter.

Cette section explore en profondeur ces deux principaux partis, précédemment introduits.

Il traitera dans un premier temps de la méthodologie de travail mise en place tout au long de l'année. Il sera question des outils primordiaux utilisés en termes de communication, versioning et gestion des tâches à effectuer, ainsi que de l'organisation au sein du groupe, ce travail ayant été réalisé en binôme et encadré par plusieurs assistants.

Ensuite, le travail autour du projet Digital Twin, ayant occupé le premier quadrimestre de l'année académique. Nous expliquerons notre première approche du projet, ainsi que les concepts constituant les éléments de performance et de réalisme que nous avons eu l'occasion d'aborder.

Enfin, le travail concernant Robotran, celui-ci ayant occupé le second quadrimestre de l'année académique. Nous y détaillerons les interactions théoriques qu'il existe entre Robotran et Digital Twin, mais aussi différents concepts plus pratiques concernant son intégration, notamment les sockets et l'orientation d'un objet dans l'espace. Nous expliquerons ensuite le cas d'étude mis en place dans le but d'observer les résultats de cette intégration et terminerons par une analyse approfondie de ces résultats sous différents aspects.

3.1 Méthodologie de travail

Ayant réalisé ce mémoire en binôme, certains aspects ont dû être réfléchis afin d'organiser le travail de la meilleure manière possible. Le premier aspect concerne le versioning, à savoir comment accéder de façon concurrente aux différentes parties du projet. Le second aspect concerne la manière dont nous avons interagis au sein de notre groupe, mais aussi le partage de nos avancées aux équipes en charge des projets Digital Twin et Robotran. Enfin, il est important de présenter les concepts mis en place afin d'assurer une bonne gestion de projet. Cela inclut notamment certains outils de gestion de tâches, qui seront présentés ultérieurement.

Cette section introduira certaines pratiques présentes dans la méthodologie de travail dite « agile » que nous avons adoptées pour arriver à nos fins. Parmi elles, il y a notamment le « pair programming », l'utilisation d'outil d'aide à la gestion de projets, les réunions à intervalles réguliers, etc.

3.1.1 Versioning

Le concept de versioning est un aspect important dans tout travail. Cela consiste à gérer les différentes versions du projet et à accéder aisément aux éléments qui le constituent. Le versioning sert également à centraliser et converger le travail vers un seul et même point, ce qui rend son partage plus facile et permet ainsi de collaborer de manière plus efficace.

3.1.1.1 Digital Twin

Lorsqu'il s'agit de versioning [8], les premières pensées se dirigent vers l'utilisation de Github [9], étant l'un des outils de versioning les plus populaires. Cependant, il présente certaines lacunes. L'une d'elles étant que les fichiers pouvant y être stockés ont une taille individuelle limitée. En effet, pour pouvoir ajouter un fichier de plus de 100 mégabytes, il faut utiliser l'outil Git Large File Storage (LFS [10]), ce qui rend l'utilisation de Github bien moins pratique lorsqu'un grand nombre de ces fichiers est présent.

Le projet Digital Twin regroupant un nombre important de fichiers lourds (notamment les fichiers de maps, textures, etc.), il nous a fallu trouver une meilleure alternative. C'est ainsi qu'après en avoir discuté avec l'équipe en charge du projet Digital Twin, nous avons pu bénéficier d'un accès direct à « Helix Visual Client » (P4V) [11]. Il s'agit d'une application, développée par Perforce, permettant d'accéder aux fichiers versionnés via une interface graphique. Le projet étant initialement hébergé sur cette application, il nous était finalement aisé de centraliser et versionner notre travail. Cependant, un léger inconvénient avec cet outil réside dans le fait qu'accéder au serveur nécessite un Virtual Private Network (VPN) qui n'autorise pas l'accès à internet simultanément (site web, outils de communication, etc.).

3.1.1.2 Robotran

Dans un premier temps, le projet développé par Robotran nous a été délivré sur Gitlab [12], une alternative à Github présentant certaines différences [13] mais dont l'utilisation est relativement similaire. N'ayant qu'un accès en lecture, nous ne pouvions cependant pas y ajouter les modifications que nous devions apporter au projet.

Cela étant, le projet Robotran ne dispose pas de fichiers aussi volumineux que ceux présents au sein du projet Digital Twin. La solution d'utiliser Github était donc, dans ce cas, envisageable. C'est pourquoi nous l'avons optée afin de gérer le projet Robotran tout au long de la phase de développement et de tests.

3.1.2 Communication

La communication est un aspect primordial lorsqu'il s'agit d'un travail impliquant plusieurs personnes. Cela ne concerne pas seulement la communication au sein du binôme. En effet, nous trouvons important le fait de communiquer notre avancement, sous forme de compte rendu, aux équipes en charge des projets Digital Twin et Robotran. Cela permet essentiellement de pouvoir donner une idée claire du travail réalisé et du travail à réaliser pour le prochain compte rendu. Cela nous permet également de pouvoir planifier nos tâches plus efficacement.

Ainsi, tout au long de notre travail, nous avons organisé des réunions avec nos encadrants et équipe en charge de Digital Twin afin de leur présenter, sur base de supports visuels, notre état d'avancement ainsi que le travail à réaliser pour la prochaine réunion. La fréquence de ces réunions était bimensuelle mais il est important de noter que nous étions tout de même régulièrement en contact avec l'équipe en question.

Ensuite, nous avons procédé à des réunions journalières en binôme afin de progresser de façon régulière dans les différentes phases de développement. Cela nous a permis de ne jamais perdre le fil de progression et de toujours garder une vision claire des tâches à réaliser.

Enfin, lorsque nous avons été amenés à travailler en collaboration avec l'équipe en charge de Robotran, nous avons jugé nécessaire de les tenir informés quant à notre état d'avancement. Pour ce faire, nous les avons contactés à une fréquence bimensuelle afin de leur faire part du travail effectué, des prochaines étapes de développement, des éventuelles difficultés rencontrées, etc. Cela a permis de garder un contact constant, ce qui participe à l'avancée du projet dans un cadre sain.

3.1.3 Gestion de projet

Comme énoncé précédemment, nous avons donc adopté certains concepts des méthodologies dites « agiles » [14] en ce qui concerne la gestion du projet du point de vue des tâches à réaliser.

Tout d'abord, nous avons utilisé Jira [15], un logiciel de suivi de projet développé par Atlassian. Cet outil nous a permis, dans un premier temps, de pouvoir définir les différentes tâches à accomplir. En effet, réussir à décrire les différentes phases de développement n'est pas toujours chose aisée mais permet de pouvoir les comprendre plus facilement. Ensuite, Jira possède une interface permettant de créer différentes tâches et de les classer à notre convenance. Dans notre projet, nous avons défini des sections qui correspondent aux différentes phases de développement, à savoir « tâches à faire », « tâches en cours » et « tâches terminées ». De même, il est possible d'attribuer ces différentes tâches à une personne spécifique, ce qui nous a permis de diviser le travail de manière optimale au sein de l'équipe de développement et d'avoir continuellement une vision claire de l'état d'avancement global.

Ensuite, nous avons adopté un concept tiré de « l'extreme programming » [16], une méthode agile dont l'objectif est de produire une application de la meilleure qualité possible, à savoir la programmation en binôme, plus connu sous son appellation anglaise « pair programming » [17]. Il s'agit d'une pratique dans laquelle les membres d'un binôme travaillent simultanément sur le même poste de travail et réfléchissent ensemble à une problématique (ou autre élément) à développer. Ce concept se prête bien à notre situation, où la division des tâches au sein du binôme n'est pas toujours évidente. En effet, le « pair programming » nous permet d'atteindre une meilleure efficacité.

3.2 Digital Twin

Dans cette section, nous aborderons la première partie de notre travail de fin d'étude dans laquelle nous décrirons nos différentes approches du projet Digital Twin.

Les notions abordées dans cette section nous ont principalement permis de prendre en main l'outil de développement UE. Ceci afin de nous familiariser avec cet environnement pour les tâches plus importantes qui seront présentées dans les prochaines sections.

3.2.1 Première approche

3.2.1.1 Carla UE4

La première piste que nous avons décidé d'explorer se porte sur le simulateur Carla [18] qui, de même que le projet Digital Twin, a été développé avec UE sous sa quatrième version. Il s'agit d'un logiciel open-source de simulation de trafic routier dans divers environnements tels qu'une ville avec des piétons ainsi qu'une météo changeante, etc. Son but premier étant d'aider les développeurs à créer des systèmes de conduite autonome dans des scénarios les plus variés et flexibles possibles.

3.2.1.2 Contenu

Comme précisé précédemment, Carla est très flexible dans sa façon de fonctionner. Cette section présente certains points qui composent le logiciel.

Environnement personnalisé

Chaque utilisateur de Carla doit pouvoir créer les situations qui conviennent à ses besoins. Un environnement peut par exemple être urbain avec une structure de routes sous forme de quadrillage type Etats-Unis ou bien plus hétéroclite avec des ronds-points, des impasses, etc.

Reproduire ces divers contextes peut se faire par l'intermédiaire de RoadRunner [19]. Il s'agit d'un logiciel (externe à Carla) permettant de modéliser des scènes en 3D spécialement conçues pour la simulation et les tests de systèmes de conduite autonome. Il permet de prendre en compte les signaux routiers spécifiques à des régions, les feux de signalisation, insérer des panneaux, des glissières de sécurité, des dégâts sur la route, sur les bâtiments, etc. Une libraire contenant des centaines de modèles est disponible afin de fournir un résultat complet, professionnel et consistant.

Par la suite, un utilisateur peut transformer sa carte personnalisée en un package et l'importer dans Carla pour effectuer ses simulations.

Suite de capteurs de conduite autonome

De multiples capteurs sont configurables [20]. Il y a entre autres les Light Detection and Ranging (LIDAR [21]), caméras [22], capteurs de profondeur [23], radars [24], etc. Parmi ces derniers, les caméras sont les composantes les plus profitables dans le cadre du projet Digital Twin afin d'obtenir des images réalistes de la simulation.

API flexible

Le paramétrage est facilité par une Application Programming Interface (API) pouvant être utilisée en vue de contrôler la simulation. Des paramètres tels que la génération du trafic (le nombre de véhicules par exemple), le comportement des piétons, la météo, les capteurs, etc. peuvent être manipulés grâce à cette API.

Simulation de scénarios de trafic

Une partie importante du sujet de recherche consiste à pouvoir générer des situations spécifiques pour en tirer des données. Il est donc primordial de pouvoir concevoir des cas ad hoc. Cette tâche, est réalisée séparément de Carla avec l'outils Scenario Runner [25]. Carla possède actuellement une liste de treize scénarios supportés. Grâce à ces scénarios de base, un utilisateur peut vérifier le bon fonctionnement de son véhicule autonome dans différents cas.

Base pour la conduite autonome

Les véhicules impliqués dans les simulations Carla sont des IA appelées agent. Des agents avec des comportements « classiques » sont fournis par défaut par le logiciel. Ainsi, un utilisateur peut créer son propre agent directement dans Carla sans devoir créer tout le nécessaire en partant de zéro.

3.2.1.3 Licence

Le projet Carla utilise une licence MIT [26] (Massachusetts Institute of Technology). Cette licence datant des années 1980 fait partie des plus populaires en raison de sa flexibilité. Elle donne le droit à toute personne d'utiliser gratuitement le logiciel ainsi que la documentation associée mais également d'y apporter des modifications, de le fusionner avec d'autres projets, de le publier, le distribuer, de sous-licencier le projet et/ou vendre des copies du logiciel. Les personnes fournissant le logiciel peuvent également transmettre ces droits aux receveurs. Ce type de licence permet donc la réutilisation du contenu de Carla dans un logiciel propriétaire tel que Digital Twin sans l'impacter de quelque manière que ce soit.

3.2.1.4 Difficultés rencontrées

Une première difficulté rencontrée avec cette solution concerne le matériel nécessaire pour faire fonctionner le projet. Celui-ci se trouve être très restrictif car il nécessite une machine avec une configuration relativement puissante qu'il n'est pas commun de retrouver dans les ordinateurs classiques (machines à usage bureautique par exemple).

Une seconde difficulté concerne la complexité de l'architecture de Carla. Il est nécessaire de maitriser Carla dans une certaine mesure avant de pouvoir y greffer son propre projet. De ce fait, Carla représente une porte ouverte par laquelle il serait possible d'intégrer un autre projet dans le futur. Celui-ci ne serait intégré qu'après avoir été réalisé et testé dans un environnement plus léger afin d'échapper aux contraintes liées au simulateur Carla.

C'est pour ces raisons que nous nous sommes pleinement tournés vers le projet Digital Twin afin de réaliser nos tests.

3.2.2 Performances

Les performances dans un jeu vidéo sont un aspect essentiel et nécessitent une attention particulière de la part du programmeur. En effet, c'est un élément qui impacte directement l'expérience utilisateur et que nous avons trouvé important d'aborder dans la première partie de notre travail.

Cependant, les optimisations de performances ne constituent pas l'objectif principal du projet. Nous n'avons donc pas exploré en profondeur l'étendue de cet aspect mais avons tout de même effectué un travail de recherche théorique [27 - 28 - 29 - 30 - 31] à partir duquel nous avons pu apporter les quelques améliorations basiques (mais importantes) qui seront présentées ci-après. Certaines pistes d'amélioration seront également proposées pour chacun des concepts abordés.

3.2.2.1 Distance d'affichage

En théorie

La qualité de distance d'affichage [32] constitue un premier concept important concernant les performances d'un jeu dans UE. En fonction de la machine utilisée pour visualiser un environnement (notamment au niveau de la carte graphique), il est possible de rencontrer une diminution notable des performances si ce paramètre n'est pas bien géré.

Un concept important à introduire pour comprendre la suite du document est le « Level Of Details » (LOD ou « niveau de détails » [33]). En graphisme, un objet peut être représenté par l'assemblage d'un certain nombre de triangles [34] (ceci dans le but de pouvoir plus facilement visualiser la manière dont ils sont modélisés). Dans UE, plus le LOD d'un objet est bas, plus il sera constitué de triangles et sera donc visuellement plus détaillé. Bien que contre-intuitif, un objet ayant un LOD de 1 sera donc plus détaillé qu'un objet ayant un LOD de 3, comme le montre la figure ci-dessous.

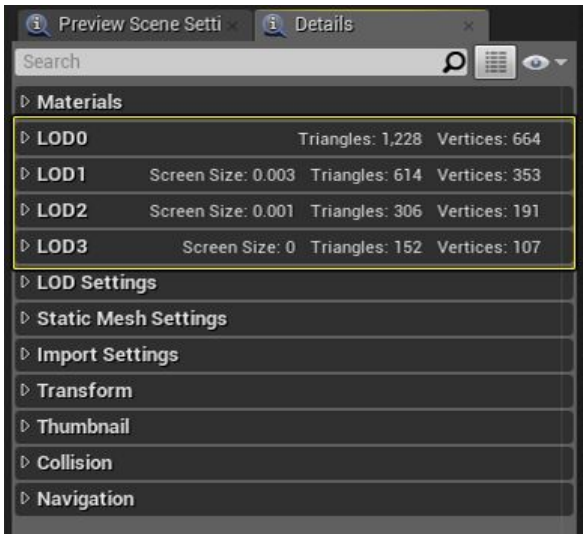


Figure 4 - Représentation du nombre de triangles en fonction du LOD dans UE

Prenons l'exemple d'un jeu dans lequel l'utilisateur se déplace au sein d'une ville. Si le LOD des bâtiments est élevé peu importe l'endroit où se trouve le joueur, celui-ci présentera d'importantes baisses de performances. Dans ce même exemple, il faudrait faire en sorte qu'un bâtiment se trouvant à 200 mètres, ne présente pas le même LOD qu'un bâtiment se trouvant à 10 mètres.

Il est cependant important de trouver une balance correcte [35] pour que le joueur ne constate pas de mauvaise qualité graphique. Cela pourrait être le cas dans l'exemple où la distance d'affichage n'est pas idéalement gérée et que trop d'éléments (dans le champ de vision direct du joueur) présentent un LOD élevé.

En pratique

Lorsque nous avons pris le projet Digital Twin en main, nous avons remarqué que certains éléments de décor présentaient un souci concernant cette distance d'affichage. Plus particulièrement au niveau des arbres présents dans l'environnement. En effet, ils présentaient un faible LOD à une distance de plusieurs centaines de mètres.

Il est important de rappeler que l'élément d'observation principal de la simulation est le carrefour ainsi que les véhicules le parcourant. Ainsi, avoir un faible LOD pour les forêts environnantes n'est pas une priorité.

Pour pallier ce problème, nous avons donc procédé à une série de tests afin de trouver la meilleure balance et avons réduit la distance d'affichage des arbres se situant au-delà de 50 mètres.

Figure 5 - Vue rapprochée des arbres montrant un faible LOD

Figure 6 - Vue éloignée des arbres montrant un LOD plus important

La *Figure 5* montre un faible LOD pour les arbres ayant une distance d'affichage rapprochée. La *Figure 6*, quant à elle, montre que les arbres ayant une distance d'affichage plus éloignée présentent un LOD plus important, voire disparaissent.

Étant donné l'ampleur du monde créé dans le projet Digital Twin, cette simple amélioration nous a permis de remarquer une amélioration notable des performances.

3.2.2.2 Shader Complexity

En théorie

Dans UE (et plus généralement dans le monde du graphisme), un « shader » [36] représente une série d'instructions spécifiques qui détaille la façon dont sera visuellement présenté un objet sur l'écran. Ce sont donc les shaders qui informeront le moteur de jeu de la valeur (ou poids) que devra avoir chaque pixel à chaque image.

Dans Unreal, lorsque nous sommes amenés à visualiser le rendu d'une scène, le moteur de jeu procédera donc, dans un premier temps, à la compilation de ces shaders.

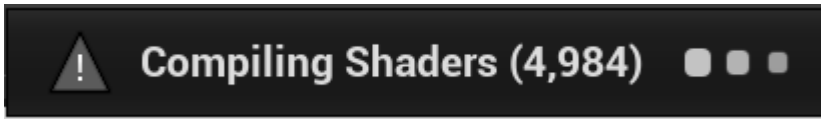


Figure 7 - Fenêtre indiquant la compilation des shaders dans UE

Il est possible que le nombre de shaders à compiler soit de plusieurs dizaines (voire centaines) de milliers pour les scènes les plus riches en matériaux. Cela peut être très désavantageux pour les machines moins puissantes qui seraient amenées à visualiser de telles scènes.

En effet, en fonction des techniques utilisées pour construire un monde dans un jeu vidéo, la complexité des shaders (souvent retrouvé dans la littérature sous sa traduction anglaise « shader complexity » [37]) peut considérablement varier et ainsi impacter les performances du jeu en question.

Dans UE, il est possible de visualiser, pour un environnement, ce niveau de complexité des shaders. Cela consiste à changer l'interface d'affichage [38] pour présenter un code couleur qui définira le poids des shaders. Cela permet de se rendre facilement compte des éléments graphiques les plus complexes à compiler et donc les plus susceptibles de diminuer les performances.

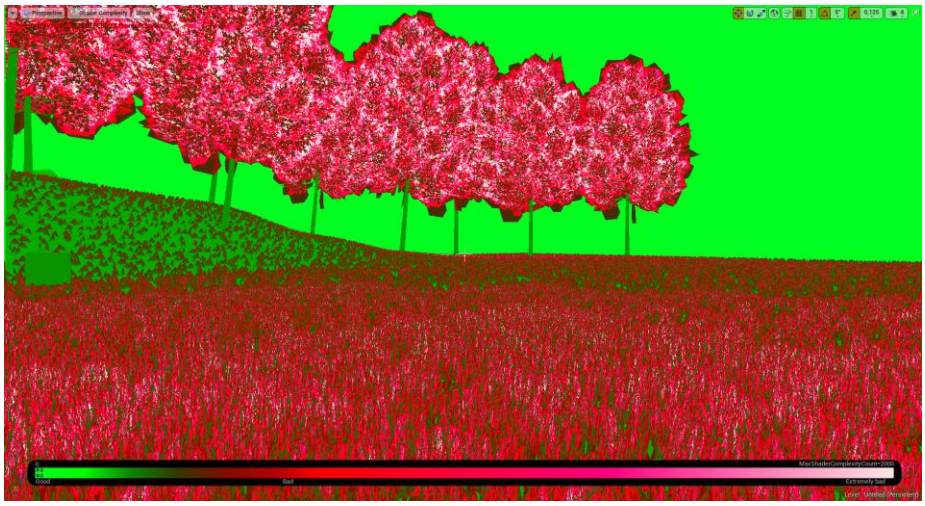


Figure 8 - Exemple d'interface présentant la shader complexity d'une scène [1]

Dans la figure ci-dessus, nous pouvons observer un environnement dans lequel se distinguent plusieurs niveaux de complexité. Dans la partie inférieure de l'image se trouve une légende indiquant un dégradé de couleur permettant de représenter la complexité des shaders. La couleur verte indique un très bon niveau de complexité, les couleurs rose et blanc indiquent un très mauvais niveau de complexité.

En pratique

La figure ci-dessous montre les niveaux de shaders complexity présents dans l'environnement du projet Digital Twin au moment où nous l'avons pris en main.

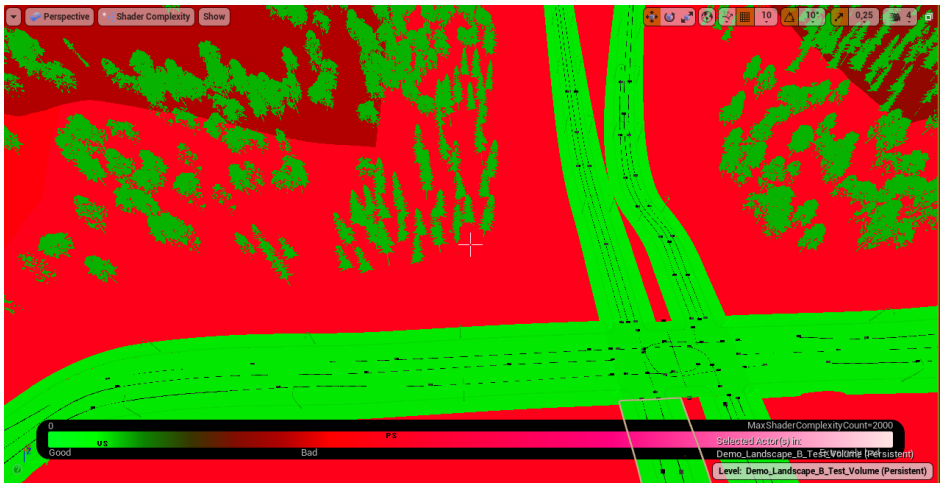


Figure 9 - Shader complexity initial de l'environnement du projet Digital Twin

[1] https://www.reddit.com/r/unrealengine/comments/b4zhzx/is_there_a_way_to_reduce_shader_complexity_with/

Comme nous pouvons le remarquer, la shaders complexity du sol est importante. Celle de la végétation (arbres, buissons, etc.) et de la route est, quant à elle, très bonne.

Pour pallier ce problème, nous avons abordé deux pistes différentes.

La première concerne le nombre de couches de textures utilisées dans le sol. Le plugin utilisé pour construire le sol présente huit couches de textures (terre, graviers, etc.). Nous avons remarqué que certaines de ces couches n'étaient pas utiles, dans le sens où elles n'apportaient rien d'un point de vue visuel. Supprimer certaines d'entre elles est donc une première piste d'amélioration.

La seconde piste consiste à réduire le nombre de triangles utilisés. Pour rappel, le LOD impacte directement le nombre de triangles [39]. Au plus le LOD d'un objet est bas, au plus il sera constitué de triangles, ce qui augmente drastiquement la complexité de calcul des shaders. Optimiser les paramètres de distribution du niveau de détails (« LOD Distribution » dans la littérature) constitue donc une piste importante à explorer en profondeur.

Après avoir procédé à une série de tests concernant ces deux pistes d'amélioration, nous avons remarqué une amélioration des performances au moment du lancement de la simulation. Cependant, le niveau de shader complexity ne s'en est pas trouvé impacté. Ceci est dû au fait que le plugin utilisé pour la texture du sol est bien trop complexe que pour présenter une shader complexity acceptable.

La performance du projet n'étant pas l'objectif principal, nous pouvons conserver la texture actuelle. Cependant, cela constituerait une amélioration importante si le projet venait à croitre en termes de consommation de ressources.

3.2.2.3 Ombres

En théorie

Dans tout jeu vidéo, les ombres sont un aspect primordial. En effet, les jeux présentent très souvent un élément essentiel au décor : le soleil. Pour éviter qu'un monde donne le sentiment d'être plat et sans relief, il est important que tout objet possède une ombre.

Ainsi, les ombres constituent un élément pouvant altérer aussi bien les performances que le réalisme d'un jeu. Disposer d'ombres de qualité élevée permet à un utilisateur de mieux apprécier son environnement. Cependant, cela rajoute un niveau de détail supplémentaire à un objet et constitue donc une potentielle source de baisse des performances.

Il est donc important de trouver la meilleure balance [40] pour obtenir une qualité d'ombre élevée en évitant au moteur de jeu d'être surchargé de calculs. Pour ce faire, une piste intéressante serait de faire un parallèle avec la distance d'affichage (cf. 3.2.2.1). Le but serait de garder une qualité d'ombre élevée à une certaine distance du joueur, et procéder à une diminution de son LOD lorsque le joueur s'écarte. La balance est cependant complexe à trouver car il faut éviter que le joueur ressente une baisse graphique trop importante.

Il est important de noter que les ombres sont étroitement liées au concept de lumière [41 - 42]. Une nouvelle piste serait d'approfondir son étude et ainsi trouve la meilleure optimisation pour avoir un résultat abouti.

En pratique

Dans le cas du projet Digital Twin, nous avons remarqué que les ombres présentaient des défauts. En effet, celles-ci étaient de mauvaise qualité pour les objets proches, et inexistantes pour les objets éloignés.

Nous avons donc réalisé une série de tests afin de trouver la représentation des ombres qui conviendrait le mieux à notre environnement.La figure suivante présente donc, sur l'imagine de droite, le changement apporté au niveau de l'ombre d'un lampadaire. Nous remarquons que celle-ci est désormais bien plus nette pour la faible distance d'affichage qu'elle présente.

Figure 10 - Ombres initiales (gauche) et ombres optimisées (droite)

3.2.3 Réalisme

Après avoir exploré quelques pistes d'amélioration de performances du projet Digital Twin, notre attention s'est portée sur les éléments pouvant augmenter son réalisme. En effet, il est important de rappeler que l'utilité principale de la simulation est d'en collecter des données se rapprochant le plus possible de la réalité. C'est donc un aspect qui a toute son importance dans le cadre de ce travail.

3.2.3.1 Environnement

L'environnement du projet Digital Twin a été inspiré d'un carrefour situé à Louvain-La-Neuve. En revanche, les matériaux, la végétation et quelques autres éléments visuels ne correspondaient pas à la réalité de l'environnement belge. Les éléments les plus flagrants étaient les types de plantes utilisées, comme des fougères que l'on retrouve dans des climats tropicaux plutôt que dans un climat tempéré tel que celui de la Belgique.

En y regardant de plus près, nous remarquons que ces détails jouent un rôle très important dans l'immersion d'un environnement. De nombreux autres éléments peuvent être pris en compte tels que :

- La qualité des routes (légèrement usées)
- Les marquages au sol ainsi que leur couleur
- Les panneaux de signalisation
- La langue des mots utilisés sur des affiches, panneaux, etc.
- Le relief et le type d'environnement (montagneux, désertique, etc.)
- L'architecture des bâtiments et les infrastructures routières (plus ou moins développées)

Pour ce qui est de Digital Twin, nous nous sommes limités à quelques modifications. Celles-ci incluent le changement des matériaux de la route, le remplacement des plantes tropicales, le remplacement des feux tricolores par un modèle plus réaliste et une modification de la densité des plantes environnantes.

3.2.3.2 Véhicules

Comme nous avons pu le constater précédemment, s'attarder sur des détails permet d'enrichir le réalisme. L'un des détails sur lequel nous avons également travaillé concerne la teinte de la carrosserie des véhicules.

En théorie

Dans la vie de tous les jours, il n'est pas impossible de croiser une voiture de couleur rose ou jaune sur les routes. La fréquence d'apparition de ces couleurs n'est toutefois pas élevée. Il est plus probable de rencontrer une couleur de carrosserie plutôt qu'une autre. Cette probabilité diffère également en fonction de la zone géographique ou encore du type de véhicule.

En pratique

En nous basant sur des ressources provenant d'internet, nous avons fait en sorte de générer des couleurs de véhicule plus réalistes car basées sur les teintes de carrosserie les plus populaires en 2020 [43]. Nous avons pris soin de permettre le changement des teintes et de leur fréquence d'apparition si besoin (dans le cas où les tendances venaient à changer).

L'implémentation que nous avons imaginée utilise un fichier de configuration dans lequel deux types de véhicules sont considérés, une voiture simple et un fourgon de travail. Pour chacun d'eux, le fichier de configuration contient des noms de teinte et leur probabilité d'apparition. (Il ne s'agit pas exactement de probabilité mais plutôt de poids. Une couleur blanche avec un poids de 25 sera plus commune qu'une couleur jaune avec un poids de 1). Le fichier de configuration utilisé se trouve en annexe 1 de ce document.

Lors de la phase de création d'un véhicule, celui-ci fait appel à une fonction « *Generate Color* ». Cette fonction étant requise pour la totalité des véhicules, nous l'avons créée dans une « Blueprint Function Library ». De la sorte, cette fonction est accessible pour de nombreux autres codes et cela sans conserver d'état. Cette fonction « *Generate Color* » débute par la récupération des valeurs dans l'objet « *CarsColorConfigObject* » qui possède une copie des valeurs du fichier de configuration.

Cette copie est réalisée au démarrage du projet pour éviter de devoir procéder à des lectures sur le disque à chaque fois que ces valeurs sont sollicitées. Quand elles sont en notre possession, nous récupérons celles qui correspondent au type du véhicule appelant la fonction et effectuons un léger traitement. Les poids sont stockés dans un tableau pour lequel chaque emplacement est égal à un poids additionné avec le poids des emplacements précédents.

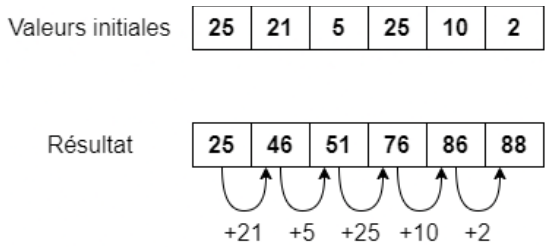


Figure 11 - Exemple d'additions des poids dans un tableau

Le tableau qui résulte de ces additions est trié par ordre croissant. Nous pouvons alors réaliser une variation du Binary Search [44] sur ce tableau. (Nous l'avons implémenté de sorte que la génération de couleurs soit la plus rapide possible). Nous générons une valeur aléatoire entre zéro et la valeur maximale présente dans le tableau et appliquons ensuite le Binary Search pour rechercher l'index de la valeur la plus proche de ce nombre aléatoire. Une fois l'index trouvé, nous l'utilisons dans un tableau contenant les couleurs afin de récupérer celle qui est associée à la valeur aléatoire générée.

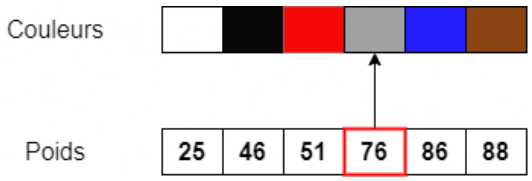


Figure 12 - Exemple de sélection de teinte selon une valeur aléatoire

Enfin, une des variantes disponibles pour cette couleur (gris clair, gris foncé, argent, etc.) est choisie aléatoirement et utilisée par le véhicule comme teinte de carrosserie.

Un élément pouvant également contribuer à l'amélioration du réalisme des véhicules concerne la physique appliquée sur ceux-ci. En effet, le moteur de jeu UE propose une physique par défaut, certes réaliste, mais pouvant être grandement améliorée sur certains aspects afin qu'il y ait une meilleure synergie avec l'environnement. Ce sont ces aspects qui constituent la seconde partie de notre travail, avec l'intégration de Robotran.

3.3 Robotran

Dans cette section, nous allons aborder la deuxième partie de notre travail de fin d'étude dans laquelle nous décrirons l'intégration de l'outil Robotran au sein de l'ensemble Digital Twin dans un but d'amélioration de la physique des véhicules.

3.3.1 Interaction théorique

Le schéma ci-dessous, notre modèle, représente une « boite noire » que nous devons intégrer au projet Digital Twin. Ce modèle reçoit différentes données à différentes étapes du cycle de vie du programme. Il a été conçu à partir du modèle mécanique présenté dans l'Annexe 2 de ce manuscrit.

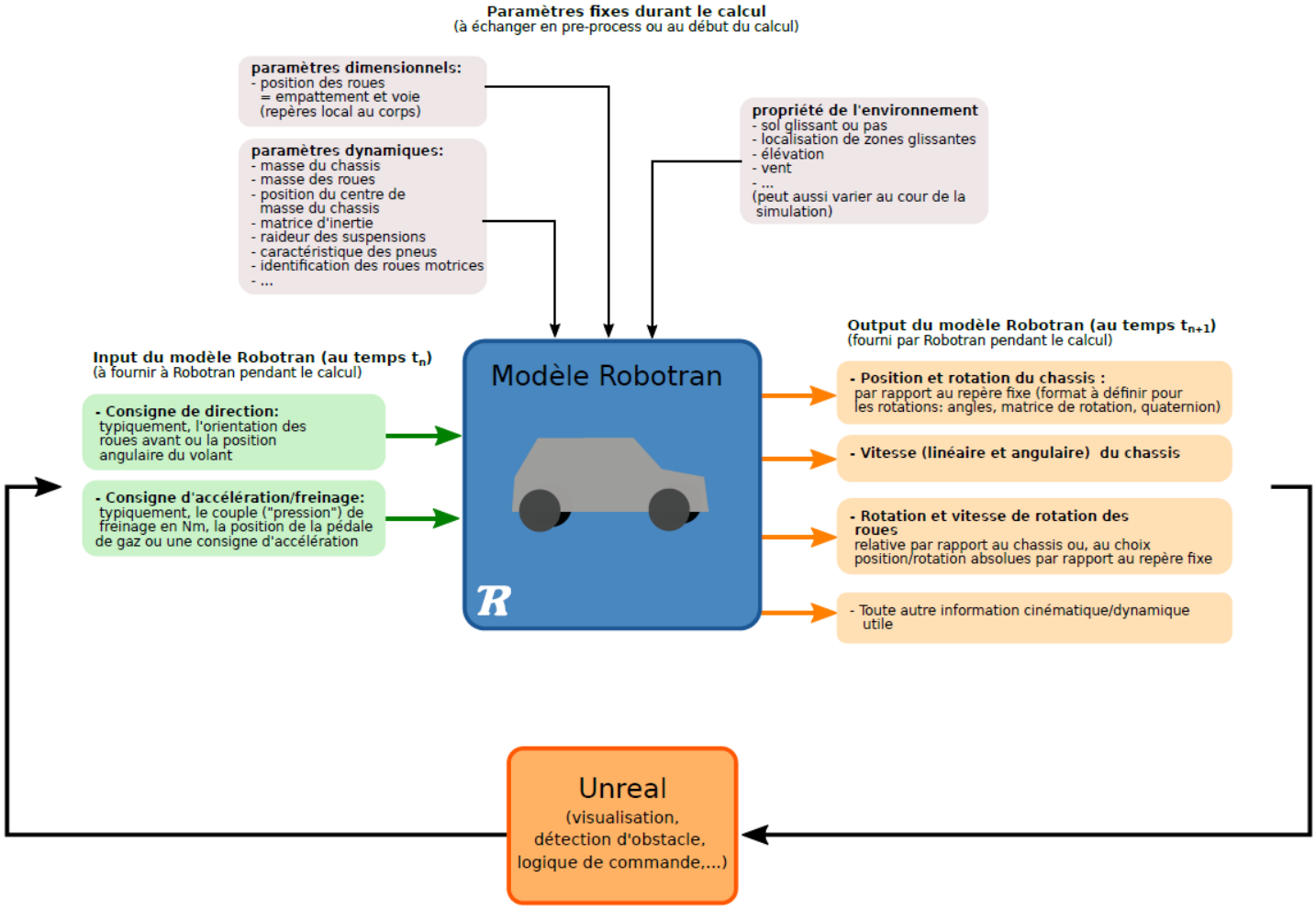


Figure 13 - Modèle Robotran (réalisé par Nicolas Docquier, 2022)

Nous allons d'abord nous attarder sur les données qui ne sont pas modifiées durant les calculs. Il s'agit des cases grises dans la partie supérieure du schéma. Ces paramètres sont (majoritairement) fixes car cela n'aurait aucun sens de les modifier durant la simulation. En les consultant, nous remarquons qu'il s'agit de dimensions, de masses, caractéristiques du véhicule et de l'environnement, etc. La majeure partie de ces valeurs définissent le véhicule tandis que les autres indiquent différents aspects pouvant impacter son comportement. En ce qui concerne le vent et l'élévation de la simulation, nous avons décidé de ne pas les prendre en compte à des fins de simplification de notre cas d'étude. L'objectif étant principalement de s'assurer du couplage des applications, nous nous limiterons à un cas d'utilisation permettant la visualisation du comportement d'un véhicule sur une zone sèche ou glissante.

Ensuite, viennent les valeurs qui fluctuent dans le temps par rapport à la conduite du véhicule. Elles sont représentées par les cases vertes à gauche du schéma. Les calculs du modèle Robotran dépendent de l'accélération du véhicule, du freinage et de la direction dans laquelle le véhicule se déplace selon l'angle de rotation du volant. Ces valeurs vont en effet évoluer au cours du temps et serviront à calculer l'état futur du véhicule en y appliquant les contraintes de l'environnement. Le nouvel état du véhicule est alors calculé en indiquant le temps écoulé entre l'état passé et ce nouvel état. Nous pouvons donc obtenir une nouvelle position, orientation, etc. à un nouvel instant de la simulation.

Enfin, nous recevons des valeurs pour la simulation UE en provenance du programme Robotran. Elles sont représentées par les cases orange clair à droite du schéma. Ces valeurs sont les informations nécessaires pour représenter le comportement physique du véhicule dans UE. Ce sont principalement les positions des différents corps composant le véhicule ainsi que leurs orientations dans l'espace.

3.3.2 Intégration pratique

Cette section aborde l'intégration de Robotran de façon plus pratique. Nous allons passer en revue les différentes phases d'analyses de plusieurs concepts ou technologies ainsi que les choix d'implémentation que nous avons fait.

3.3.2.1 Communication entre programmes

Afin d'échanger les données citées au point précédent, il est fondamental de mettre en place une stratégie de communication. Cela doit se faire en prenant en compte les particularités des deux parties, leur environnement et leur capacité à fonctionner ou s'adapter si cet environnement venait à changer. Nous analyserons également le format des données qui transitent entre les programmes.

3.3.2.2 Particularités des partis

A première vue, les deux partis que sont Robotran et le projet Digital Twin semblent faire la paire. Le logiciel Robotran, comme mentionné précédemment (cf. 2.2.1), peut fonctionner sous plusieurs langages. Ceux-ci étant Matlab, C/C++ et Python. En ce qui concerne le projet Digital Twin, il a été réalisé avec le moteur de jeu UE qui fonctionne avec le langage C++. Il était donc naturel de travailler avec la version C/C++ de Robotran.

Malgré tout, cette combinaison n'est pas parfaite. Nous le remarquons en observant la façon dont le projet Digital Twin a été réalisé. Ce dernier utilise très certainement le langage C++ mais de manière indirecte. Il utilise presque exclusivement un concept précédemment cité (cf. 2.1.1) se nommant le BVS. Cependant, cela ne nous mène pas dans une impasse car il est tout à fait possible d'utiliser du code hybride en C++ et BVS ou bien encore uniquement du C++.

L'utilisation majoritaire du BVS est bénéfique sur plusieurs points. L'ajout de code est nettement plus rapide et la lecture est plus aisée. Le projet étant un prototype, ces points sont importants car ils nous accompagnent vers l'un de nos objectifs qui est de vérifier la faisabilité d'intégration avant tout.

3.3.2.3 Environnement

Le second aspect à prendre en considération dans la stratégie de communication concerne le/les environnement(s) dans le/lesquel(s) le projet est exécuté. Celui-ci étant dans une phase de prototypage, il est plus simple (d'un point de vue pratique) d'exécuter celui-ci dans un environnement local. C'est-à-dire dans un même réseau et sur une unique machine. S'interroger sur son utilisation future permet néanmoins d'éviter des retours en arrière dans son développement.

La décision d'exécuter Robotran et Digital Twin dans le même réseau est plutôt simple. Ajouter un réseau entre les deux logiciels ferait naître des désavantages tels que des erreurs de communications à la suite d'une instabilité de la connexion (de même qu'une coupure complète), un trafic d'informations important si la simulation est utilisée pour modéliser un grand nombre de véhicule ainsi qu'une nécessité de synchroniser les deux partis. La simulation pourrait en être affectée en ayant comme effet d'être saccadée dû à une perte de Frames Par Seconde (FPS). L'exécution en temps réel n'est pas indispensable mais est favorable si l'on souhaite pouvoir consulter visuellement le résultat de la simulation.

Le projet Digital Twin et Robotran sont actuellement exécutés sur un ordinateur de bureau. Cependant, l'utilisation d'un serveur serait judicieuse en phase finale du prototype afin de bénéficier d'une plus grande puissance de calcul ainsi que pour tester la scalabilité du projet. L'un des objectifs finaux étant la génération de données, avoir à disposition un serveur permettrait également de laisser fonctionner la simulation pendant une longue période.

3.3.2.4 Introduction aux sockets

Les sockets [45] sont un bon compromis pour prendre en compte chacun des points cités précédemment. Un socket est une extrémité d'un tunnel de communication entre deux programmes dans un réseau. Ce réseau peut être local, c'est-à-dire sur la même machine au sein du même réseau, ou bien non local et passer au travers un ou plusieurs réseaux.

Généralement, ils sont utilisés dans des applications client-serveur. Dans notre cas, Robotran est considéré comme un serveur qui attend que le client Digital Twin effectue une connexion. Une fois la connexion réalisée, le client envoie une requête au serveur qui lui retournera un résultat. Cette communication est donc bidirectionnelle. Pour identifier les deux acteurs qui échangent au travers d'un réseau, chacun a besoin d'une adresse. Cette dernière est composée d'une adresse IP (Internet Protocol) et d'un port.

Il existe deux types de socket [46] : le « Datagram socket » qui fonctionne avec le protocole « User Datagram Protocol » (UDP) et le « Stream Socket » qui fonctionne avec le protocole « Transmission Control Protocol » (TCP). Le TCP est généralement utilisé pour la communication inter-processus. Une connexion est établie entre les deux machines communicantes avant de débuter leurs échanges. Il est fiable car le transfert de données est garanti sans erreur. Si une partie des données est perdue ou altérée, le protocole se charge de renvoyer les données en question. Il est plus lent que le protocole UDP qui ne nécessite pas de connexion entre les communicants mais n'offre aucune garantie concernant l'intégrité des données.

Les sockets sont également une technologie très générale qui fonctionne sur beaucoup de système d'exploitation (Unix, Windows, MacOS…).

Nous avons sélectionné cette solution avec le protocole TCP pour plusieurs raisons. La première étant la simplicité de mise en place grâce à un plugin nommé « TCP Socket Plugin » [47] disponible sur le Marketplace UE et utilisable avec le BVS. La seconde étant la garantie de l'intégrité des données, bien que le risque d'erreur dans un transfert local soit extrêmement faible voire inexistant.

3.3.2.5 Les sockets dans UE

Nous avons donc décidé d'utiliser la technologie des sockets pour créer un tunnel de communication entre nos deux programmes. Ceux-ci ont été implémentés en BVS du côté de UE et ont nécessité quelques manipulations supplémentaires.

L'objet (véhicule) que l'on utilise dans notre simulation hérite de la classe « AI Car Pawn ». Ce parent quant à lui hérite d'une classe propre à UE nommée « Wheeled Vehicule » qui est une base nécessaire pour faire de notre objet un véhicule. Pour ajouter les sockets, notre objet devait également hériter d'une autre classe « TCP Socket Connection ». Le problème étant qu'UE n'autorise pas l'héritage multiple. Un objet ne pouvant hériter que d'un unique parent, il nous était impossible d'employer la librairie TCP telle quelle dans notre véhicule. Ne pas pouvoir utiliser l'héritage multiple peut être compensé par l'utilisation d'interfaces. Cette façon de fonctionner est préférable pour de multiples raisons mais n'est pas adaptée à notre situation.

Pour remédier à cet inconvénient, nous avons ajouté un second acteur à notre simulation. Cet acteur existe dans la simulation mais ne l'influence pas physiquement. Ce dernier hérite de « TCP Socket Connection » et sert d'outil de communication au véhicule pour échanger de l'information avec Robotran. Le véhicule et l'outil ayant chacun une référence l'un vers l'autre, ils peuvent donc se coordonner.

3.3.2.6 Format de données

La question portant sur le choix du format de données utilisé est importante pour la suite de notre travail car celui-ci impactera la manière dont nous allons structurer notre code tant du côté d'UE que du côté de Robotran.

Le schéma suivant permet de visualiser la manière dont se fait le transfert de données entre le modèle Robotran et notre environnement UE.

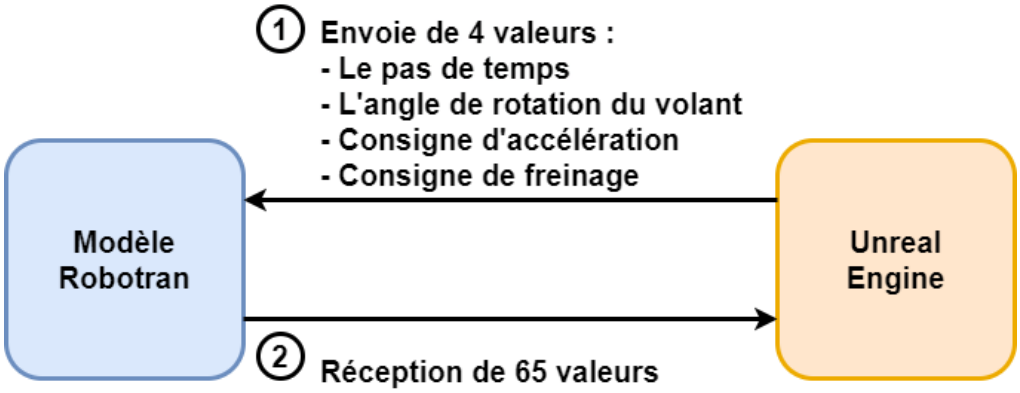


Figure 14 - Représentation du transfert entre les applications Robotran et UE

L'envoi de données en première phase est suffisamment simple que pour envoyer directement les valeurs sous forme d'une simple chaîne de caractères. La seconde phase en revanche, est indéniablement plus complexe. UE reçoit un total de 65 valeurs divisées en cinq groupes de 13 valeurs. La gestion de cet ensemble de valeurs requiert une structure formelle et manipulable. Dans chaque ensemble, nous retrouvons exactement les mêmes types de données mais pour des usages différents. L'un est utilisé pour le châssis du véhicule tandis que les quatre autres sont utilisés pour chacune des roues du véhicule.

Un ensemble est constitué des champs suivants :

- x : position calculée sur l'axe x
- y : position calculée sur l'axe y
- z : position calculée sur l'axe z
- vx : vélocité calculée sur l'axe x
- vy : vélocité calculée sur l'axe y
- vz : vélocité calculée sur l'axe z
- q0 : partie réelle du quaternion
- qx : terme x du quaternion
- qy : terme y du quaternion
- qz : terme z du quaternion
- omx : vélocité angulaire sur l'axe x
- omy : vélocité angulaire sur l'axe y
- omz : vélocité angulaire sur l'axe z

Nous étudierons plus précisément certains de ces champs dans les sections suivantes.

Durant la phase de transmission des données, nous avons décidé d'utiliser un format qui facilite leur manipulation. Il existe plusieurs formats très connus comme le « eXtensible Markup Language » (XML [48]), « JavaScript Object Notation » (JSON [49]) ou encore « YAML Ain't Markup Language » (YAML [50]) . L'emploi du format JSON est simple et très largement utilisé dans les applications client-serveur. C'est le format que nous avons choisi car une librairie prête à l'emploi (« Json Blueprint » [51]) était disponible sur le Marketplace d'UE. Celle-ci nous permet d'analyser le résultat en JSON (« parsing » [52]) fourni par Robotran et en récupérer les valeurs pour les traiter de manière simple et efficace.

Voici une représentation simplifiée de la structure finale en JSON :

```
{
    "sedan":[
        {"chassis" :{
            "x" : 0.00000,
            "y" : 0.00000,
            "z" : 0.00000,
            "vx" : 0.00000,
            "vy" : 0.00000,
            "vz" : 0.00000,
            "q0" : 0.00000,
            "qx" : 0.00000,
            "qy" : 0.00000,
            "qz" : 0.00000,
            "omx" : 0.00000,
            "omy" : 0.00000,
            "omz" : 0.00000
            }
        },

        {"roues":[
                { "FL wheel" :{
                    // Même ensemble de valeur
                    }
                },
                { "FR wheel" :{
                    // Même ensemble de valeur
                    }
                },
                { "RL wheel" :{
                    // Même ensemble de valeur
                    }
                },
                { "RR wheel" :{
                    // Même ensemble de valeur
                    }
                }
            ]}]}
```

Figure 15 - Exemple simplifié de données sous format JSON

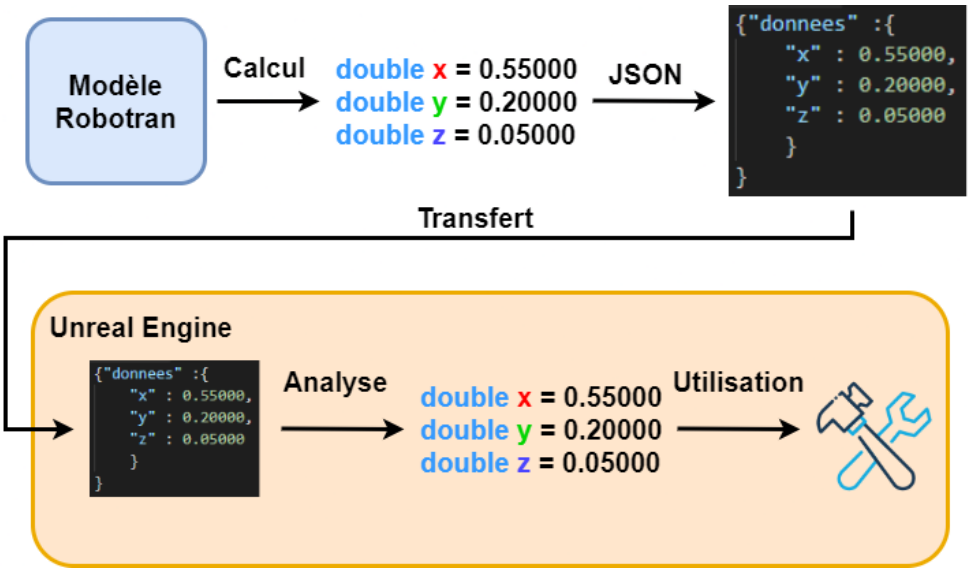


Figure 16 - Représentation du transfert de l'application Robotran vers UE

3.3.2.7 Système de coordonnées

Lorsque nous mentionnons une position, celle-ci doit exister par rapport à un repère. Il s'agit d'un point de référence à partir duquel nous pouvons mesurer l'emplacement d'un objet dans un espace [53]. Ce dernier peut avoir zéro dimension, une dimension, deux dimensions, trois dimensions ou plus encore. Dans le domaine des jeux vidéo et de la simulation, ces espaces sont restreints à deux ou trois dimensions [54].

Notre simulation Digital Twin existe dans un espace en 3D (un repère cartésien est utilisé pour interpréter les coordonnées). Il en va de même pour le logiciel Robotran. Tout objet existant dans un tel espace peut donc se déplacer selon trois axes nommés comme suit : X, Y et Z. Le rôle de ce système de coordonnée est donc d'utiliser des nombres pour indiquer la localisation d'un point ainsi que la direction dans laquelle l'objet est orienté.

Si nous essayons de représenter un objet dans le système de coordonnée de Robotran et que nous représentons ce même objet dans UE, nous pourrons remarquer que l'objet n'est pas orienté exactement de la même manière. Robotran utilise un système de coordonnées dit « main droite » [55] comme illustré ci-dessous.

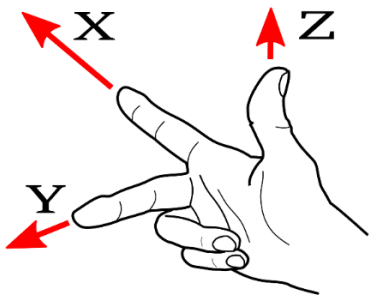


Figure 17 - Représentation du système de coordonnées main droite [2]

UE quant à lui, utilise un système de coordonnées « main gauche ».

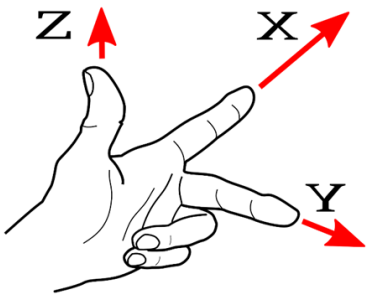


Figure 18 - Représentation du système de coordonnées main gauche [2]

[2] https://stackoverflow.com/questions/19747082/how-does-coordinate-system-handedness-relate-to-rotation-direction-and-vertices

Il n'y a pas d'ambiguïté concernant l'axe X indiquant le sens « avant/arrière » et l'axe Z indiquant le sens « haut/bas ». La différence entre ces deux systèmes se situe au niveau de l'axe Y. Les flèches rouges sur les illustrations précédente indiquent un vecteur positif. Le vecteur Y s'appelle le « Right vector » en anglais (ce qui n'est pas très intuitif étant donné qu'il n'indique pas forcément la droite). Ce vecteur qui indique une valeur positive vers la gauche pour Robotran est négative dans le système de coordonnées de UE et vice versa.

Afin de visualiser cette différence de manière plus concrète, nous avons conçu un objet en 3D avec le logiciel Blender [56]. Ce logiciel de modélisation 3D utilise un système de coordonnées main droite tout comme Robotran. Par la suite, nous avons exporté l'objet dans UE. Nous constatons en effet pour l'axe Y, représenté avec une flèche verte, une inversion de la direction positive de la valeur Y.

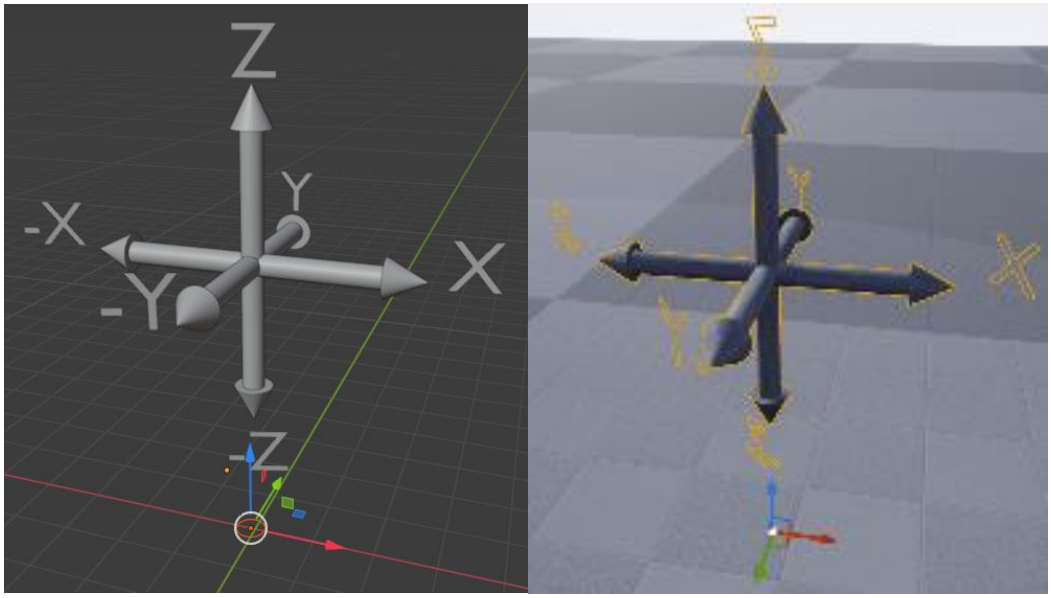


Figure 19 - Comparaison d'un objet dans deux systèmes de coordonnées différents

Cette particularité implique qu'une « conversion » est nécessaire dès lors que l'on souhaite utiliser des valeurs en provenance de Robotran dans un projet UE. Dans le cas de la coordonnée d'un point tel que Position = (X=5, Y=10, Z=5), cette position sera utilisée avec les valeurs suivantes Position = (X=5, Y=-10, Z=5). Il nous suffit donc de multiplier par -1 les valeurs obtenues sur l'axe Y.

3.3.2.8 Orientation dans l'espace

En plus d'être positionné dans l'espace, un corps peut aussi subir des modifications consistant à l'orienter. L'orientation d'un corps rigide dans un environnement 3D constitue un élément primordial lors de la mise en place d'un projet. Cela permet de décrire la façon dont ce corps sera placé dans l'espace auquel il appartient.

Il est tout d'abord nécessaire de distinguer les termes « orientation » et « rotation ». En effet, ceux-ci sont souvent confondus, à tort. L'orientation décrit la direction dans laquelle est dirigé un objet. Tandis que la rotation décrit l'acte de changer l'orientation d'un objet. Il est important de noter qu'une orientation sera relative, ce qui signifie qu'elle ne sera décrite qu'en fonction d'un repère de référence.

Trois valeurs indépendantes sont nécessaires pour décrire l'orientation de cette référence. C'est ce qu'on appelle « référence locale du corps » ou « système de coordonnées local ». Ce concept est expliqué de manière plus détaillée dans le point suivant.

Nous pouvons donc définir l'orientation d'un objet comme étant la rotation que doit subir cet objet afin d'être placé de la même façon qu'une référence. Ainsi, changer l'orientation d'un corps s'apparente à appliquer une rotation des axes de coordonnées liés à ce corps.

Ces rotations autour des axes portent d'ailleurs des appellations bien spécifiques : « roll », « pitch » et « yaw » en anglais [57].

Roll désigne une rotation autour de l'axe X, aussi appelé « axe longitudinal ».
Pitch désigne une rotation autour de l'axe Y, aussi appelé « axe transversal ».
Yaw désigne une rotation autour de l'axe Z, aussi appelé « axe vertical ».

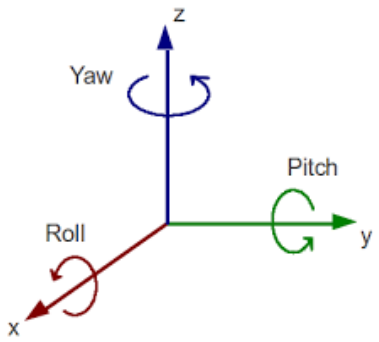


Figure 20 - Représentation des rotation roll, pitch et yaw [3]

Il existe différentes façons de décrire l'orientation d'un corps dans un environnement 3D [58]. Cette section a pour objectif de les décrire dans le cadre d'UE, en évitant d'émettre les démonstrations mathématiques les constituant.

Ces méthodes sont donc au nombre de quatre : les angles d'Euler, les vecteurs de rotation, les matrices de rotation et les quaternions. [59 - 60]

Il se trouve que Robotran est capable de prendre en charge ces quatre différentes technologies. Il était donc important d'étudier ces différentes possibilités en amont pour leur faire part de celle qui nous semblait être la plus adaptée à nos besoins.

[3] http://doc.aldebaran.com/2-4/family/robots/joints_robot.html

Angles d'Euler

En raison de son aspect intuitif, les angles d'Euler [61] constituent la façon la plus simple d'aborder le problème de la rotation d'un corps. Cette technique, mise au point par Leonhard Euler, consiste à décomposer l'orientation d'un objet en utilisant trois axes : X, Y et Z.

Comme énoncé précédemment, il est plus intuitif de saisir le fonctionnement des angles d'Euler en abordant une explication schématique. Le modèle représentant un avion est souvent utilisé afin de visualiser et étudier la rotation d'un corps dans l'espace de façon théorique.

Prenons donc en exemple les trois schémas ci-dessous.

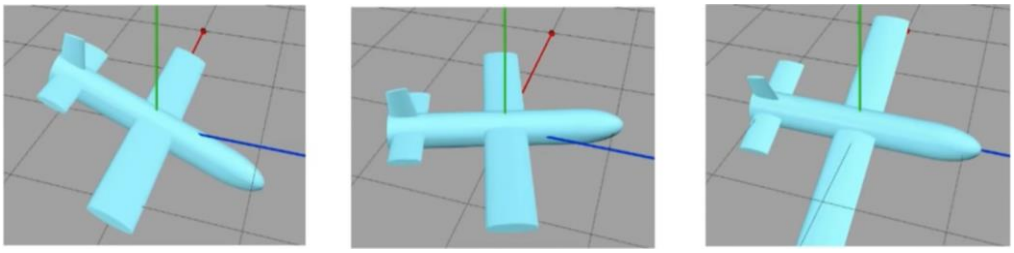


Figure 21 - Schéma comparatif de la rotation autour des trois angles d'Euler [4]

De gauche à droite, le premier schéma permet de visualiser la rotation d'un corps autour de l'axe X (ou « pitch », représenté par une rotation autour de l'axe de couleur rouge). Le second, autour de l'axe Y (ou « yaw », représenté par une rotation autour de l'axe de couleur vert) et le troisième autour de l'axe Z (ou « roll », représenté par une rotation autour de l'axe de couleur bleu).

Prenons le premier schéma. La rotation autour de l'axe X permettra de faire pivoter le corps dans le plan formé par les deux autres axes Y et Z. Cette logique est bien évidemment d'application pour l'ensemble des trois axes. C'est-à-dire que la rotation autour de l'axe Y permettra de faire pivoter le corps dans le plan formé par les axes X et Z, etc. C'est ainsi qu'en combinant différentes valeurs pour chacun des trois axes, il est possible d'orienter un objet dans n'importe quelle direction désirée.

Bien qu'intuitive à utiliser car étant facile à visualiser, cette méthode comporte certains problèmes majeurs.

Le premier est dû à la décomposition de la rotation en utilisant ces trois axes. Par exemple, un objet A sur lequel est effectué une rotation autour de l'axe X suivi d'une rotation autour de l'axe Y, et un objet B sur lequel est effectué une rotation autour de l'axe Y puis une rotation autour de l'axe X. En considérant que les valeurs utilisées pour les deux objets soient les mêmes, il se peut que les orientations respectives de A et B soient différentes. En effet, il faut être vigilant par rapport au fait que l'ordre dans lequel sont appliquées les rotations affecte l'orientation finale d'un objet.

[4] https://www.youtube.com/watch?v=q0jgqeS_ACM

Gimbal Lock

Le second problème qui se présente en utilisant les angles d'Euler se nomme le « blocage de cadran », plus connu sous le nom anglais de « gimbal lock » [62 - 63]. C'est une difficulté qui survient lorsque, après avoir subi des modifications au niveau de leurs angles, deux des trois cadrans se trouvent portés dans la même direction. Le cadran est utilisé pour représenter un anneau fixé de telle sorte à pouvoir tourner autour d'un axe. Ceux-ci sont imbriqués de sorte à pouvoir tourner autour de plusieurs axes simultanément.

Prenons en exemple la figure ci-dessous.

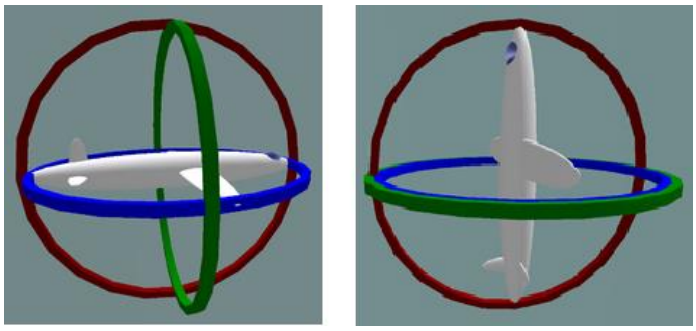


Figure 22 - Représentation du Gimbal Lock [5]

Dans la figure de gauche, un corps (représenté par un avion) est présenté en position neutre avec ses trois cadrans distinctement visualisables. Dans la figure de droite, une rotation « pitch » de -90° a été appliquée, ce qui implique que les cadrans bleus et verts sont désormais verrouillés ensemble. Une rotation du cadran vert impliquera donc toujours une rotation du cadran bleu, ce qui signifie que les rotation « pitch » et « yaw » auront désormais le même comportement. Ceci a pour conséquence la perte d'un degré de liberté quant à l'orientation d'un objet.

Vecteurs de rotation

Au fil du temps, Euler s'est rendu compte que : « dans l'espace tridimensionnel, tout déplacement d'un corps rigide tel qu'un point sur ce corps reste fixe, équivaut à une seule rotation autour d'un axe passant par le point fixe. Ce qui implique que la composition de deux rotations équivaut à une seule et même rotation autour d'un axe fixe différent. ». Ce théorème (Théorème de rotation d'Euler [64]) ne sera pas approfondi dans ce document mais il permet d'introduire une nouvelle façon de représenter une rotation dans l'espace.

Cette nouvelle méthode consiste à décrire n'importe quelle rotation grâce à un vecteur représentant chaque axe de rotation ainsi qu'une valeur séparée permettant de décrire l'angle de ce vecteur [65].

[5] https://fr.wikipedia.org/wiki/Blocage_de_cardan

Sa représentation est la suivante :

$$(axe, angle) = \left(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}, \theta \right)$$

Ainsi, le vecteur d'Euler suivant :

$$\left(\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \frac{\pi}{2} \right)$$

représente une rotation de $\frac{\pi}{2}$ radians (90°) autour de l'axe Y.

L'intérêt du vecteur de rotation réside dans le fait que cette représentation, intuitive à comprendre, soit aisée à convertir dans d'autres représentations (notamment en quaternions, qui seront expliqués plus loin dans le document).

Les vecteurs de rotation souffrent cependant, de même que les angles d'Euler, du problème du gimbal lock, présenté dans le point précédent.

Matrices de rotation

D'un point de vue historique, l'apparition des matrices a mené à la réécriture des théorèmes d'Euler. Cela a permis de pouvoir décrire une rotation sur base de matrices orthogonales appelées « matrices de rotation » ou « matrices de cosinus directionnels » [66].

Dans un premier temps, il y a ce qu'on appelle les matrices de rotation élémentaires. Ces matrices 3x3 permettent de représenter la rotation autour d'un seul des trois axes du système de coordonnées.

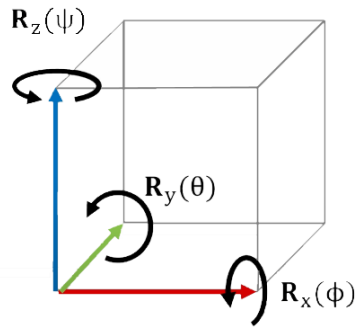


Figure 23 - Représentation sur laquelle se basent les matrices de rotation élémentaires

Par exemple, les matrices de rotation élémentaires pivotant les axes X (avec un angle ϕ), Y (avec un angle θ) et Z (avec un angle Ψ) et utilisant le système de coordonnées main droite peuvent être décrites comme suit :

$$R_x(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}$$

$$R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

$$R_z(\Psi) = \begin{pmatrix} \cos\Psi & -\sin\Psi & 0 \\ \sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Prenons la première matrice R_x représentant la rotation autour de l'axe X. Il est intéressant de noter que la ligne et la colonne associées à X (première ligne et première colonne) sont vides. Ce qui est évident étant donné que lors d'une rotation autour de l'axe X, tout vecteur pointant dans la direction de X restera inchangé.

Aussi, la sous-matrice 2x2 formée par les éléments $\begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix}$ correspond à la matrice de rotation définissant une orientation dans le plan formé par les axes Y et Z.

Ces observations sont valables pour chacune des trois matrices de rotation élémentaires présentées précédemment.

Ensuite, il y a ce qu'on peut définir comme étant les matrices de rotation générales, qui sont le produit des trois matrices de rotation élémentaires présentées précédemment.

Par exemple, la matrice de rotation générale dont les angles d'Euler $\phi, \theta\ et\ \Psi$ autour des axes X, Y et Z, peut être décrite comme suit :

$$R = R_z(\Psi)R_y(\theta)R_x(\phi)$$

$$= \begin{pmatrix} \cos\theta\ \cos\Psi & \sin\phi\sin\theta\cos\Psi - \cos\phi\sin\Psi & \cos\phi\sin\theta\cos\Psi - \sin\phi\sin\Psi \\ \cos\theta\ \sin\Psi & \sin\phi\sin\theta\sin\Psi + \cos\phi\cos\Psi & \cos\phi\sin\theta\sin\Psi - \sin\phi\cos\Psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\ \cos\theta \end{pmatrix}$$

Il est important de signaler que l'utilisation de ces matrices de rotation ne donnera un résultat correct que dans le cas où elles sont appliquées dans un ordre spécifique. Cela fait appel à un problème qui a déjà été mentionné dans la présentation des angles d'Euler. Dans le cas des matrices de rotation, il est nécessaire que les opérations de rotation se fassent de droite à gauche en termes de vecteur colonne.

Par exemple, dans la matrice de rotation reprise ci-dessus, le premier vecteur à devoir être appliqué sur un corps sera le suivant :

$$\begin{pmatrix} \cos\phi \sin\theta \cos\Psi - \sin\phi \sin\Psi \\ \cos\phi \sin\theta \sin\Psi - \sin\phi \cos\Psi \\ \cos\phi \cos\theta \end{pmatrix}$$

Il est évident que cette méthode utilisant les matrices de rotation est bien moins intuitive que celle utilisant les angles d'Euler, étant donné les opérations nécessaires à leur bonne utilisation. Aussi, il est important de noter que l'utilisation des matrices de rotation ne permet pas de résoudre le problème du gimbal lock. Leur intérêt réside dans le fait qu'une seule représentation de matrice permette de visualiser la totalité des différentes rotations devant être effectuées (une rotation étant donc représentée par un vecteur colonne).

Quaternions

Dans le but de clore l'énumération des différentes méthodes d'orientation dans un environnement 3D, il est indispensable de présenter celle qui consiste à utiliser la technologie des quaternions [67 - 68].

Bien que cette technologie s'apparente à celle des matrices et vecteurs de rotation, elle reste une solution davantage stable et efficace. En effet, les quaternions ont été développés dans le but de satisfaire un panel d'application bien plus large et peuvent donc s'adapter à un nombre de situations bien plus important.

De plus, l'un des principaux avantages des quaternions réside dans le fait qu'ils soient la seule manière d'éviter le gimbal lock. Ceci est dû au fait qu'ils n'utilisent pas un système de cadrans, à l'inverse des autres technologies. En effet, les angles d'Euler et les matrices de rotation (principalement basés sur les angles d'Euler dans leur utilisation) utilisent un système de trois cadrans dont chacun permet à un corps de tourner autour d'un axe unique. C'est bel et bien ce système de cadran qui est à la source du gimbal lock.

Cependant, les quaternions sont loin d'afficher le même caractère intuitif ou la facilité de compréhension que présentent par exemple les angles d'Euler. Ceci étant majoritairement dû à la complexité algébrique et mathématique qui les composent.

Cette section ne traitera pas de cet aspect mathématique de manière approfondie mais énoncera les éléments nécessaires à la bonne compréhension des quaternions dans le cadre de l'orientation d'un corps dans un espace tridimensionnel.

Tout d'abord, un quaternion est représenté par quatre éléments sous la forme suivante :

$$q = q_0 + iq_x + jq_y + kq_z$$

Où q_0 , q_x , q_y et q_z sont des nombres réels et *i, j* et *k* sont des vecteurs unitaires satisfaisant les relations quaternioniques suivantes :

$$i^2 = j^2 = k^2 = ijk = -1$$

Le terme q_0 représente le « composant réel » et les termes q_x , q_y et q_z représentent les « composants imaginaires ». D'un point de vue pratique, la notation imaginaire (c'est-à-dire les composants *i, j* et *k*) est implicite. Ce sont les quatre coefficients réels qui sont utilisés pour définir un quaternion.

Nous pouvons donc désormais définir un quaternion sous la forme suivante :

$$q = (q_0 , q_x , q_y , q_z)$$

La meilleure façon de comprendre le fonctionnement des quaternions est de repartir de la représentation proposée par les angles d'Euler, ou plus précisément de leur représentation vectorielle. S'en suivra une conversion vers la forme de quaternion.

En reprenant la notation ci-dessus et étant donné le vecteur (θ , x , y , z), (x , y , z) représente le vecteur unitaire définissant les axes de rotation et θ représente l'angle de rotation autour de ce vecteur unitaire.

Nous pouvons définir les différents termes d'un quaternion de la manière suivante :

$$q_0 = \cos\left(\frac{\theta}{2}\right)$$

$$q_x = \mathrm{x}\sin\left(\frac{\theta}{2}\right)$$

$$q_y = \mathrm{y}\sin\left(\frac{\theta}{2}\right)$$

$$q_z = \mathrm{z}\sin\left(\frac{\theta}{2}\right)$$

Comme nous pouvons le remarquer, le composant réel q_0 n'est déterminé qu'à partir de l'angle de rotation. Tandis que les trois composants imaginaires q_x , q_y et q_z correspondent aux trois axes de rotation $x , y \; et \; z$ auxquels se multiplie un facteur commun : $\sin\left(\frac{\theta}{2}\right)$.

Application d'un quaternion

Nous allons maintenant énoncer la marche à suivre afin d'utiliser ces quaternions dans le but d'effectuer une rotation [69].

Pour ce faire, deux concepts sont importants : la multiplication de quaternions et l'inversion de quaternions.

Tout d'abord, étant donné « c », le quaternion résultant de la multiplication de deux quaternions « a » et « b » :

$$c = a \times b$$
$$(c_0\ ,\ c_x\ ,\ c_y\ ,c_z) = (a_0\ ,\ a_x\ ,\ a_y\ ,a_z) \times (b_0\ ,\ b_x\ ,\ b_y\ ,b_z)$$

Les éléments constituant le quaternion « c » peuvent être définis de la manière suivante :

$$c_0 = (a_0b_0 - \ a_x\ b_x - \ a_yb_y \ - \ a_zb_z)$$
$$c_x = (a_0b_x + \ a_x\ b_0 - \ a_yb_z + \ a_zb_y)$$
$$c_y = (a_0b_y + \ a_x\ b_z + \ a_yb_0 \ - \ a_zb_x)$$
$$c_z = (a_0b_z - \ a_x\ b_y + \ a_yb_x + \ a_zb_0)$$

Il est important de noter que la multiplication de quaternion est associative ($(ab)c = a(bc)$) mais qu'elle n'est pas commutative ($ab \neq ba$).

Ensuite, l'inversion d'un quaternion est obtenue en rendant les composants imaginaires (à savoir q_x , q_y et q_z) négatifs :

$$q^{-1} = (q_0\ ,\ -q_x\ ,\ -q_y\ ,\ -q_z)$$

Inverser un quaternion a également pour effet d'inverser les axes de rotation, ce qui implique que l'application d'un quaternion inversé entraînera une rotation dans la direction opposée à l'originale.

Ainsi, en utilisant le quaternion de rotation « q », appliquer une rotation à un point P dont les coordonnées dans l'espace sont (x, y, z) nécessitera trois étapes qui utiliseront les concepts de multiplication et inversion de quaternions expliqués précédemment.

La première étape consiste à convertir le point P sous la forme d'un quaternion qui sera nommé « p ». Pour ce faire, il suffit d'assigner la valeur zéro au composant réel p_0 et d'assigner les coordonnées du point (x, y, z) aux composants imaginaires p_x , p_y et p_z . Le quaternion « p » résultant de cette étape sera donc :

$$p = (p_0 , p_x , p_y , p_z) = (0, x, y, z)$$

La seconde étape consiste à appliquer le quaternion de rotation « q » au quaternion « p » défini à la première étape. Pour ce faire, il faudra appliquer une double multiplication du quaternion « p ».

Il est possible d'effectuer deux doubles multiplications différentes en fonction de la rotation voulue. En effet, il existe la rotation active et la rotation passive. Une rotation active permettra d'appliquer une rotation à un point « P » en fonction du système de coordonnées mis en place. Une rotation passive, elle, permettra d'appliquer une rotation sur le système de coordonnées en fonction du point « P ». La nuance est importante car ces deux rotations sont opposées l'une de l'autre et la double multiplication énoncée précédemment sera différente pour chacune de ces rotations.

Si le quaternion « p' » est le résultat de ces doubles multiplications, appliquer une rotation active (avec un quaternion « q ») sur un point « P » (sous la forme du quaternion « p » défini à la première étape) peut être définie de la manière suivante :

$$p' = q^{-1} \times p \times q$$

De la même manière, une rotation passive peut être définie de la manière suivante :

$$p' = q \times p \times q^{-1}$$

Pour rappel, la multiplication entre quaternions n'est pas commutative. Les résultats des deux doubles multiplications ci-dessus seront donc bel et bien différents.

La troisième et dernière étape consiste à extraire les nouvelles coordonnées du résultat « p' ». Le quaternion « p' » résultant des étapes précédentes sera le suivant :

$$p' = (0, x', y', z')$$

Les nouvelles coordonnées (x', y', z') du point « P » sur lequel a été appliqué une rotation correspondent donc aux composants imaginaires du quaternion « p' ».

Technologie utilisée

Comme énoncé précédemment, Robotran nous a laissé le choix quant à la technologie à utiliser pour représenter la rotation d'un corps dans l'espace.

Notre choix s'est porté sur les quaternions.

Tout d'abord, il est légitime de se demander pourquoi utiliser une représentation si peu intuitive lorsqu'on sait que les différents composants d'un quaternion sont facilement calculables à partir des vecteurs d'Euler, ceux-ci étant bien plus faciles à aborder. Le fait est que, utiliser ces vecteurs d'Euler de manière efficace requiert d'appliquer des opérations trigonométriques. Ainsi, les faire en amont (sous la forme de quaternion donc) nous permet d'économiser des ressources de calcul inutiles.

Ensuite, un problème récurrent lors de l'utilisation des méthodes autres que les quaternions (et des vecteurs d'Euler dont l'utilisation a été exclue suite aux arguments ci-dessus) est le gimbal lock. Comme défini précédemment (cf. 3.3.2.8), il peut en effet être évité grâce à l'utilisation des quaternions.

De plus, en termes de stockage, les quaternions sont bien plus avantageux que les matrices de rotation, étant elles aussi régulièrement utilisées dans la pratique. En effet, l'utilisation des quaternions demande de stocker uniquement quatre valeurs, contrairement aux matrices de rotation qui en nécessitent neuf.

Malgré leur caractère non-intuitif qui peut être repoussant, notre choix s'est tout de même porté sur les quaternions.

Voici ci-dessous un tableau comparatif reprenant les avantages et inconvénients de chacune des technologies présentées :

	Avantages	**Inconvénients**
Angles d'Euler	Représentation intuitive.	Gimbal lock. L'ordre d'application des rotations importe.
Vecteurs de rotation	Représentation intuitive. Facile à convertir vers d'autres représentations.	Gimbal lock.
Matrices de rotation	Visualisation facilitée de la totalité des rotations à effectuer.	Gimbal lock. Lourd à stocker (9 valeurs nécessaires)
Quaternions	Pas de gimbal lock. Facilité de stockage (4 valeurs nécessaires).	Conception et compréhension complexes.

Tableau 1 - Avantages et inconvénients des différentes technologies d'orientation dans l'espace 3D

3.3.3 Cas d'étude

3.3.3.1 Environnement mis en place

L'environnement dont nous disposions initialement n'était pas adapté à notre situation. Il s'agissait d'un carrefour dans lequel circulaient de nombreuses IA. Le principal inconvénient étant le relief, que nous avions décidé de ne pas prendre en considération. Il ne nous était pas non plus nécessaire d'avoir un environnement aussi riche que celui présenté dans le projet (paysage très étendu, forte végétation, signalisation routière, etc.). Nous avons donc élaboré un nouvel environnement consacré à l'intégration du logiciel Robotran.

Générateurs procéduraux de ville

Afin de disposer d'un environnement de test pour notre cas d'étude, nous avons utilisé une librairie nommée :
« Next Gen Modular City V3 [Bundle-Exteriors & Interiors] » [70]

Celle-ci contient des Blueprints spéciaux tels que :

- Procedural Road Generator - PRG
- Procedural Building Generator - PBG.

Le premier nous a permis de générer automatiquement des routes avec des éléments de décors. De plus, sa conception est extrêmement rapide comparé à un environnement ad-hoc réalisé manuellement. Le second a été utilisé pour générer des bâtiments et étoffer le décor afin de le rendre plus réaliste.

Figure 24 - Aperçu de l'environnement créé à partir de PRG et PBG

Figure 25 - Aperçu supérieur de la disposition de la route

Véhicule utilisé

Le projet Digital Twin se compose de nombreuses librairies et modèles 3D. Parmi les véhicules, nous retrouvons des camions, 4x4, citadines, etc. Ceux ayant été utilisés jusqu'à présent dans le projet Digital Twin disposent de la même IA. Notre choix s'est porté sur une Sedan, un véhicule « classique », pour lequel nous avons réussi à collecter les caractéristiques techniques à procurer au logiciel Robotran. La section « Valeurs utilisées » ci-après est consacrée à l'analyse de ces valeurs.

Chaque véhicule se déplace selon une même logique. Le trajet de l'IA est défini par des « splines » [71]. Il s'agit d'une ligne en 3D qui peut être modifiée (allongée, courbée) afin d'obtenir une sorte de chemin balisé. Nous pouvons comparer une spline à une corde que les randonneurs suivent lors d'une randonnée en montagne. La direction dans laquelle le véhicule se déplace est donc définie par cette spline.

Infrastructure de la route

Comme nous pouvons le constater sur l'illustration précédente, l'infrastructure routière est relativement simple. Il s'agit d'une route composée de deux voies de circulations (une dans chaque sens) avec un virage à mi-chemin. L'intention voulue avec cette configuration consiste à ajouter une flaque d'eau dans le virage et constater un changement dans le comportement du véhicule lors de son passage à ce niveau. Par l'altération des valeurs d'adhérence du véhicule par Robotran, nous souhaitons parvenir à engendrer un phénomène de perte de contrôle (glissade). L'objectif étant à nouveau d'obtenir un effet plus proche de la réalité.

3.3.3.2 Valeurs utilisées

Nous avons précédemment décrit les valeurs nécessaires à Robotran pour fonctionner (cf. 3.3.1). Dans cette section, nous allons les aborder d'un point de vue plus pratique en mettant en avant leurs usages.

Valeurs de configuration

Le modèle de véhicule fournit par Robotran requiert des paramètres de configuration au préalable.

```
void setup_car(double wheel_base, double track,
               double mass, double com_x, double com_y, double com_z,
               double Ixx, double Iyy, double Izz,
               double mass_wheel,
               double k_front_susp, double k_rear_susp);
```

Figure 26 - Fonction de configuration du véhicule

Dans un premier temps, nous avons récolté le plus d'information possible concernant notre modèle de Sedan dans UE pour en faire une fiche technique. Celle-ci est consultable dans l'Annexe 3 de ce document. Après une discussion avec l'équipe Robotran, nous avons pu observer que la majeure partie des valeurs obtenues était raisonnable. Le modèle de véhicule UE pouvant être paramétré selon les envies de l'utilisateur, une vérification était plus que bienvenue quant aux caractéristiques physiques utilisées par ce dernier. Le choix des valeurs assignées aux suspensions a été laissé aux soins de l'équipe Robotran.

Inspectons ces valeurs :

Paramètre	**Signification**
Wheel_base	Aussi appelé « empattement », il s'agit de la distance longitudinale entre les roues.
Track	Aussi appelé « voie », il s'agit de la distance latérale entre les roues.
Mass	Masse du véhicule.
Com_x	Position du centre de masse du châssis sur l'axe x par rapport au point de référence du véhicule.
Com_y	Position du centre de masse du châssis sur l'axe y par rapport au point de référence du véhicule.
Com_z	Position du centre de masse du châssis sur l'axe z par rapport au point de référence du véhicule.
Ixx	Moment d'inertie sur l'axe x.
Iyy	Moment d'inertie sur l'axe y.
Izz	Moment d'inertie sur l'axe z.
Mass_wheel	Masse d'une roue.
K_front_susp	Raideur des suspensions avant.
K_rear_susp	Raideur des suspensions arrière.

Tableau 2 - Valeurs de configuration du modèle Robotran

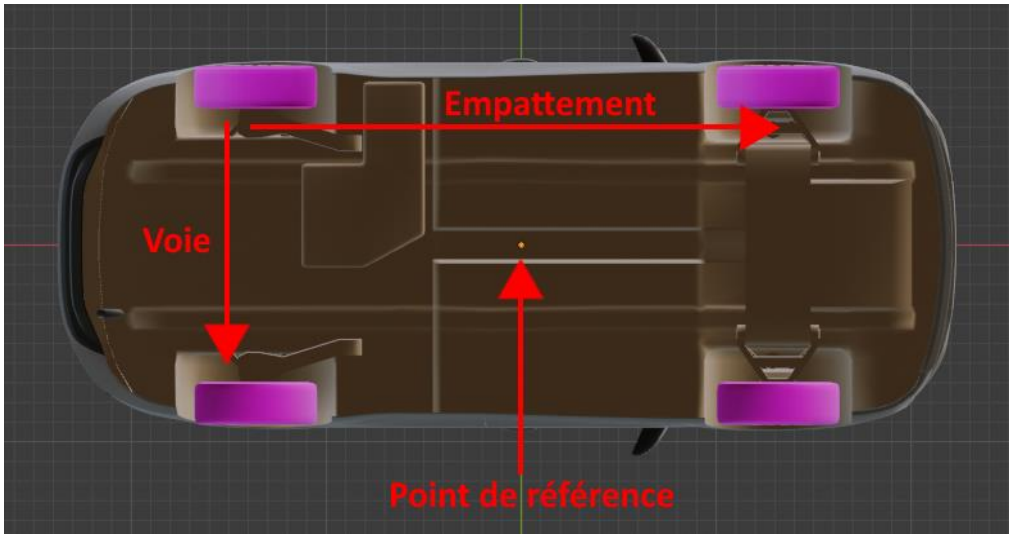


Figure 27 - Illustration de l'empattement et voie d'une Sedan

Valeurs durant l'exécution

Lorsque le modèle Robotran est configuré, nous pouvons alors l'utiliser pour réaliser des calculs de position par l'intermédiaire de la fonction suivante.

```
void compute_next_position(double dt, double steering,
                           double throttle, double brake);
```

Figure 28 - Fonction de calcul de la nouvelle position du véhicule

Les paramètres de cette fonction indiquent le comportement du véhicule au modèle Robotran.

Paramètre	**Signification**
dt	Indicateur du temps auquel nous souhaitons obtenir la nouvelle position. Par exemple, sachant qu'un véhicule se trouve à un endroit x à un temps t, nous souhaitons obtenir sa nouvelle position à t + dt millisecondes dans le futur.
Steering	Position du volant du véhicule.
Throttle	Position de la pédale d'accélération. Une valeur de 0 signifie que l'on ne touche pas à la pédale tandis qu'une valeur de 1 signifie que la pédale est entièrement enfoncée.
Brake	Position de la pédale de frein. Les valeurs sont utilisées de la même façon que pour le Throttle.

Tableau 3 – Valeurs entrantes du modèle Robotran

Nous pouvons également exercer des forces extérieures au véhicule en modifiant la fonction « *user_ExtForces* ».

```
double* user_ExtForces(double PxF[4], double RxF[4][4],
                       double VxF[4], double OMxF[4],
                       double AxF[4], double OMPxF[4],
                       MbsData *s, double tsim, int ixF);
```

Figure 29 - Fonction d'application de force externe au véhicule

Seul un des paramètres en argument de la fonction nous intéresse. Il s'agit de la variable « *PxF* » qui contient la position du centre d'une roue. Cette fonction est donc appelée pour chacune des roues du véhicule simulé. Connaissant la position d'une roue, nous pouvons la comparer à l'environnement de la simulation et plus précisément aux zones glissantes.

La position de la flaque d'eau de notre simulation est fixe. Nous choisissons une coordonnée indiquant le centre de la flaque et sa taille est définie par un rayon autour de ce point central. Avec ces informations (position de la roue et de la flaque), nous pouvons déterminer si la roue se trouve dans la surface que recouvre la flaque d'eau et ainsi y appliquer des forces si nécessaire.

L'application des forces se fait dans le corps de la fonction « *user_ExtForces* ». Nous retrouvons dans celle-ci un appel à une autre fonction nommée « *tgc_bakker_contact* ». Cette fonction calcule les forces de contact entre le pneu et le sol. Les forces résultantes de ce calcul sont stockées dans une variable nommée « *Fwhl_R* ». C'est après cet appel de fonction que nous pouvons ajouter notre code pour modifier les forces de contact (présente dans « *Fwhl_R* ») si le pneu traité par la fonction se trouve sur une surface glissante. Toutefois, ces modifications doivent être effectuées avant les appels de fonctions « *matrix_product* » pour être effectives. Un exemple de modification serait le suivant : *Fwhl_R[1]* = 0, *Fwhl_R[2]* = 0. Ces assignations indiquent que la force longitudinale est nulle (aucun frottement) et qu'il en est de même pour la seconde variable qui représente la force latérale appliquée sur le pneu. Ces valeurs doivent être réduite d'un même facteur pour qu'une cohérence soit gardée.

3.3.3.3 Observations

Maintenant que nous avons clairement analysé chaque possibilité et défini les choix d'implémentation, nous allons pouvoir établir un bilan en comparant les résultats théoriques souhaités et les résultats obtenus en pratique.

Résultats attendus

Le résultat souhaité consistait en une modification de l'emplacement du véhicule et de son orientation. UE envoie les données d'accélération, de direction et de freinage à Robotran qui, sur base de celles-ci, calcule la nouvelle position, orientation, etc. du véhicule à un instant t dans le futur. UE réceptionne les résultats des calculs et les utilise pour définir le nouvel état du véhicule. Cette modification doit s'effectuer à de multiples reprises afin d'observer un déplacement naturel.

Dans la situation où le véhicule entre en contact avec une surface autre que la route (flaque), Robotran doit effectuer ses calculs en altérant les valeurs d'adhérence avec le sol pour calculer les nouvelles valeurs de position, orientation, etc. et ainsi permettre à UE de modifier le déplacement du véhicule afin de donner un effet de glissade/dérapage.

Résultats obtenus

A ce stade, les résultats obtenus n'étaient pas concluants. En premier lieu, les mouvements du véhicule étaient saccadés. Nous n'observions plus un déplacement mais plutôt une téléportation de position en position. A cela s'ajoutait un retour du véhicule à une position précédente ainsi que des changements de direction non désirés. La seconde complication concerne l'évolution de la simulation au cours du temps. Notre intuition nous amenait à supposer un problème de synchronisation entre Robotran et UE. Ce dernier engendrait des valeurs extrêmement anarchiques pour déplacer le véhicule. Quelques secondes suffisaient pour observer ce comportement. Sur base de ces observations, nous avons mené une série de tests analytiques afin de comprendre l'origine de ces complications et ainsi proposer des pistes d'amélioration pertinentes.

3.3.4 Analyse des résultats

Comme énoncé précédemment, les résultats obtenus au terme de la phase de développement ne correspondent pas à ce que nous avions envisagé.

Cette section met en avant différents points qui nous ont permis de définir l'origine du problème. Certains étant davantage théoriques, d'autres étant plus portés sur des aspects pratiques tel que le calcul de performances et de rapidité de communication entre programmes.

3.3.4.1 Conduite

Notre première suspicion concernant les déplacements indésirables s'est portée sur l'IA du véhicule.

Comme expliqué précédemment, les véhicules se déplacent en suivant des splines. En revanche, Robotran n'a pas connaissance de ce mécanisme et ne le prend donc pas en considération lors du calcul de position. C'est à cause de cela que nous observons un comportement dans lequel une force contraint l'IA du véhicule à se rediriger vers la spline afin de rester alignée avec celle-ci. Nous ne sommes donc pas dans une situation où seules les positions fournies par Robotran sont utilisées.

Afin de régler le conflit, nous avons contrôlé le véhicule manuellement avec les flèches directionnelles du clavier. Cela signifie que nous avons extrait une partie du code dont l'IA était composée pour retirer sa dépendance à se mouvoir par rapport à la spline.

Ce test n'avait résolu qu'une partie des problèmes car le phénomène de téléportation du véhicule était quant à lui encore présent.

3.3.4.2 Synchronisation

En seconde piste d'investigation, nous avons porté notre attention sur la manière dont le véhicule s'accordait avec son outil de communication TCP. Comme expliqué précédemment (cf. 3.3.2.5), le véhicule est un acteur dans la simulation et possède une référence pointant vers un autre acteur qui se charge de communiquer avec Robotran (nous appellerons cet autre acteur le « communicateur »). L'ordre logique du processus existant entre ces deux acteurs est le suivant :

1. Le véhicule envoie ses valeurs au communicateur et attend une réponse.
2. Le communicateur envoie ces valeurs à Robotran.
3. Le communicateur reçoit un résultat de Robotran.
4. Le communicateur envoie le résultat au véhicule.
5. Le véhicule traite le résultat et le processus recommence à l'étape 1.

Durant ce processus, le véhicule et le communicateur agissent l'un en fonction de l'autre mais cela ne signifie pas qu'ils ne sont pas actifs lorsqu'une réaction est attendue de la part de l'autre. Le communicateur n'effectue rien de plus que la communication mais le cycle de vie du véhicule quant à lui continue durant le processus.

Une possibilité aurait été de mettre la simulation UE en pause en attendant la réponse de Robotran. Nous avons exploré cette possibilité sans arriver à obtenir un résultat. Cet échec s'explique par la façon dont fonctionne UE et notre communicateur. Ce dernier attend une réponse de la part de Robotran qui va déclencher un de ses événements durant l'exécution de la simulation. Dès l'instant où la simulation est en pause, le communicateur n'est plus actif et ne remarque pas l'arrivée d'une réponse. Nous ne pouvons donc pas savoir quand enlever la pause.

Il est possible de forcer un acteur à exécuter du code à chaque fois qu'une image est affichée même lorsque la simulation est mise en pause mais ce code ne peut fonctionner avec les événements. Étant donné que notre communicateur utilise des événements, cette tentative ne peut aboutir.

Une autre possibilité aurait été d'utiliser un composant externe (ex : un fichier texte) pour marquer la réception de la réponse par Robotran. Ce cas-ci n'a pas été mis en pratique à cause de certains désavantages (temps de lecture du fichier notamment) et du fait que cela ne soit pas une solution conventionnellement acceptable.

Compte tenu des explications ci-dessus, nous avons implémenté le processus de communications de deux manières différentes.

Synchronisation via variable

En premier lieu, nous avons confectionné un mécanisme utilisant une variable en tant que drapeau. Lorsque celle-ci est à *false*, cela signifie que le communicateur n'a pas encore reçu le résultat de la part de Robotran. Inversement, lorsqu'il est à *true*, le véhicule sait que le résultat est disponible. À cet instant, il traite ce résultat et demande au communicateur un nouvel envoi de données à Robotran.

Synchronisation via Event

En second lieu, nous avons utilisé le système d'événement d'UE. Lorsqu'un événement est déclenché par un acteur, les autres acteurs qui sont à l'écoute de ce même événement sont tenus au courant de son déclenchement. Le véhicule déclenche donc un événement pour avertir le communicateur qu'il souhaite envoyer des données et le communicateur déclenche un événement lorsqu'il a reçu le résultat. Cette implémentation est préférable à la précédente car elle implique une réaction presque instantanée contrairement à la variable. Pour cette dernière, il est possible que le véhicule vérifie la variable, que celle-ci soit à *false*, et que le communicateur reçoive le résultat juste après. Le véhicule devra alors attendre d'exécuter son code une nouvelle fois avant de pouvoir utiliser le résultat.

La seconde implémentation (via Event) est en théorie plus performante que la première (via variable). Cependant, en pratique, nous remarquons que le temps économisé est négligeable.

Le résultat de ce test n'est pas non plus concluant concernant notre problème initial de téléportation du véhicule lors de son déplacement.

3.3.4.3 Méthodes de déplacement

A ce stade d'avancement du projet, nous étions hésitant quant à la façon de trouver l'origine du problème. Nous avons alors décidé de prendre contact avec une connaissance externe travaillant dans le développement de jeux vidéo et dont UE est un outil familier. C'est suite à ses conseils que nous avons réalisé les tests présentés dans les sections concernant les méthodes de déplacement et la gestion du temps.

Modification directe de la position

Parmi l'ensemble des données reçues en réponse à une requête envoyée à Robotran, nous retrouvons la nouvelle position du véhicule avec les valeurs pour les axes x, y et z. Nous avions donc tout simplement appliqué ces valeurs directement au véhicule sans manipulation quelconque.

Ajout de valeurs à la position actuelle

Ultérieurement, nous avons changé notre façon de raisonner. Nous avons rajouté une variable nommée « *position to add* ». Cette variable contient la différence entre la nouvelle position du véhicule calculée par Robotran et la position actuelle du véhicule. Par conséquent, nous avons la distance entre les deux localisations pour chacun des axes x, y et z.

A cet instant, nous devons additionner « *position to add* » avec la position actuelle du véhicule pour en récupérer la nouvelle position et l'appliquer. Cette variable est nécessaire pour la suite de nos tests.

Chronologiquement, nous avons d'abord réalisé certains des tests de la section suivante avant de faire la modification concernant « *position to add* ». L'utilité de cette modification peut être constatée dans le paragraphe à venir concernant le déplacement du véhicule à chaque frame.

3.3.4.4 Gestion du temps

La dernière piste que nous avons explorée concerne la manière dont nous gérons le temps entre UE et Robotran.

Actor ticking & frames

Pour comprendre la suite de l'analyse, il est essentiel d'avoir connaissance des mécanismes de « ticking » et de « frames ».

Les jeux vidéo possèdent ce qu'on appelle un « frame rate » [72]. Il est défini comme étant la fréquence à laquelle des images consécutives sont affichées à l'écran. Un frame rate avec une valeur faible donnera une impression de ralentissement voir de saccade dans l'image. Tandis qu'une valeur élevée donnera un aspect lisse et fluide dans l'affichage.

Le frame rate du projet Digital Twin a été fixé à 30 FPS (Frame per Second) maximum. Si nous prenons le véhicule de notre simulation comme exemple, nous pouvons dire qu'il sera affiché 30 fois par seconde. Le frame rate n'est pas fixe. Il peut fluctuer en fonction de la charge de travail réalisée par la simulation ou bien par la machine sur laquelle le projet est exécuté. Une machine avec une meilleure carte graphique peut atteindre un taux de FPS plus élevé.

Chaque acteur dans la simulation UE peut posséder un nœud *Tick* [73 - 74].

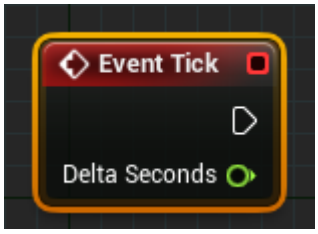


Figure 30 - Noeud tick en BVS

Ce nœud est le point d'entrée du programme de l'acteur qui sera exécuté à chaque frame (la valeur de delta est le temps écoulé depuis le tick précédent. Pour ce qui est de notre véhicule, le code exécuté derrière ce nœud réalise les actions suivantes :

- La mise à jour de la position du véhicule.
- L'envoi d'une requête au communicateur si le résultat de la requête précédente a été reçu.
- Le traitement du résultat de la requête lorsqu'il est reçu.

Pas de temps selon le Tick

Dans un premier temps, nous configurions Robotran pour qu'il nous fournisse la localisation du véhicule au temps t + delta. L'exemple suivant illustre cette situation.

En considérant une simulation s'exécutant à 30 images par seconde, nous estimons un temps delta d'environ 0,033. Si une requête est envoyée au temps t_0, nous demandons la position du véhicule au temps t_0 + 0.033.

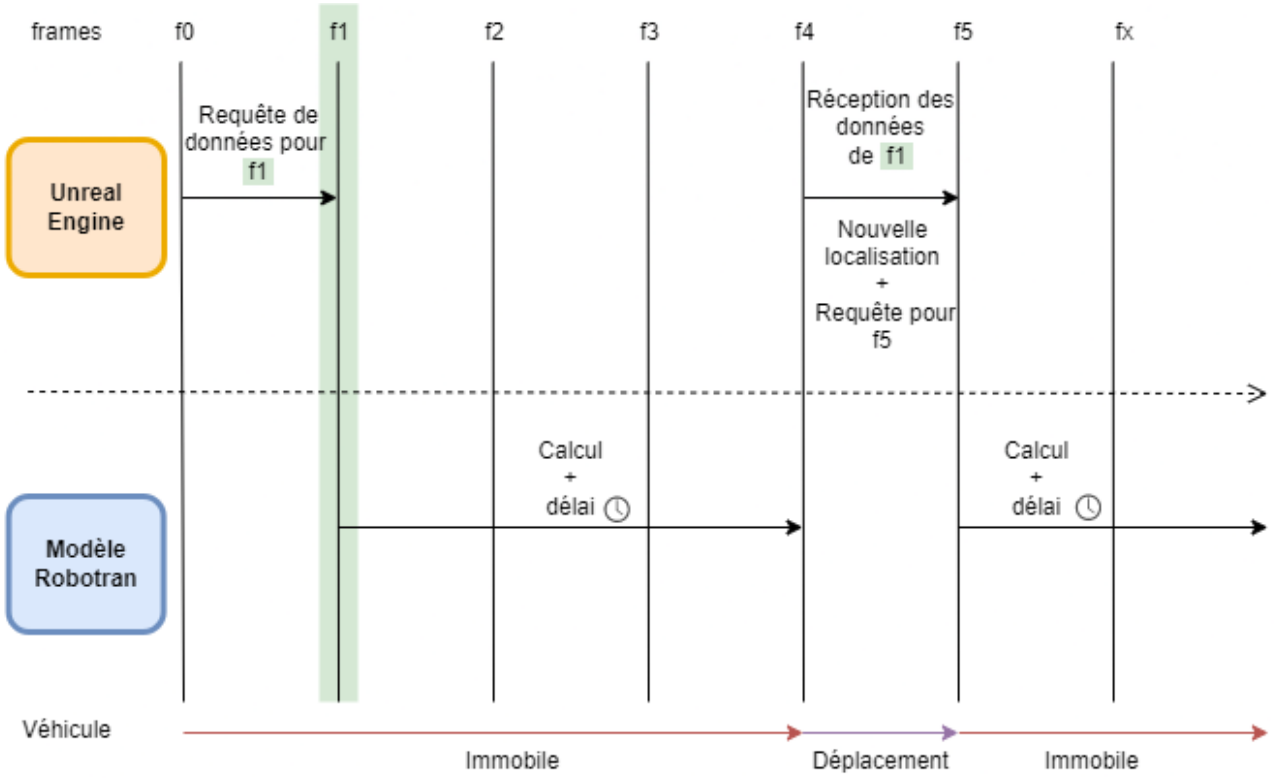


Figure 31 - Pas de temps selon le Tick (ligne du temps)

L'illustration ci-dessus montre que plusieurs frames ont été affichées avant de recevoir le résultat, celui-ci correspondant à une frame déjà passée. Environ trois frames sont nécessaires à la réalisation des calculs de Robotran ainsi que le temps de transmission via les sockets mais aussi le traitement des données reçues.

Le déplacement du véhicule se fait après réception et traitement des données. Nous pouvons voir sur le schéma que le déplacement en tant que tel se fait entre la frame 4 et la frame 5. A partir de la frame 0 jusqu'à la 4, le véhicule est immobile.

Pas de temps selon le nombre de frames écoulées

Afin d'effectuer des requêtes plus sensées, nous demandons à Robotran une nouvelle position pour le véhicule en fonction du nombre de frames écoulées entre l'envoi et la réception des données.

Pour ce faire, nous démarrons un compteur (pour les frames) et envoyons une requête à vide. Lors de la réception de la réponse par Robotran, nous arrêtons le compteur. Toujours avec l'estimation du delta à 0.033, nous multiplions cette valeur par celle du compteur. Si trois frames ont été affichées, nous envoyons alors notre première vraie requête à Robotran avec un pas de temps valant $(compteur * 0.033)$, où la variable « *compteur* » est égale à trois. Le compteur est redémarré à chaque nouvel envoi pour calculer le delta de la requête suivante.

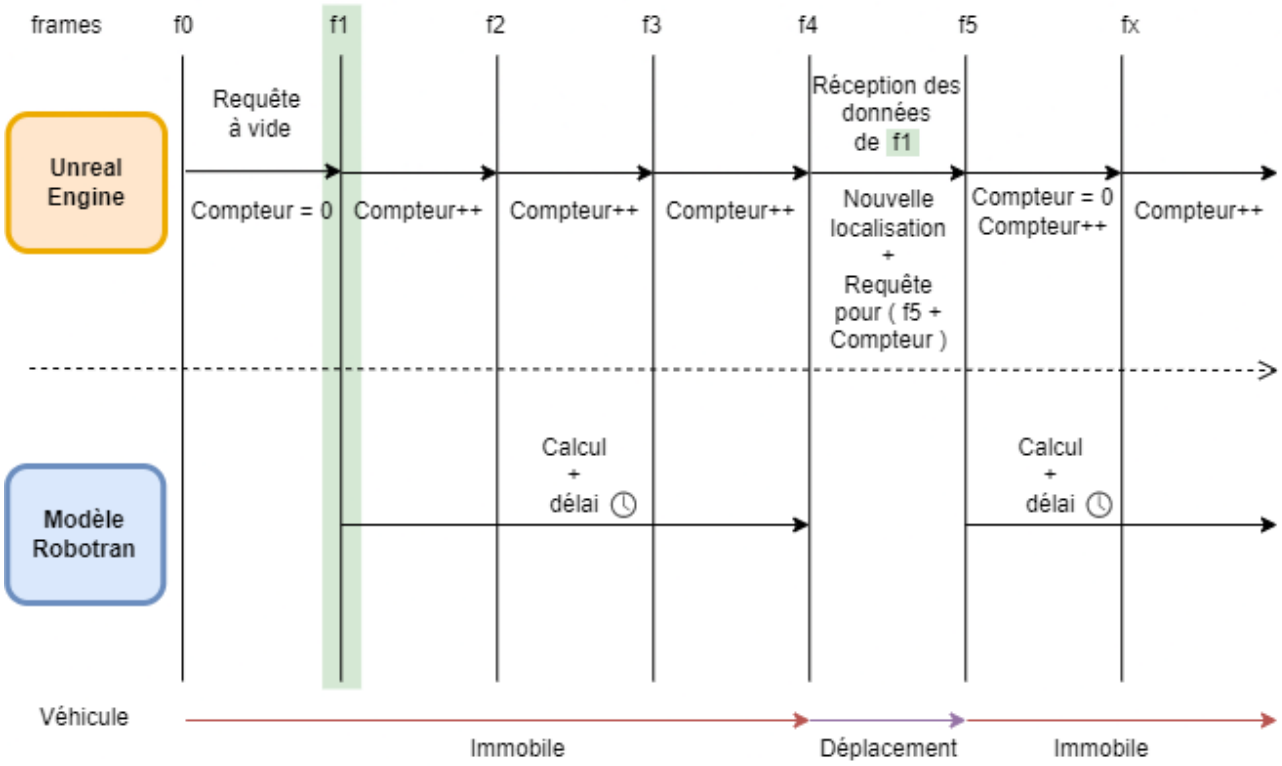


Figure 32 - Pas de temps selon le nombre de frames écoulées (ligne du temps)

Cette méthode permet de réduire approximativement le retard engendré par les calculs et autres délais. Toutefois, elle ne résout pas notre problème initial car le véhicule se déplace toujours de manière saccadée.

Déplacement à chaque frame

En vue d'obtenir un mouvement continu et fluide, nous n'avons pas eu d'autre choix que de déplacer le véhicule à chaque frame.

Nous allons nous baser sur l'exemple suivant pour comprendre les étapes réalisées afin de mettre le véhicule en mouvement : Robotran nous indique que le véhicule doit avancer de 30 cm vers l'avant, 0 cm en latéral et une position verticale inchangée. Nous avons donc un vecteur *deplacement = {30, 0, 0}*. Grâce au compteur que nous avons créé précédemment, nous savons que 3 frames ont été écoulées entre l'envoi de la requête et la réception du résultat. Sachant cela, nous allons diviser le vecteur déplacement par ce compteur et obtenir une nouvelle valeur *deplacement_par_frame = {10, 0, 0}*.

Nous allons donc pouvoir déplacer le véhicule de « *deplacement_par_frame* » à chaque frame en attendant les résultats de la requête suivante. Comme nous pouvons le voir sur le schéma ci-dessous, la flèche mauve indique un déplacement continu.

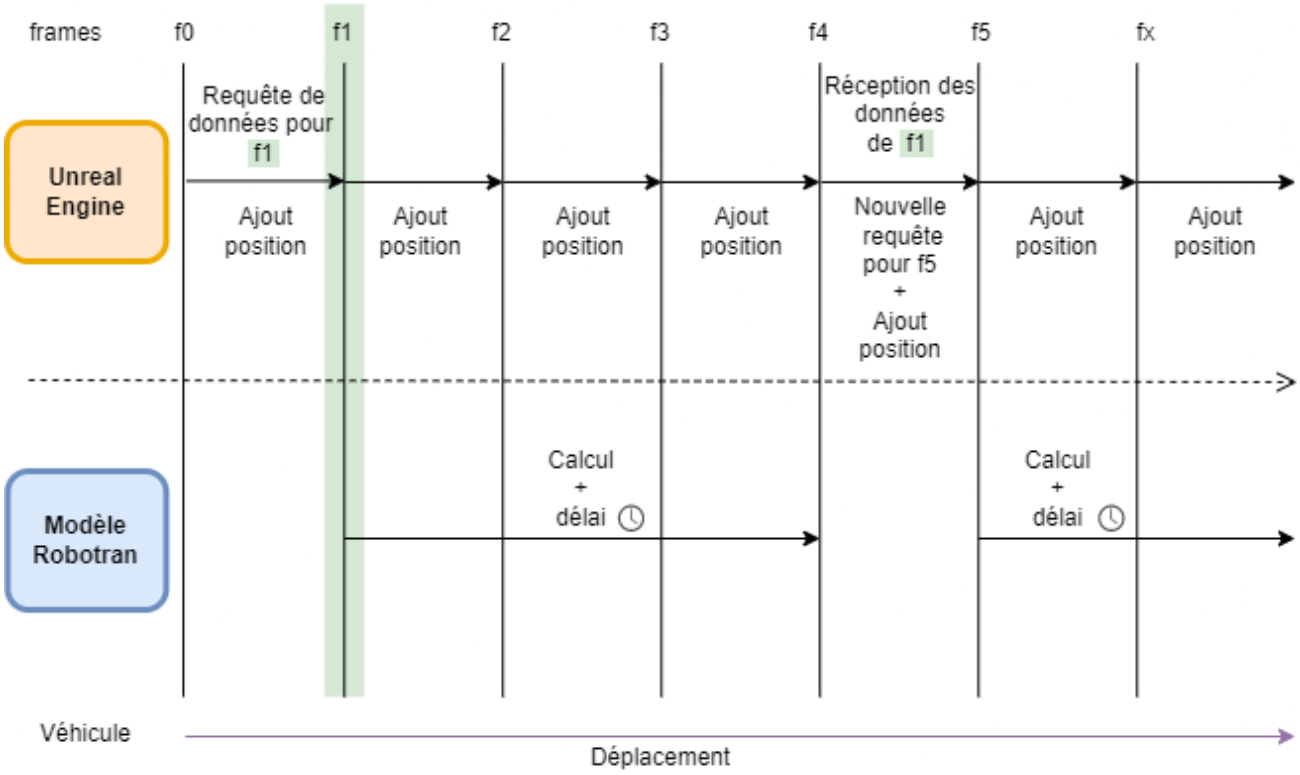


Figure 33 - Déplacement à chaque frame (ligne du temps)

Les résultats de cette méthode furent étranges mais instructifs quant à l'origine du problème. Le phénomène de téléportation n'était plus présent et le véhicule se déplaçait correctement durant une courte période. Passé ce délai, le véhicule avançait très rapidement pour ensuite revenir en arrière. Ce mouvement continuait en prenant de plus en plus d'ampleur jusqu'à envoyer le véhicule en dehors du décor après plusieurs secondes. Nous constations donc une dérive avec un effet boule de neige.

Ce nouvel incident nous a amené à réaliser une nouvelle suite de tests dans l'optique de trouver la source de la dérive. Nous nous sommes lancés dans la réalisation de tests de performances des différentes composantes du projet.

3.3.4.5 Tests de performance

Nous entamons une nouvelle analyse pour découvrir l'origine de l'inconsistance des déplacements du véhicule au cours du temps.

Test sur le parsing

Dans un premier temps, nous avons mesuré la vitesse de traitement des données provenant de Robotran. Ces données, sous le format JSON, doivent être traduites en un ensemble de données manipulables par UE. Pour ce faire, nous avons utilisé le plugin « Json Blueprint » (cf. 3.3.2.6) pour lequel la fiche technique ne contient aucune mention concernant les performances de traitement. Nous avons donc mesuré le temps d'exécution et constaté que celui-ci était pratiquement constant et ne dépassait pas la milliseconde.

Le parsing est le seul gros traitement réalisé. Ce premier test nous indique que l'inconsistance des déplacements du véhicule se trouve à un autre niveau de la chaîne d'exécution. Plus précisément, au niveau de la communication via sockets ou encore Robotran.

Les Sockets

Dans un second temps, nous nous sommes penchés sur le tunnel de communication entre nos deux programmes (cf. 3.3.2.4). L'hypothèse d'avoir un problème à ce niveau nous semblait peu probable. La communication se faisant sur une même machine, il ne devrait pas y avoir de latence possible.

Nous avons toutefois voulu nous en assurer car à nouveau, nous utilisions une librairie pour laquelle nous n'avions pas d'indication concernant les performances.

Nous avons créé un projet UE qui n'effectuait qu'une simple création de client TCP. Le client se connecte à un serveur, envoie une requête, attend le résultat et recommence jusqu'à avoir envoyé 300 requêtes.

Du côté serveur, nous avons conservé le programme Robotran contenant uniquement le code nécessaire à la communication. Il s'agit donc d'un serveur qui répond immédiatement après avoir reçu un message. Le résultat est explicitement indiqué dans la figure suivante :

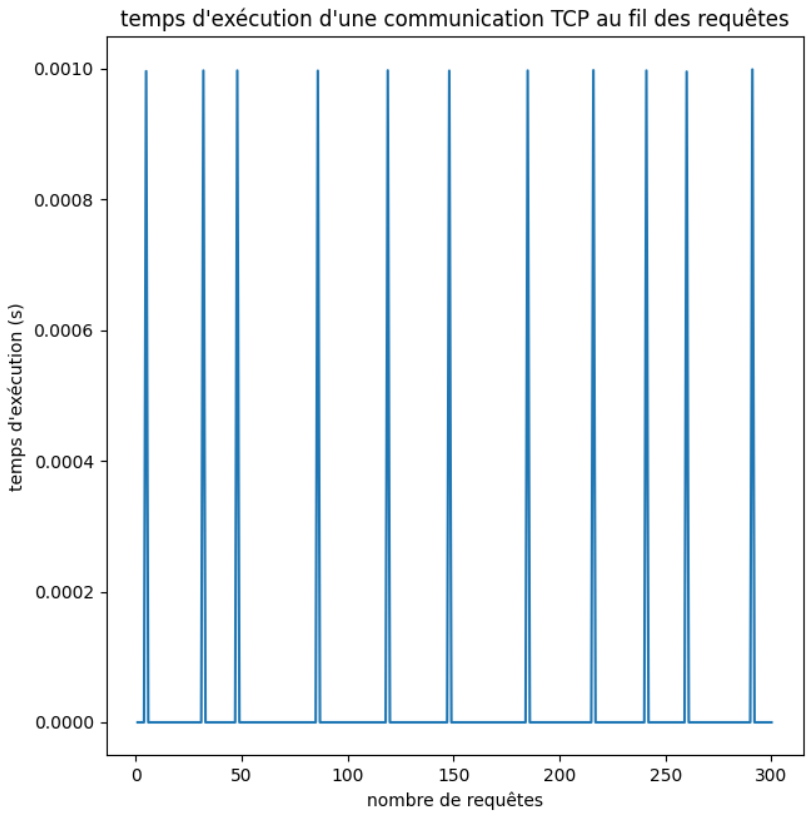


Figure 34 - Temps d'exécution des requêtes TCP

Comme envisagé, la communication via socket n'est pas à l'origine de notre problème. Comme le montre le graphe, l'envoi et la réception d'une réponse ne requiert qu'une seule milliseconde (au maximum). Nous pouvons donc réduire notre champ de recherche à nos deux programmes UE et Robotran.

Digital Twin - Python (TCP)

Ensuite, nous avons mesuré le temps d'exécution des programmes en eux-mêmes avec une communication par socket TCP. Nous aurions raisonnablement pu dire que les sockets n'étaient pas nécessaires dans ce test (l'impact de leur vitesse de communication ayant été prouvé comme étant négligeable dans le point précédent).

Cependant, par sécurité, nous les avons conservés pour nous assurer de ne pas omettre d'erreur telle qu'une surcharge dans la communication (ex : buffer) à la suite de l'ajout des projets.

Ceci est surtout valable pour le test suivant pour lequel une quantité plus importante de données transite au travers du tunnel de communication.

Nous avons débuté avec le projet Digital Twin, représentant le « client ». Celui-ci envoie une requête au « serveur » (simulé par un simple script Python) et attend une réponse. Des mesures de temps sont effectuées juste avant l'envoi de la requête et après le traitement de la réponse. Avec un total de 300 requêtes envoyées, nous avons constaté un temps d'exécution faible et constant (de l'ordre de la milliseconde). De toute évidence, le problème se situait donc du côté de Robotran.

Robotran - Python (TCP)

Enfin, nous avons réalisé le même test que précédemment mais pour l'application Robotran. Cette fois-ci, Digital Twin représente le « client » (pour lequel nous avons adapté le script Python en conséquence) et Robotran le « serveur ».

Comme nous le constatons dans l'illustration ci-dessous, le temps d'exécution augmente de manière linéaire jusqu'à atteindre un cinquième de seconde vers la 300ème requête.

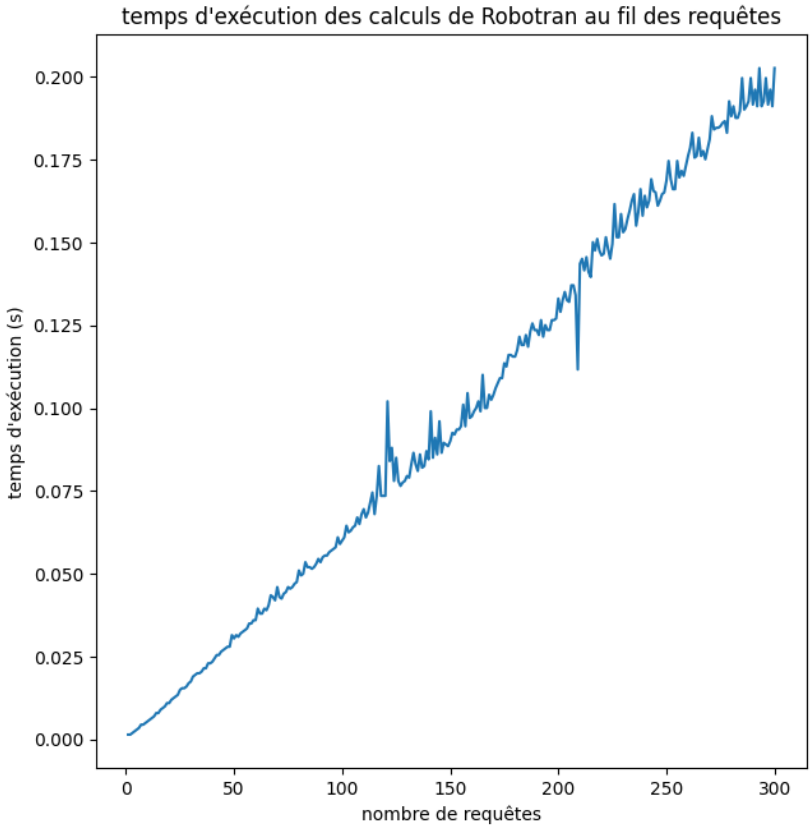


Figure 35 - Temps d'exécution Robotran

Notons également la variation du temps d'exécution en fonction des paramètres fournis à Robotran. Pour l'illustration ci-dessus, nous avons attribué des faibles valeurs en paramètre pour que le déplacement du véhicule soit minime et ainsi alléger les calculs à effectuer. Si l'on attribue des valeurs plus élevées, le temps d'exécution augmente pour chaque requête mais conserve sa croissance linéaire.

Nous avons poursuivi notre analyse en effectuant des mesures de temps au sein du programme Robotran et avons constaté que la fonction *compute_next_position* (calculant la nouvelle position du véhicule pour un intervalle de temps donné) s'exécutait de plus en plus lentement au fil des appels. De nombreux appels de fonction sont exécutés au sein de cette même fonction. En mesurant le temps d'exécution individuel de ces fonctions, nous avons identifié la fonction à l'origine du ralentissement. Il s'agissait d'une fonction nommée *mbs_dirdyn_loop* qui réalisait des calculs sans conserver d'état par rapport au temps courant de la simulation. Le programme conserve le temps écoulé depuis le début de la simulation et les calculs recommençait à l'instant zéro jusqu'à ce temps écoulé.

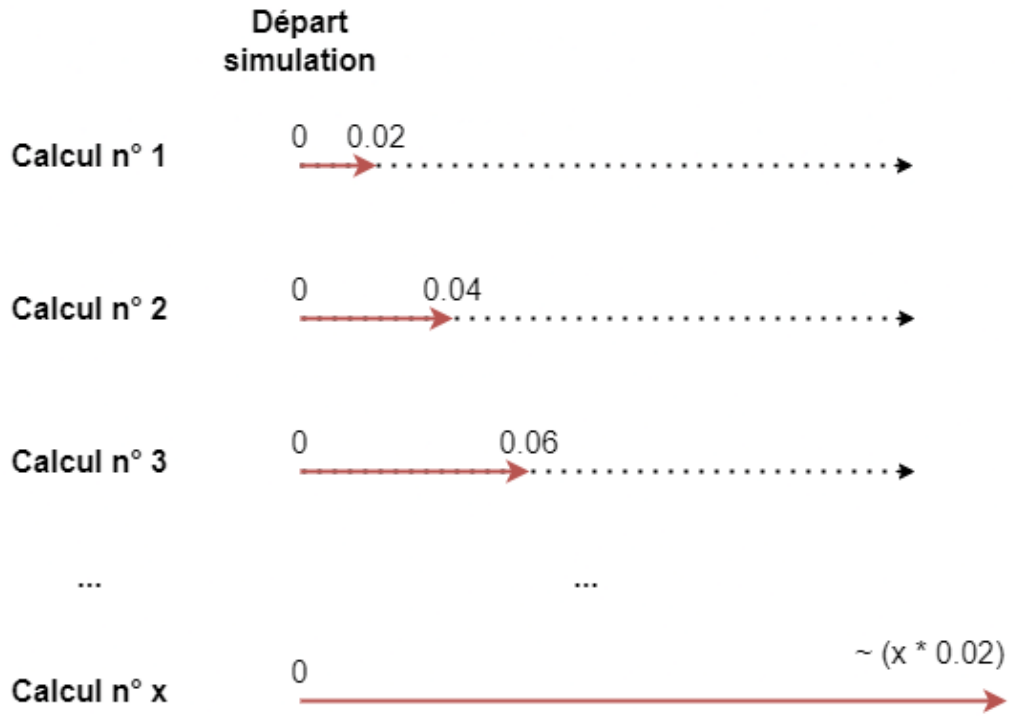


Figure 36 - Représentation de l'exécution des calculs à partir de l'instant zéro

Solution

Après signalement et discussion du problème rencontré avec l'équipe Robotran, nous avons pu trouver une solution à notre problème. Le modèle Robotran a été paramétré différemment. Ainsi, les calculs ne s'exécutent plus à partir de l'instant zéro de la simulation mais à partir de l'instant courant de la simulation. De cette façon, seuls les calculs pour le pas de temps de la nouvelle position du véhicule sont exécutés.

Dès l'instant où ce problème a été résolu, nous avons constaté une nette amélioration dans la simulation UE. Notre véhicule se déplaçait de manière fluide et sans générer d'effet indésirable au cours du temps.

4 Conclusion

Cette section permet de conclure le travail réalisé tout au long de cette année académique et ainsi de répondre à la question de recherche présentée dans l'introduction de ce document.

Tout d'abord, nous y présentons un résumé des différentes étapes de production de notre thèse. Ce sont ces étapes qui nous permettent d'apporter des éléments de réponses à la question portant sur l'amélioration d'une simulation dans le but de la rendre plus réaliste.

Ensuite, nous abordons différentes pistes d'amélioration qui constituent de futures itérations afin de compléter le travail réalisé dans le cadre de ce mémoire.

Enfin, nous fournissons une analyse introspective de notre contribution à ce projet. Il s'agit d'une étape importante car elle nous permet de nous rendre compte de nos qualités et défauts quant au travail effectué.

4.1 Résumé

Nous avons tout d'abord présenté les différents éléments mis en place afin d'organiser le travail de la meilleure façon possible. Dans un premier temps, les outils de versionnage utilisés, à savoir Perforce P4V pour le projet Digital Twin et Github pour le projet Robotran. Ensuite, en termes de communication tout au long du projet, nous avons procédé à des réunions régulières avec les équipes en charge de Digital Twin et de Robotran afin de leur présenter continuellement nos états d'avancement sous forme de comptes rendus. Enfin, concernant la gestion du projet, nous avons adopté certaines méthodologies agiles afin d'avoir une vision claire des tâches à accomplir et ainsi proposer la meilleure efficacité possible. Notamment l'outil Jira permettant de définir et attribuer les tâches, et le pair programming lorsqu'il s'agissait du travail en binôme.

Ensuite, la première partie du mémoire qui concerne la prise en main et l'amélioration du projet Digital Twin. Dans un premier temps, nous avons exploré une première piste qu'est le simulateur Carla pour en évaluer le potentiel d'intégration de Digital Twin. Cependant, nous l'avons abandonné car bien qu'étant complet, Carla nécessite trop de ressources physiques, ce qui rend son utilisation difficile. Nous avons donc pris en main le projet Digital Twin, pour lequel nous avons tout d'abord procédé à l'amélioration des performances en travaillant sur des métriques telles que la distance d'affichage, le concept de « shader complexity » ainsi que les éléments relatifs aux ombres de l'environnement. Pour terminer, nous avons amélioré son réalisme sur certains points présents dans la simulation, notamment la végétation et les routes. La couleur des véhicules est également un élément de réalisme auquel nous avons porté une attention particulière.

Pour terminer, la seconde partie de notre mémoire concerne l'intégration de Robotran afin d'améliorer la physique des véhicules et ainsi augmenter le réalisme de la simulation. Dans un premier temps, nous y présentons les interactions théoriques montrant les données d'entrée et de sortie que nécessite un modèle Robotran afin d'être intégré. Ensuite, nous abordons son intégration de façon pratique, en présentant notamment la façon dont nous avons formaté les données afin de faciliter leur traitement. De même, nous avons mené un travail de recherche concernant l'orientation d'un corps dans un espace 3D et avons démontré pourquoi, parmi toutes les possibilités, les quaternions sont les plus adéquats. Ensuite, nous nous sommes servis de générateurs procéduraux de routes et de bâtiments pour mettre en place un environnement permettant de procéder aux tests d'intégration. Par après, nous avons utilisé les données sortantes de l'application Robotran afin de paramétrer notre véhicule et ainsi procéder aux premières phases d'observation. Nous avons découvert un problème qui causait des comportements indésirables concernant notre véhicule. Celui-ci présentait des déplacements saccadés et nous avons donc procédé à une série de tests pour en expliquer la cause. C'est ainsi que nous avons remarqué qu'une erreur d'implémentation avait été commise dans le code qui nous a été fourni. Celui-ci ayant été rapidement corrigé grâce à la collaboration efficace de l'équipe en charge de Robotran, nous avons finalement pu définir les prochaines étapes à suivre afin d'obtenir des résultats satisfaisants.

Ainsi, nous pouvons apporter des éléments de réponse à la question présentée dans l'introduction de ce document. Pour améliorer une simulation afin de la rendre plus réaliste, deux pistes ont donc été envisagées. La première consiste à manipuler les métriques disponibles au sein d'UE ainsi qu'à adapter certains éléments de la simulation afin de les rendre plus cohérents avec la réalité. La seconde consiste à enrichir la physique des véhicules par le biais d'un outil de gestion de systèmes multicorps tel que Robotran.

4.2 Pistes d'amélioration

Le projet que nous avons fourni nécessitera un travail supplémentaire avant de pouvoir être utilisé comme outil final à des fins de génération de données, de simulateur de conduite, etc. Celui-ci constitue cependant une base solide pouvant être améliorée grâce à certains des points suivants.

4.2.1 Déplacement du véhicule

La recherche de solutions aux différents inconvénients rencontrés a fortement impacté l'avancée globale du projet. Cela nous a notamment empêché d'atteindre les résultats escomptés en ce qui concerne le déplacement du véhicule.

Tout d'abord, le fait de générer un effet de glissade lors du passage du véhicule sur une surface glissante. La marche à suivre pour atteindre ce résultat a été abordée dans une précédente section (cf. 3.3.3.2). Une future première étape serait donc d'implémenter et tester ce comportement au sein de notre cas d'étude.

De plus, nous avons constaté l'apparition d'un nouveau comportement étrange. Avant de résoudre le problème de téléportation, le véhicule avançait de manière saccadée mais sur des distances mesurables en mètres. Dans la version corrigée, le véhicule ne se déplace plus que de quelques centimètres à chaque intervalle de temps. Ce comportement étrange persiste également lorsque l'on force l'accélération du véhicule à sa valeur maximum. Une future seconde étape consisterait donc à trouver l'origine de cette anomalie et la corriger afin que le véhicule se déplace correctement.

4.2.2 Intelligence artificielle

Nous avons également remarqué que l'IA utilisée ne convient pas tout à fait à l'emploi que nous en faisons dans ce cas d'étude. Cependant, quelques modifications nous ont permis de l'utiliser mais il faudra toutefois songer à recourir à une autre IA ou bien y apporter des changements considérables pour atteindre un résultat concluant.

Cette nouvelle IA devra pouvoir déterminer toutes les actions possibles et les valeurs qui en résultent sans pour autant les mettre en application. Cette réflexion se basait initialement sur le mécanisme des splines (cf. 3.3.3.1) mais celui-ci engendrait des déplacements non désirés. Dans le cadre de l'intégration avec Robotran, nous avons supprimé ce mécanisme et l'avons remplacé en imposant des valeurs au véhicule (accélération, frein, etc.).

Il faudra considérer l'emploi d'un autre modèle de véhicule pour l'IA. En effet, le châssis et les roues actuellement utilisés forment un bloc unique et ne sont donc pas représentés par des composants indépendants. Dissocier les différentes parties du véhicule permettrait d'influencer aisément les roues avec les valeurs de Robotran, notamment afin de changer leur position ou leur orientation.

4.2.3 Scalabilité

Dans cette section concernant la scalabilité, nous allons mettre en avant les différents points à prendre en considération afin de pouvoir utiliser ce projet à une plus large échelle. Nous entendons par là, avoir une simulation contenant de multiples véhicules.

4.2.3.1 Véhicules

L'ajout de véhicules en lui-même n'est pas une tâche complexe. Il existe déjà des mécanismes, au sein du projet Digital Twin, faisant apparaître plusieurs acteurs (véhicules ou autres). Ces mécanismes peuvent s'appliquer à notre cas d'étude, dans lequel les véhicules se connecteront au serveur (Robotran) pour effectuer ses requêtes.

4.2.3.2 Robotran

Le projet Robotran actuel se comporte comme un serveur qui attend la connexion d'un client pour répondre à ses requêtes. Afin d'élargir son champ d'actions à plusieurs véhicules, le serveur devra être capable d'accepter plusieurs connexions TCP. Une des façons de gérer cela est de démarrer un « thread » [75] pour chacun des clients qui se connectent au serveur. Ce concept de « multi-threading » [76] permettra à chaque véhicule de communiquer avec le serveur qui lui pourra continuer d'attendre les connexions de nouveaux clients afin de leur allouer un thread à leur tour.

4.2.4 Réalisme

Dans une version plus avancée du projet, il serait possible d'ajouter un supplément de réalisme grâce à différentes propriétés de l'environnement offertes par Robotran. Parmi celles-ci, nous retrouvons la force exercée par le vent ou encore le dénivelé du terrain. Ces éléments influencent la physique du véhicule en fonction de leur valeur et seront pris en compte dans les calculs de position de Robotran.

Le relief des routes pourrait également être pris en compte afin de visualiser les forces qui s'exercent sur les suspensions des véhicules. Jusqu'à présent, l'environnement que nous avons utilisé était entièrement plat et la physique des suspensions était prise en charge par UE.

4.3 Analyse Introspective

Lors de notre recherche de travail de fin d'étude, nous avons choisi ce sujet car il représentait un bon moyen de nous mettre au défi avec un projet conséquent et dont la thématique offre des perspectives qui nous intéressent.

Le plus grand défi a été le travail réalisé durant le second quadrimestre. L'intégration du projet Robotran en tant que tel n'a pas été pas trop complexe. En revanche, la synchronisation des deux programmes nous a poussé à faire des analyses plus avancées et à nous entretenir avec diverses personnes pour continuer à aller de l'avant.

Notre esprit d'équipe nous a permis de mettre en place une communication efficace au sein de notre binôme et avec les différentes équipes ayant contribués à cette thèse. Nous sommes persuadés que la discipline dont nous avons fait preuve à ce niveau nous a permis d'améliorer la qualité de notre travail.

Bien que les objectifs initialement fixés n'aient pas été atteints, nous sommes fiers de constater que le fruit de notre travail soit le point de départ d'un projet présentant une capacité d'évolution si conséquente.

Le travail réalisé au fil de cette thèse nous a confronté à de nombreux imprévus. La procédure permettant de les résoudre a engendré un retard non négligeable dans notre avancée. C'est notamment grâce à l'avis de personnes externes ayant un meilleur recul sur le projet, que nous avons réussi à en fournir l'analyse complète et précise présentée dans ce manuscrit.

Ce travail nous a apporté de nombreux enseignements. Nous avons notamment appris que la source d'un problème se trouve parfois à des endroits qu'on ne suspecte pas. Cette leçon nous sera très bénéfique dans la mesure où elle nous permettra de poser un regard différent sur les futurs obstacles que nous serons amenés à rencontrer.

5 Bibliographie

[1] IBM. (s.d.). *Qu'est-ce qu'un jumeau numérique ?* . Consulté en 2022 sur : https://www.ibm.com/fr-fr/topics/what-is-a-digital-twin

[2] Unreal Engine. (s.d.). *Unreal Engine Features* . Consulté en 2022 sur : https://www.unrealengine.com/en-US/features

[3] Unreal Engine. (s.d.). *Unreal Engine Release Notes* . Consulté en 2022 sur : https://docs.unrealengine.com/4.27/en-US/WhatsNew/Builds/

[4] Unreal Engine. (s.d.). *Blueprints Visual Scripting* . Consulté en 2022 sur : https://docs.unrealengine.com/5.0/en-US/blueprints-visual-scripting-in-unreal-engine/

[5] Shaip. (s.d.). *Annotation des données et étiquetage des données* . Consulté en 2022 sur : https://fr.shaip.com/blog/the-a-to-z-of-data-annotation/

[6] Robotran. (s.d.). *Robotran Features* . Consulté en 2022 sur : https://www.robotran.be/features/

[7] Robotran. (s.d.). *Modeling Multibody Systems with ROBOTRAN* . Consulté en 2022 sur : https://robotran-doc.git-page.immc.ucl.ac.be/RobotranBasic/Robotran_basics.pdf

[8] Eloïse Salson. (2022, 19 Jan). *Suivez vos modifications à la trace avec les 8 meilleurs logiciels de versioning* . Consulté en 2022 sur : https://www.appvizer.fr/magazine/services-informatiques/gestion-versions/outils-versionning

[9] Github. (s.d.). *Github Features* . Consulté en 2022 sur : https://github.com/features

[10] Atlassian. (s.d.). *Git LFS* . Consulté en 2022 sur : https://www.atlassian.com/git/tutorials/git-lfs

[11] Perforce. (s.d.). *Helix Visual Client (P4V) for Helix Core* . Consulté en 2022 sur : https://www.perforce.com/products/helix-core-apps/helix-visual-client-p4v

[12] Gitlab. (s.d.). *Gitlab Features* . Consulté en 2022 sur : https://about.gitlab.com/features/?stage=plan

[13] Ionos. (2020, 06 Oct). *GitLab vs. GitHub : comparatif des deux systèmes de contrôle de version* . Consulté en 2022 sur : https://www.ionos.fr/digitalguide/sites-internet/developpement-web/gitlab-vs-github/

[14] Manager-go. (2022, 02 Feb). *L'essentiel sur les méthodes Agiles* . Consulté en 2022 sur : https://www.manager-go.com/gestion-de-projet/methodes-agiles.htm

[15] Atlassian. (s.d.). *Fonctionnalités de Jira Software* . Consulté en 2022 sur : https://www.atlassian.com/fr/software/jira/features

[16] Digite. (s.d.). *Introduction to Extreme Programming (XP)* . Consulté en 2022 sur : https://www.digite.com/agile/extreme-programming-xp/

[17] Alexander S. Gillis. (s.d.). *pair programming* . Consulté en 2022 sur : https://www.techtarget.com/searchsoftwarequality/definition/Pair-programming

[18] CARLA Simulator. (s.d.). *CARLA Introduction* . Consulté en 2022 sur : https://carla.readthedocs.io/en/latest/start_introduction/

[19] CARLA Simulator. (s.d.). *Generating Maps in RoadRunner* . Consulté en 2022 sur : https://carla.readthedocs.io/en/latest/tuto_M_generate_map/

[20] CARLA Simulator. (s.d.). *Sensors reference* . Consulté en 2022 sur : https://carla.readthedocs.io/en/latest/ref_sensors/

[21] CARLA Simulator. (s.d.). *LIDAR sensor* . Consulté en 2022 sur : https://carla.readthedocs.io/en/latest/ref_sensors/#lidar-sensor

[22] CARLA Simulator. (s.d.). *RGB camera* . Consulté en 2022 sur : https://carla.readthedocs.io/en/latest/ref_sensors/#rgb-camera

[23] CARLA Simulator. (s.d.). *Depth camera* . Consulté en 2022 sur : https://carla.readthedocs.io/en/latest/ref_sensors/#depth-camera

[24] CARLA Simulator. (s.d.). *Radar sensor* . Consulté en 2022 sur : https://carla.readthedocs.io/en/latest/ref_sensors/#radar-sensor

[25] CARLA Simulator. (s.d.). *ScenarioRunner* . Consulté en 2022 sur : https://carla-scenariorunner.readthedocs.io/en/latest/

[26] StringFixer. (s.d.). *Licence MIT* . Consulté en 2022 sur : https://stringfixer.com/fr/MIT_license

[27] UltraBend. (s.d.). *Rendu temps réel et optimisation* . Consulté en 2022 sur : https://www.ultrabend.org/blog/rendu-temps-reel-unreal-engine/

[28] Kimmo Kaunela. (s.d.). *Creating immersive VR expériences Part 2* . Consulté en 2022 sur : https://www.artstation.com/kimmokaunela/blog

[29] Yuriy Denisyuk. (2021, 18 Nov). *Top 4 Unreal Engine Tools For Optimizing Game Development* . Consulté en 2022 sur : https://pinglestudio.com/blog/full-cycle-development/top-4-unreal-engine-tools-for-optimizing-game-development

[30] CodeLikeMe. (2022, 22 Dec). *Unreal Engine Performance Optimization* . Consulté en 2022 sur : https://www.youtube.com/watch?v=FQxriR4_akM

[31] Unreal Engine. (2020, 30 Oct). *Performance Optimization for Environments | Inside Unreal* . Consulté en 2022 sur : https://www.youtube.com/watch?v=ZRaeiVAM4LI

[32] Mathew Wadstein. (2016, 20 Oct). *View Distance Quality in Unreal Engine 4 (UE4)* . Consulté en 2022 sur : https://www.youtube.com/watch?v=fQsBuyc1qRI

[33] Clay Slover. (s.d.). *UE4: Optimization & Performance | Pt.2 – LODs* . Consulté en 2022 sur : https://www.artstation.com/blogs/samslover/QwNE/ue4-optimization-performance-pt-2-lods

[34] armDeveloper. (s.d.). *Geometry best practices for Unreal Engine* . Consulté en 2022 sur : https://developer.arm.com/documentation/102695/0100/Triangle-and-polygon-usage

[35] Ryan Manning. (2019, 15 Feb). *UE4 - Generating LODs & Creating Custom LOD Groups* . Consulté en 2022 sur : https://www.youtube.com/watch?v=li5qraDIZIM

[36] Ben Cloward. (2019, 21 Nov). *What Is A Shader? UE4 Materials 101 - Episode 1* . Consulté en 2022 sur : https://www.youtube.com/watch?v=uQG0SWv5lbw

[37] armDeveloper. (s.d.). *Material and Shader Best Practices for Unreal Engine* . Consulté en 2022 sur : https://developer.arm.com/documentation/102676/0100/Profile-and-visualize-shader-complexity

[38] Unreal Engine. (s.d.). *View Modes* . Consulté en 2022 sur : https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LevelEditor/Viewports/ViewModes/

[39] GDi4K. (2021, 21 Apr). *Unreal Engine Landscape Performance : Shader Complexity vs Triangle Count (with LOD Distribution)* . Consulté en 2022 sur : https://www.youtube.com/watch?v=HkjBYfnr3co

[40] Polygon University. (2019, 18 Jul). *How to improve your shadow quality in UE4 - Lightmap Resolution Explained* . Consulté en 2022 sur : https://www.youtube.com/watch?v=I2kl-dcPtwI

[41] Nick Mower. (s.d.). *Lighting Companion: Light Mobility in UE4 Explained* . Consulté en 2022 sur : https://www.techarthub.com/lighting-companion-light-mobility-in-ue4-explained/

[42] Ryan Manning. (2019, 15 Mar). *UE4 Lightning Types* . Consulté en 2022 sur : https://www.youtube.com/watch?v=2UowdJetXwA

[43] Géraldine Gaudy. (2020, 17 Dec). *Couleur de voiture. Le palmarès 2020 des teintes les plus populaires* . Consulté en 2022 sur : https://www.largus.fr/actualite-automobile/couleur-de-voiture-le-palmares-2020-des-teintes-les-plus-populaires-10494405.html

[44] GeeksForGeeks. (s.d.). *Binary Search* . Consulté en 2022 sur : https://www.geeksforgeeks.org/binary-search/

[45] Tvaira Free. (s.d.). *Programmation Réseau : Socket TCP/UDP* . Consulté en 2022 sur : http://tvaira.free.fr/bts-sn/reseaux/cours/cours-sockets.pdf

[46] Lawrence Williams. (2022, 16 Apr). *TCP vs UDP: Key Difference between TCP and UDP Protocol* . Consulté en 2022 sur : https://www.guru99.com/tcp-vs-udp-understanding-the-difference.html

[47] SpartanTools. (2019, 14 May). *TCP Socket Plugin* . Consulté en 2022 sur : https://www.unrealengine.com/marketplace/en-US/product/tcp-socket-plugin

[48] MDN Web Docs. (2022, 16 Mar). *Introduction à XML* . Consulté en 2022 sur : https://developer.mozilla.org/fr/docs/Web/XML/XML_introduction

[49] MDN Web Docs. (2022, 16 Mar). *Manipuler des données JSON* . Consulté en 2022 sur : https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/JSON

[50] RedHat. (2021, 18 June). *YAML, qu'est-ce que c'est ?* . Consulté en 2022 sur : https://www.redhat.com/fr/topics/automation/what-is-yaml

[51] Maksim Shestakov. (2018, 22 Sept). *Json Blueprint* . Consulté en 2022 sur : https://www.unrealengine.com/marketplace/en-US/product/json-blueprint

[52] Christian Féron. (s.d.). *A quoi sert le parsing ?* . Consulté en 2022 sur : http://chrisferon.free.fr/programmation/bases-parsing.php

[53] StringFixer. (s.d.). *système de coordonnées cartésiennes* . Consulté en 2022 sur : https://stringfixer.com/fr/Cartesian_plane

[54] TechnoChouette. (2020, 07 Oct). *Jeux 2D vs jeux 3D: quelles sont les différences ?* . Consulté en 2022 sur : https://technochouette.istocks.club/jeux-2d-vs-jeux-3d-quelles-sont-les-differences/2020-10-07/

[55] StringFixer. (s.d.). *Règle de la main droite* . Consulté en 2022 sur : https://stringfixer.com/fr/Right_hand_rule

[56] BlenderBaseCamp. (s.d.). *What Are The Main Features Of The Blender Software?* . Consulté en 2022 sur : https://blenderbasecamp.com/home/what-are-the-main-features-of-the-blender-software/

[57] aviator. (s.d.). *Roll Pitch Yaw (axes and maneuvers)* . Consulté en 2022 sur : https://www.aviationfile.com/roll-pitch-yaw-axes/

[58] UTexas. (s.d.). *Rotations and Orientation* . Consulté en 2022 sur : https://www.cs.utexas.edu/~theshark/courses/cs354/lectures/cs354-14.pdf

[59] Moti Ben-Ari. (s.d.). *A Tutorial on Euler Angles and Quaternions* . Consulté en 2022 sur : https://www.weizmann.ac.il/sci-tea/benari/sites/sci-tea.benari/files/uploads/softwareAndLearningMaterials/quaternion-tutorial-2-0-1.pdf

[60] Woolfrey. (2018, 03 Aug). *2.3 Rotations in 3D* . Consulté en 2022 sur : https://www.youtube.com/watch?v=wg9bI8-Qx2Q

[61] StringFixer. (s.d.). *Angles d'Euler* . Consulté en 2022 sur : https://stringfixer.com/fr/Relative_orientation

[62] Matthew Brett. (s.d.). *Gimbal Lock* . Consulté en 2022 sur : https://matthew-brett.github.io/transforms3d/gimbal_lock.html

[63] Fabio Garcia Rojas. (2015, 28 Apr). *Gimbal Lock* . Consulté en 2022 sur : https://www.youtube.com/watch?v=Mm8tzzfy1Uw

[64] StringFixer. (s.d.). *Théorème de rotation d'Euler* . Consulté en 2022 sur : https://stringfixer.com/fr/Euler's_rotation_theorem

[65] Wikipedia. (s.d.). *Axis-angle representation* . Consulté en 2022 sur : https://en.wikipedia.org/wiki/Axis%E2%80%93angle_representation

[66] StringFixer. (s.d.). *Matrice de rotation* . Consulté en 2022 sur : https://stringfixer.com/fr/Infinitesimal_rotation

[67] 3Blue1Brown. (2022, 06 Sept). *Que sont les quaternions, et comment les visualiser ? Une histoire en quatre dimensions* . Consulté en 2022 sur : https://www.youtube.com/watch?v=d4EgbgTm0Bg

[68] 3Blue1Brown. (2018, 26 Oct). *Quaternions and 3d rotation, explained interactively*. Consulté en 2022 sur : https://www.youtube.com/watch?v=zjMuIxRvygQ

[69] D. Rose. (s.d.). *Rotation Quaternions, and How to Use Them* . Consulté en 2022 sur : https://danceswithcode.net/engineeringnotes/quaternions/quaternions.html

[70] Master HiPoly. (2017, 20 Dec). *Next Gen Modular City V3 [Bundle-Exteriors & Interiors]* . Consulté en 2022 sur : https://www.unrealengine.com/marketplace/en-US/product/modular-city

[71] Vintz. (2019, 06 Aug). *Unreal Engine 4 | Les splines* . Consulté en 2022 sur : https://www.youtube.com/watch?v=tvjzfhsnASM

[72] NFI. (s.d.). *Frame Rate – Everything You Need To Know* . Consulté en 2022 sur : https://www.nfi.edu/frame-rate/

[73] underscore. (2021, 30 Mar). *UE4 Tutorial: Event Tick* . Consulté en 2022 sur : https://www.youtube.com/watch?v=ZaKzy9WzsGw

[74] Jay Versluis. (2022, 10 Jan). *Creating a Tick Event in the Unreal Engine Editor (Blueprint)* . Consulté en 2022 sur : https://www.versluis.com/2022/01/ue4-editor-tick/

[75] Techno-Science. (s.d.). *Thread (informatique) - Définition et Explications* . Consulté en 2022 sur : https://www.techno-science.net/glossaire-definition/Thread-informatique.html

[76] Ionos. (2021, 13 Sep). *Le multithreading : plus de puissance pour les processeurs* . Consulté en 2022 sur : https://www.ionos.fr/digitalguide/serveur/know-how/le-multithreading-explique/

Annexes

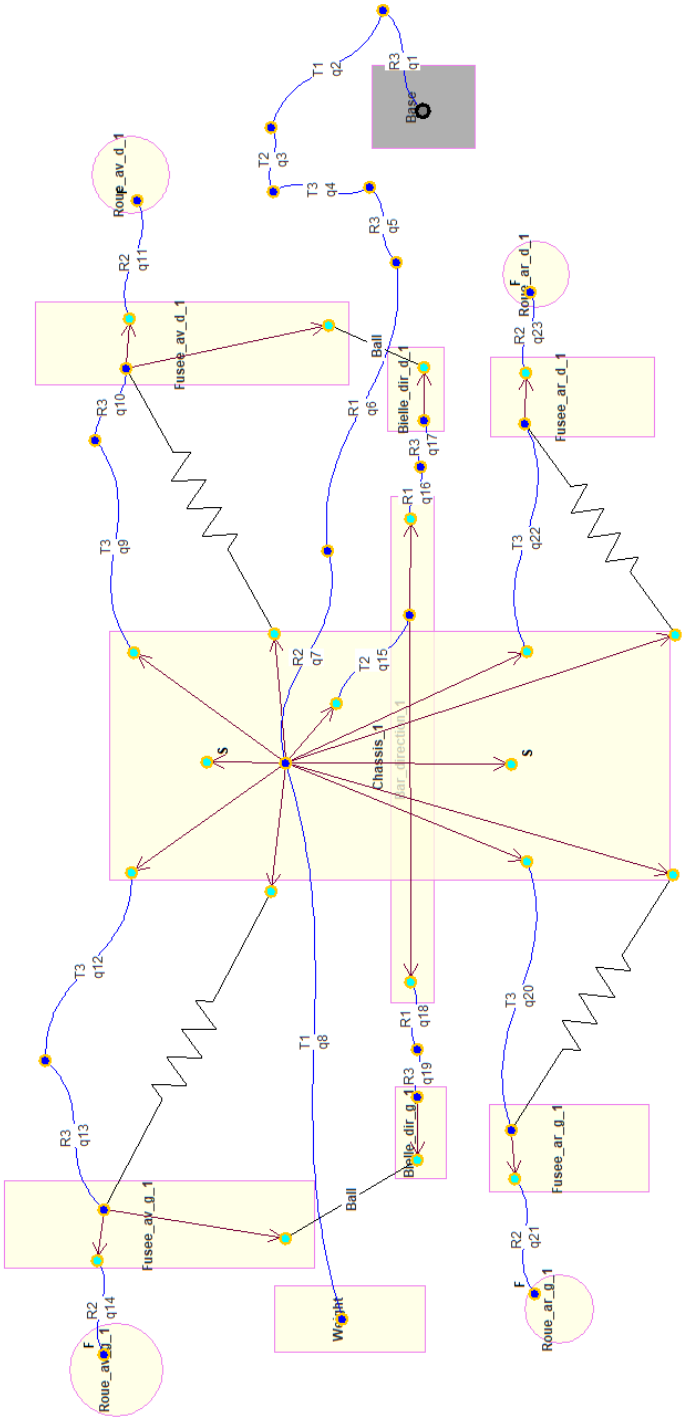
Annexe 1 - Fréquence des teintes de carrosserie

Cette annexe présente le fichier de configuration des fréquences de teintes de carrosserie. Celui-ci est utilisé dans la section dédiée au réalisme de la simulation en ce qui concerne les véhicules (cf. 3.2.3.2).

```
[/Game/OH_SmartCities/BP_CarsColorConfigObject.BP_CarsColorConfigObject_C]
Car_Color_Array=White
Car_Color_Array=Black
Car_Color_Array=Red
Car_Color_Array=Gray
Car_Color_Array=Blue
Car_Color_Array=Brown
Car_Color_Array=Green
Car_Color_Array=Yellow
Car_Color_Weight_Array=25
Car_Color_Weight_Array=21
Car_Color_Weight_Array=5
Car_Color_Weight_Array=25
Car_Color_Weight_Array=10
Car_Color_Weight_Array=2
Car_Color_Weight_Array=1
Car_Color_Weight_Array=1
Light_Workvan_Color_Array=White
Light_Workvan_Color_Array=Black
Light_Workvan_Color_Array=Gray
Light_Workvan_Color_Weight_Array=5
Light_Workvan_Color_Weight_Array=10
Light_Workvan_Color_Weight_Array=5
```

Annexe 2 - Modèle développé par Robotran

Cette annexe présente le modèle développé par Robotran d'un point de vue mécanique. C'est sur base de ce modèle qu'ont été menés les travaux présentés dans la seconde partie de notre thèse. Ce modèle est entre autres utilisé dans la partie concernant les interactions théoriques entre Digital Twin et Robotran (cf. 3.3.1).



Annexe 3 - Fiche technique du véhicule utilisé

Cette annexe présente la fiche technique du véhicule (Sedan) utilisée pour configurer le modèle Robotran. Cette fiche technique résulte d'une recherche concernant les données d'entrée et sortie présentées dans la section sur les interactions théoriques entre Digital Twin et Robotran (cf. 3.3.3.2).

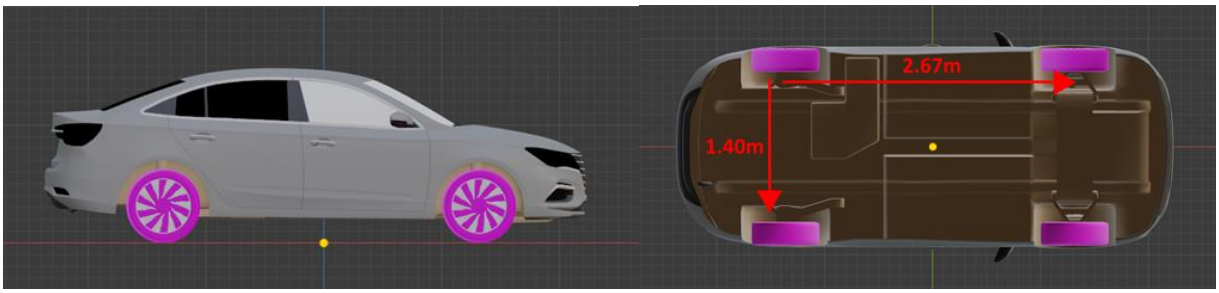
Input - Output du modèle Robotran

Input - Fiche technique Sedan :

1. Paramètres fixes:

Référentiel :

Le point référentiel du véhicule (notamment utilisé pour mesurer le centre de masse) est représenté par la marque jaune sur les images ci-dessous.
Il se situe au centre du rectangle formé par l'empattement et la voie.



Paramètres dimensionnels :

Empattement (m)	Longueur entre les deux essieux : 2.6776
Voie (m)	Largeur entre deux pneumatiques : 1.40

Paramètres dynamiques :

Masse du châssis (kg)	1 500.00
Masse des roues (kg)	Wheel_FR (avant droit) : 39.477188 Wheel_FL (avant gauche) : 39.464252 Wheel_RR (arrière droit) : 39.47794 Wheel_RL (arrière gauche) : 39.464237
Position du centre de masse du châssis (m)	X = 0.13486 Y = - 0.19997 Z = 0.62540 Ces coordonnées sont mesurées à partir du point référentiel spécifié plus haut.
Matrice d'inertie	Tenseur d'inertie (kg*cm²) : X=5 090 531.500 Y=21 595 106.000 Z=23 862 782.000 Matrice résultante (kg*m²) : $\begin{matrix} 509.053 & 0 & 0 \\ 0 & 2\,159.510 & 0 \\ 0 & 0 & 2\,386.278 \end{matrix}$
Raideur des suspensions (Nm^{-1})	$k = M\,(2\pi f)^2$ où M est la masse du véhicule (kg) et f est la fréquence d'oscillation des suspensions (Hz) • Suspensions arrière : k = $1500\,(2\pi * 15)^2$ = 13 323 966 • Suspensions avant : k = 1500 $(2\pi * 7)^2$ = 2 901 664
Roues motrices	Roues avant (traction).

Propriétés de l'environnement :

Sol glissant	Uniquement des zones spécifiques.
Localisation sol glissant	Position du point central (par rapport au repère fixe) + diamètre de la zone glissante.
Élévation	Pas d'élévation de la route.
Vent	Pas de vent.

2. Input du modèle :

Consignes de direction :

Angle volant	Angle de rotation du volant (de -270.0° à 270.0°)

Consignes d'accélération :

Pression de freinage	Couple de freinage max : 1500 Nm La valeur (brake input) récupérée est un pourcentage indiquant la pression sur le frein : • 0.0 = aucune pression exercée • 1.0 = pédale pressée au maximum
Accélération	La valeur (throttle input) récupérée est un pourcentage indiquant la pression sur l'accélérateur : • 0.0 = aucune pression exercée • 1.0 = pédale pressée au maximum

Output - Données nécessaires :

1. **Position du châssis :**
   Coordonnées par rapport au repère fixe (X=0 ; Y=0 ; Z=0).

2. **Rotation du châssis :**
   Format attendu : quaternion.

3. **Vitesse du châssis :**
   Vitesse linéaire en cm/s.

Rotation et vitesse de rotation des roues :
Position/rotation absolues par rapport au repère fixe pour ne pas avoir à prendre en compte le travail des suspensions.
Utile pour ajouter du réalisme à l'animation des roues.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**