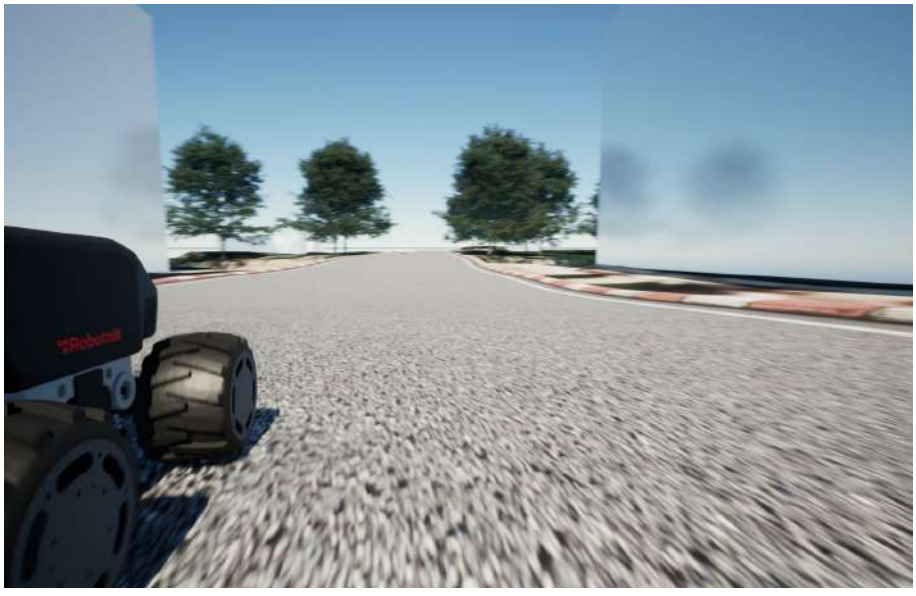
Simuler sur Unreal Engine 5 pour la Robotique



Stage assistant ingénieur Robotique

30 septembre 2023

ROUSSEL Etienne

Sommaire

Résumé

Ce rapport reprend les travaux sur l'implémentation d'un simulateur pour la robotique sur le logiciel Unreal Engine lors de mon stage à l'École Royale Militaire de Bruxelles. Il traitera des avancées et des difficultés rencontrées, mais il sera aussi axé sur la découverte du logiciel Unreal Engine 5. En effet, les objectifs du stage incitent à la recherche et à la découverte de différentes fonctionnalités du logiciel. Certaines parties décrivent donc principalement les méthodes utilisées, tandis que d'autres se concentrent logiquement sur les résultats.

Abstract

This report presents the work on implementing a simulator for robotics using Unreal Engine software. It will discuss the progress and challenges encountered, but it will also focus on exploring Unreal Engine 5. Indeed, the internship's objectives encourage research and exploration of various software functionalities. Therefore, some sections predominantly describe the methods used, while others logically focus on the results.

Remerciements

Je tiens d'abord à exprimer ma gratitude envers M. Émile Le Flecher, mon tuteur de stage, pour son accueil et son accompagnement attentif tout au long de cette expérience professionnelle. Je remercie également M. Emmanouil Maroulis, un chercheur avec qui j'ai travaillé, pour son assistance précieuse pendant la durée de mon stage. Enfin, je suis reconnaissant envers mon professeur Luc Jaulin à l'ENSTA Bretagne pour l'opportunité de stage qu'il m'a offerte.

Introduction

Quatres mois de stages destinés à découvrir le travail d'un ingénieur. Voilà comment a été abordé ce stage. Ce dernier s'est déroulé à l'étranger, l'occasion d'admirer la diversité culturelle belge. En effet, c'est à l'École Royale Militaire, à Bruxelles, que le laboratoire de Robotique et de Systèmes Autonomes (RAS Lab) m'a accueilli. La mission portait sur l'implémentation d'une interface de simulation avec le logiciel Unreal Engine 5. Nous avons donc participé au développement de cette interface en effectuant deux tâches principales : la construction automatique d'un robot sur Unreal Engine 5 par importation d'un fichier URDF ainsi que la conception d'un environnement réaliste de simulation. Le projet à plus long terme est de profiter du réalisme que peut offrir le logiciel Unreal Engine 5 afin d'entraîner efficacement des robots dans un environnement virtuel. Ce rapport détaillera les objectifs ainsi que l'environnement de stage dans la première partie puis se contrera sur les résultats des deux tâches principales dans les deux dernières parties.

1 Contexte et Organisation

1.1 École Royale Militaire et Laboratoire

Le stage s'est déroulé à l'École Royale Militaire de Belgique, à Bruxelles et en particulier dans le laboratoire de Robotique et de Systèmes Autonomes (RAS Lab) du département de mécanique. Les sujets d'études du département de mécanique sont variés. Ils ne sont pas tous liés à l'armement comme on pourrait s'y attendre. Les autres stagiaires du département avaient des sujets concernant des systèmes de déminage certes, mais aussi concernant la sécurité sur une grue ou encore sur la respiration assistée. Le RAS Lab, quant à lui, travaillait principalement sur un robot terrestre de déminage pendant la période du stage. Il y a tout de même quelques recherches annexes notamment sur les drones aériens. Une cage pour drones était même en construction dans le laboratoire. De plus, on y trouve des travaux utiles pour diverses applications comme c'est le cas de ce stage : offrir un bon environnement de simulation pour n'importe quel robot.

La majorité du travail du stage se faisait en autonomie, mais pour une partie, il a fallu se coordonner avec un chercheur du laboratoire. C'était l'occasion de parler anglais. En effet, les membres du RAS Lab étaient principalement étrangers et le chercheur en question était grec. Les interactions étaient donc légèrement perturbées par cette barrière de langage, mais il n'y a eu aucune incompréhension. Nous pouvons ajouter à cela des rapports oraux chaque semaine pour présenter le travail effectué. Finalement, malgré l'autonomie accordée, la notion de groupe était bien présente pour tenir à jour les autres membres. L'outil Jira a été utilisé pour organiser les tâches lorsque différentes personnes travaillent sur le même projet. Bien que le travail du stage fût majoritairement isolé du travail des laborantins, Jira offre tout de même de la visibilité sur l'avancement d'un projet. Il était donc intéressant de l'utiliser.

1.2 Objectifs

L'objectif général est d'offrir un environnement de simulation réaliste pour un robot donné. Le stage couvre deux parties dans cette réalisation : importer le robot dans le logiciel de simulation et créer une scène réaliste dans laquelle le robot évoluera. Le robot considéré lors du stage sera un robot terrestre de type rover, le Robotnik SUMMIT-XL visible sur la figure 1.

La recherche d'outils de simulation réaliste est intéressante en robotique, notamment pour entraîner les robots avec des algorithmes de machine learning. C'est pourquoi nous nous orientons vers le logiciel Unreal Engine 5 (UE5) pour simuler le robot dans un environnement virtuel. Aujourd'hui, le logiciel Gazebo est largement utilisé en robotique pour la simulation. Il est certes très pratique, surtout pour son intégration de ROS et son caractère open-source, mais malgré tout, d'autres logiciels offrent un rendu visuel plus réaliste, comme UE5. UE5 a été initialement développé pour créer des jeux vidéo en mettant l'accent sur la qualité visuelle. Cependant, il peut aussi être utilisé pour des simulations de robots, notamment avec l'aide de plugins développés par la communauté d'UE5. En effet, UE5 est un logiciel open-source, ce qui plaît aux chercheurs roboticiens.

FIGURE 1 – Photo du Robotnik

Comme énoncé précédemment, les missions du stage se découpent en deux parties. La première consiste à reconstruire le Robotnik sur UE5 de manière automatique à partir de sa description URDF [1]. La manipulation espérée est l'importation unique du fichier URDF. L'idéal serait d'établir la communication ROS au moment de l'importation. Nous verrons plus tard les difficultés liées à cela. La deuxième mission consiste à créer ou générer une scène (aussi appelée Map ou Level sur UE5) dans laquelle circulera le Robotnik. Pour rendre la scène proche de la réalité, nous devons intégrer à notre scène des objets courants comme des bâtiments, des arbres et des routes. Ils devront également servir d'obstacles. Le sol devra avoir un relief variable et praticable. De plus, il serait utile de créer des scènes de manière modulaire en utilisant des outils qui automatisent certaines parties de la conception. De cette manière, il sera plus facile de créer d'autres scènes. Une dernière mission était l'implémentation d'une interface reliant ROS2 et UE5 pour le contrôle du robot et la visualisation des données de capteurs. Cependant, cette dernière tâche n'a pas été réalisée, non seulement parce que la gestion de ROS2 sur UE5 était difficile, mais aussi parce que le temps a été occupé par la réalisation d'une scène.

C'est ainsi que s'est passé le stage : une autonomie pour découvrir et progresser vers les objectifs, sans être abandonné dans les tâches à accomplir.

2 Importer un robot

L'objectif principal consiste à simplifier au maximum l'importation d'un robot au sein d'une scène créée sur le logiciel Unreal Engine 5 (UE5). Bien que la reconstruction manuelle du robot sur UE5 soit envisageable, cette approche se révèle fastidieuse, propice aux erreurs et sujette à des imprécisions. Ainsi, opter pour une démarche automatisée s'avère des plus pertinents, particulièrement appréciée par les roboticiens. Pour appréhender la procédure d'importation d'un robot, quelle que soit sa structure, il est judicieux de se pencher sur l'approche de Gazebo. Ce dernier recourt à un fichier au format URDF pour décrire la configuration du robot, et par extension, pour intégrer la communication ROS au robot dans l'environnement Gazebo. Par conséquent, nous ambitionnons d'exploiter ce format de fichier afin de réaliser une reconstruction automatisée des robots au sein d'UE5, tout en permettant l'intégration de la communication via ROS.

2.1 Projets et Plugins sur UE5

Pour démarrer, prenons le temps de nous familiariser avec le logiciel. UE5 opère à travers des projets distincts, chacun avec ses propres réglages et contenu spécifique. Ainsi, pour accéder à un élément d'un autre projet, l'importer s'avère nécessaire. De plus, un projet peut englober plusieurs scènes, également connues sous le nom de « maps ». Voici comment la structure d'un projet se présente :

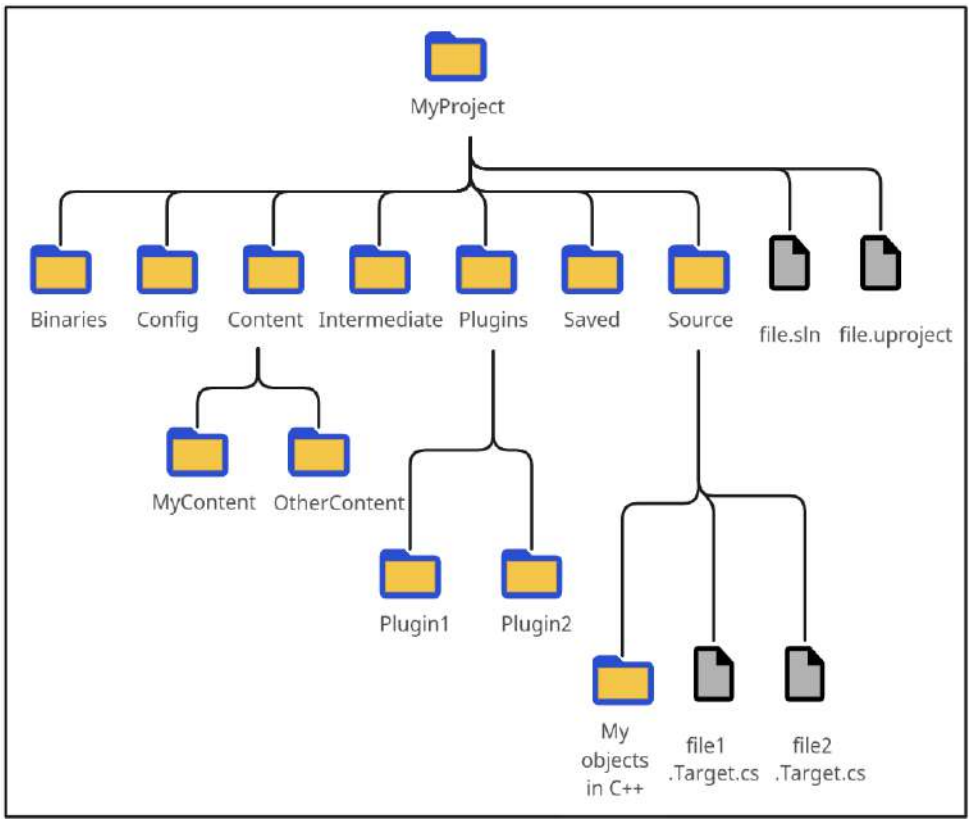


Figure 2 – Structure d'un projet sur Unreal Engine 5

Tous les objets créés par l'utilisateur se trouvent dans le dossier **Content**. Cependant, il peut aussi utiliser des éléments de base provenant directement des répertoires sources du logiciel (non situés dans le dossier du projet) ou issus du dossier **Plugins**. Le dossier **Binaries** résulte de la compilation du projet. Les dossiers **Intermediate** et **Saved** contiennent des fichiers temporaires nécessaires à l'exécution du projet, ainsi que des fichiers de sauvegardes. Ces fichiers deviennent pertinents lorsqu'il s'agit de diagnostiquer des pannes du logiciel. Une ressource pour le dépannage des plantages est disponible en annexe 3.2.3. Pendant la durée du stage, il a fallu apprendre à corriger les problèmes de nature « logicielle », d'autant plus que les projets peuvent intégrer des éléments en langage C++. En effet, une simple erreur de code peut entraîner un plantage d'UE5.

De la même manière, il a été essentiel d'apprendre à utiliser les plugins, qui permettent d'ajouter des fonctionnalités au logiciel. Cette connaissance sera d'une grande utilité. Par ailleurs, nous avons également dû apprendre à modifier les codes sources des plugins, ce qui peut également conduire à des plantages. Voici la configuration d'un plugin :

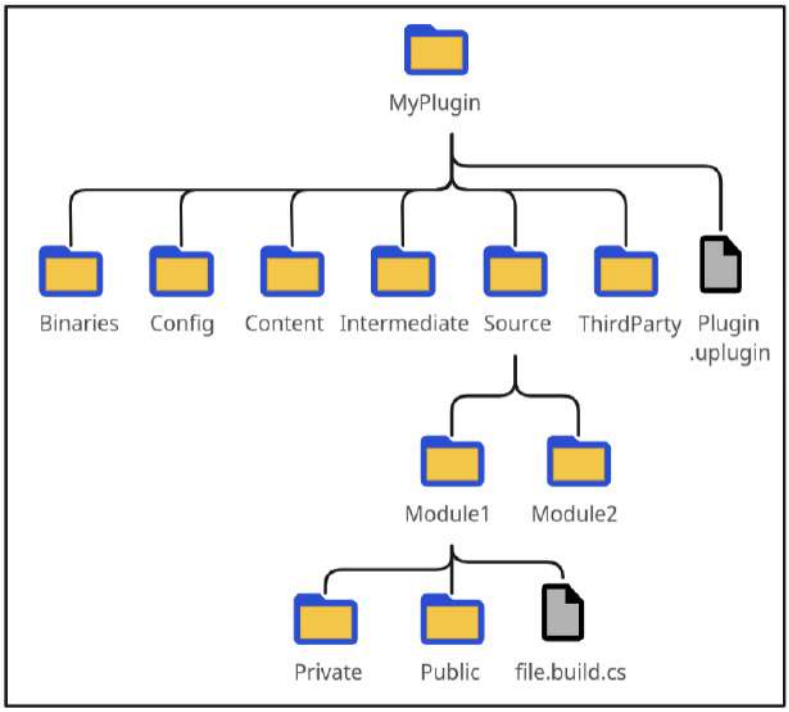


FIGURE 3 – Structure d'un plugin sur Unreal Engine 5

Le répertoire **Content** renferme les éléments fournis par le plugin. Le dossier **Source** peut englober plusieurs modules, et chaque module contient les codes sources spécifiques au plugin. Les plugins peuvent avoir des dépendances vis-à-vis de modules provenant d'autres plugins, mais non vis-à-vis d'un plugin entier. Cette subtilité peut être déroutante, notamment lorsque le nom d'un module correspond à celui du plugin.

2.2 Description Robotique

2.2.1 URDF

Dans le domaine de la robotique, les caractéristiques d'un robot sont précisément représentées au moyen d'un fichier URDF (URDF : Unified Robotic Description Format). Ce format permet une description complète du robot, incluant la disposition des composants, leur forme, la localisation des articulations, leur degré de mobilité, ainsi que d'autres propriétés géométriques et mécaniques telles que le couple maximal d'une articulation rotative, par exemple. Ces fichiers URDF reposent sur le format XML, ce qui signifie qu'ils peuvent être édités et lus sans nécessiter de logiciel spécifique. Voici un exemple de description de robot au format URDF :

```
<?xml version="1.0"?>
<robot name="summit_xl">
  <link name="summit_xl_base_link">
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://path/to/chassis.dae"/>
      </geometry>
    </visual>
    <collision>
      <origin rpy="-1.5707963267948966 0 0 " xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://path/to/chassis_collision.dae"/>
      </geometry>
    </collision>
  </link>

  <joint name="front_right_wheel_joint" type="continuous">
    <parent link="summit_xl_base_link"/>
    <child link="summit_xl_front_right_wheel"/>
    <origin rpy="0 0 0" xyz="0.229 -0.235 0.0"/>
    <axis rpy="0 0 0" xyz="0 1 0"/>
    <limit effort="100" velocity="100"/>
    <joint_properties damping="0.0" friction="0.0"/>
  </joint>

  <link name="summit_xl_front_right_wheel">
    <visual>...</visual>
    <collision>...</collision>
    <inertial>
      <mass value="6.5"/>
      <origin xyz="0 0 0"/>
      <inertia ixx="0.03185" ixy="0" ixz="0"
               iyy="0.039325" iyz="0" izz="0.03185"/>
    </inertial>
  </link>

</robot>
```

Dans cet exemple, on peut clairement constater que le format repose sur le langage XML. Un fichier URDF comporte une balise unique <robot>, à l'intérieur de laquelle se trouvent plusieurs balises <link> et <joint>. Les balises <link> représentent les composants du robot, tandis que les balises <joint> représentent les liaisons qui connectent ces composants. Les balises <parent> et <child> d'une balise <joint> sont ainsi indispensables. Chaque balise <link> englobe les informations géométriques du composant, à la fois la représentation visuelle et la géométrie de collision. Ces géométries peuvent être importées (balise <mesh>) ou définies par une forme simple, telle que <cylinder>. De plus, les balises <link> renferment également les propriétés dynamiques du composant, qui sont spécifiées dans la balise <inertial>. Il est évident que ce format permet une description exhaustive d'un robot.

Nous pouvons également examiner deux autres formats de fichiers similaires à l'URDF : Xacro et SDF. Le format Xacro (une fusion de XML et « macro ») est aussi bien adapté à la description des robots que les fichiers URDF. Il a été spécialement conçu pour simplifier la création de modèles de robots en autorisant l'utilisation de macros et de variables dans les fichiers XML. L'utilisation de macros permet de définir des modèles réutilisables qui peuvent être intégrés à d'autres fichiers, ce qui rend les descriptions robotiques plus modulaires et plus aisées à gérer. Par conséquent, il est logique que le Robotnik soit décrit en utilisant le format Xacro. D'autre part, le format SDF (Simulation Description Format) est un format de description de scène (et pas seulement de robot) largement employé avec Gazebo. C'est un format modulaire qui s'adapte parfaitement aux simulations multi-robots dans des environnements plutôt réalistes.

Il est important de noter une complexité imprévue qui est apparue au cours de ce stage. La manipulation de ces différents formats a soulevé des problèmes de conversion qui ne seront pas discutés en détail ici. Il a été nécessaire de convertir la description originale du Robotnik en un fichier .urdf, qui à son tour a été converti en .sdf. La justification de cette dernière conversion sera explorée dans la section 2.2.3. Une fois ces obstacles contournés, la conversion peut être réalisée simplement en utilisant ces lignes de commandes :

```
!/bin/bash
    xacro summit_xl_std.urdf.xacro -o summit_xl_std.urdf
    gz sdf -p summit_xl_std.urdf > summit_xl_std.sdf
```

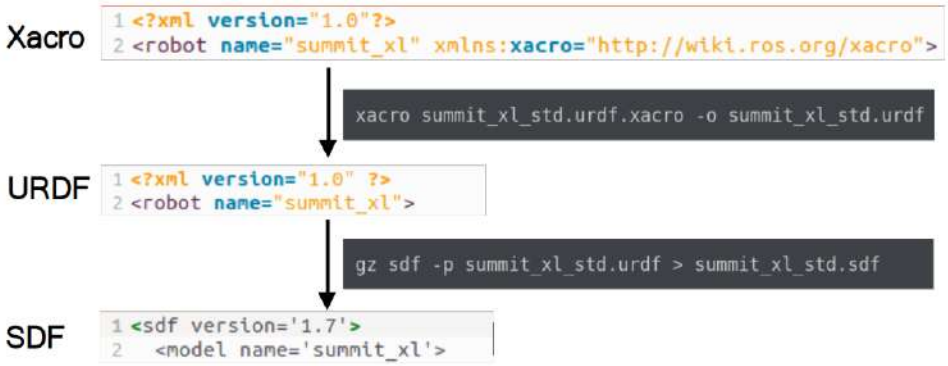


FIGURE 4 – Conversions de Xacro à SDF

Dans la figure 4, on peut observer que les formats Xacro et URDF reposent sur l'XML. Par ailleurs, on remarque qu'un robot dans le format SDF est dénommé <model> et la mise en retrait suggère qu'il peut en contenir plusieurs, ce qui diffère des formats URDF et Xacro.
Maintenant que la structure descriptive d'un robot est mieux saisie, il est primordial de mettre en place une méthode permettant à UE5 d'interpréter ce type de fichier pour le transformer en un robot fonctionnel.

2.2.2 Robot sur Unreal Engine

Pour créer un robot dans UE5, il est nécessaire de répliquer sa structure hiérarchique. Chaque partie du robot doit être associée à un repère propre, qui est positionné en relation avec le repère de son élément parent. Même si UE5 n'est pas exclusivement conçu pour la robotique, il s'avère tout à fait adapté à cette tâche. Que ce soit à travers le langage C++ d'UE5 ou manuellement dans le logiciel, établir la relation de parenté entre différents objets est un processus simple.

Mon expérience préalable en simulation s'est principalement déroulée au cours du quatrième semestre avec le logiciel V-Rep, qui se focalise spécifiquement sur la robotique. Par conséquent, travailler avec la même approche de pensée que dans V-Rep s'avère plutôt confortable. La seule distinction notable réside dans le fait que, dans UE5, les composants enfants sont liés directement au composant parent plutôt qu'à travers des liaisons, comme illustré dans la figure 5 ci-dessous.

FIGURE 5 – Hierarchie structurelle sur UE5 (à gauche) et sur V-rep (à droite)

Au cours de la phase initiale du stage, il a été impératif de se familiariser avec le langage de programmation employé par UE5. Bien que basé sur le C++, il nécessite l'utilisation des objets et fonctions spécifiques au logiciel. La multitude de classes d'objets variées ainsi que les conventions d'écriture propres à UE5l rendent ce processus d'adaptation relativement lent.
La figure 6 montre un exemple de code C++ qui illustre la manière de construire un robot dans UE5.

exemple.cpp

```cpp
// Components (=link) : Base, WheelLeft, WheelRight
// Constraints (=joint) : Base_WheelLeft, Base_WheelRight

// Attachment of components
WheelLeft->SetupAttachment(Base);
WheelRight->SetupAttachment(Base);

WheelLeft->SetRelativeLocation(FVector(3.2, -8, 2.3));
WheelRight->SetRelativeLocation(FVector(3.2, 8, 2.3));

// Attachment of constraints
Base_WheelLeft->SetupAttachment(WheelLeft);
Base_WheelRight->SetupAttachment(WheelRight);

// Physics
Base->SetSimulatePhysics(true);
WheelLeft->SetSimulatePhysics(true);
WheelRight->SetSimulatePhysics(true);

Base_WheelLeft->ComponentName2.ComponentName = TEXT("Base");
Base_WheelLeft->ComponentName1.ComponentName = TEXT("WheelLeft");
Base_WheelLeft->SetDisableCollision(true);
Base_WheelLeft->SetRelativeLocation(FVector(0, -8, 2.3));
Base_WheelLeft->SetRelativeRotation(FRotator(0, -90, 0));

// idem for Base_WheelRight
```

FIGURE 6 – Fonctions principales à la construction d'un robot en c++ d'Unreal Engine 5

Les noms des fonctions sont assez explicites. Par exemple, **SetupAttachment()** établit une hiérarchie entre deux éléments, tandis que **SetRelativeLocation()** positionne l'élément enfant dans le repère du parent. Les liaisons, également appelées « Constraints », permettent de lier un composant enfant à son parent, ce qui rend les liaisons indépendantes de la hiérarchie globale. Dans UE5, les liaisons n'ont donc pas nécessairement besoin d'un composant parent et d'un composant enfant. Néanmoins, il est judicieux de les intégrer dans la hiérarchie pour les positionner correctement. En effet, les fichiers URDF placent les liaisons par rapport à leur parent.

Finalement, en comprenant quelques caractéristiques des pièces et des liaisons, il devient possible de construire un modèle de robot cohérent. Plus on intègre de propriétés, plus le robot simulé aura un comportement réaliste. Grâce à une bonne maîtrise des objets et fonctions d'UE5, on commence à saisir comment bâtir un robot à partir d'un fichier URDF : en reproduisant la hiérarchie des éléments du robot tout en incorporant les caractéristiques physiques à l'aide des fonctions appropriées.

Ainsi, l'approche consiste à développer un interpréteur (plus précisément appelé parseur) qui examinera le fichier URDF, interprétera chaque ligne et la traduira en une fonction correspondante en langage C++ d'UE5.

2.2.3 plugin UroboSim

Avant de se lancer dans la création d'un parseur personnalisé qui traduira les informations d'un fichier URDF en code C++ utilisable dans UE5, il est pertinent d'examiner les solutions existantes pour l'intégration de robots dans UE5.

Le projet de grande envergure dans le domaine de la robotique pour UE5 est le plugin ROSIntegration. Cependant, ce plugin se concentre uniquement sur l'intégration de ROS dans le logiciel. Il est important de noter qu'il a été développé pour UE4 et n'est pas compatible avec UE5. Un autre plugin intéressant est **AGX Dynamics** [2].

Ce dernier plugin offre une multitude de fonctionnalités, parmi lesquelles figure l'importation de fichiers URDF. Après avoir installé le plugin, l'importation d'un fichier URDF n'a posé aucun problème. Cependant, un obstacle s'est présenté lors de la simulation en raison d'un manque de licence. L'utilisation de ce plugin impliquait des frais, ce qui a conduit à l'abandon de cette solution.

La solution retenue est l'utilisation du plugin **URoboSim** [3]. Ce plugin permet d'importer un fichier de description robotique, mais au format SDF uniquement. Heureusement, il est simple de convertir un URDF en SDF, ce qui pose peu de problèmes. Cependant, on notera que le plugin a été développé pour fonctionner avec UE4, ce qui a nécessité des modifications pour réduire au maximum les erreurs de compilations. La plupart des erreurs étaient correctibles en changeant manuellement le nom de fonctions et de chemin de fichiers car ils étaient obsolètes. Cependant, un erreur persistait. C'est en cherchant des informations sur le github d'UroboSim [3] que la solution a été trouvée : une des branches du projet github contenait une version du plugin fonctionnel avec UE5. Il a donc été possible d'importer un premier robot. Ce robot était un robot bipède provenant d'un fichier URDF en ligne [4].

Afin d'assurer le bon fonctionnement de la simulation, il a été nécessaire d'apporter certaines modifications aux propriétés du robot dans le logiciel. Par exemple, il a fallu activer la physique pour tous les composants. Étant donné que cette tâche pouvait être fastidieuse, il s'est avéré judicieux d'apporter des ajustements au code source du plugin. Cela a permis d'intégrer ces propriétés dès la construction du robot. Lors de la tentative d'importation d'un autre robot, en l'occurrence le célèbre Turtlebot, un problème majeur est survenu : les dimensions des composants ne concordaient pas. Concrètement, la balise <scale> des fichiers SDF n'était pas prise en compte. Une nouvelle fois, il a été nécessaire d'intervenir sur le code source du plugin URoboSim afin de mettre en œuvre cette fonctionnalité.
Voici Upkie et Turtlebot Burger une fois correctement importé :

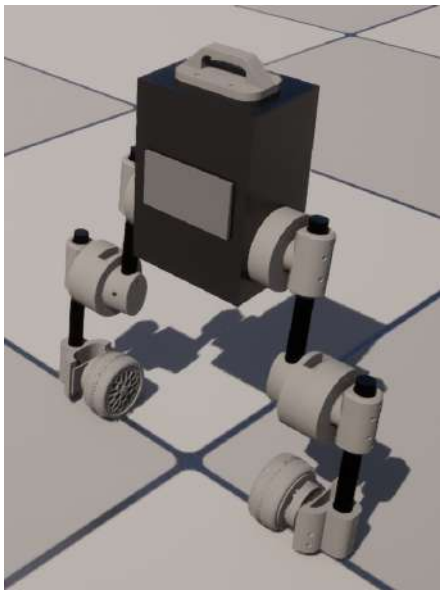


FIGURE 7 – Robot Upkie sur UE5

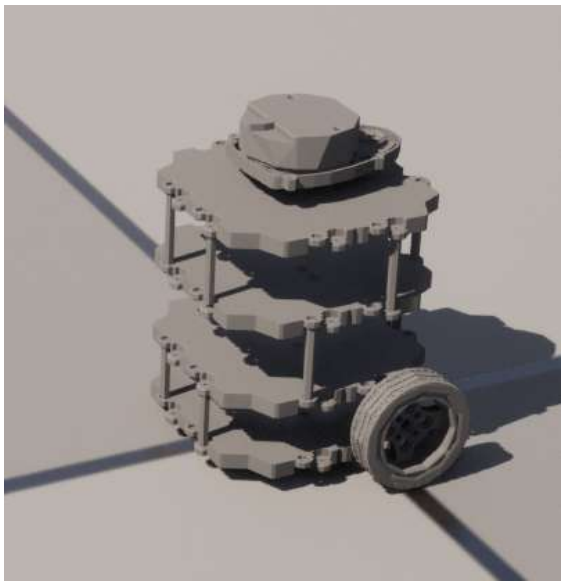


FIGURE 8 – Robot Turtlebot (modèle Burger) sur UE5

Un aspect distinctif d'UE5 est sa restriction à l'importation de fichiers 3D aux formats .obj et .fbx. Le plugin URoboSim, quant à lui, ne permettait que l'importation de fichiers .fbx. Ainsi, nous avons suivi la méthode exposée dans le fichier readme.md d'URoboSim pour convertir les fichiers 3D au format .dae ou .stl en fichiers .fbx. Cette approche repose sur l'utilisation d'un script Python à exécuter dans le logiciel Blender. Vous pourrez trouver ce script en annexe 3.2.3.

En fin de compte, après avoir réalisé la conversion des fichiers 3D du Robotnik en format .fbx, ainsi que sa description Xacro en URDF puis SDF, nous avons pu importer le Robotnik grâce à la version modifiée du plugin. Vous le trouverez ci-dessous.

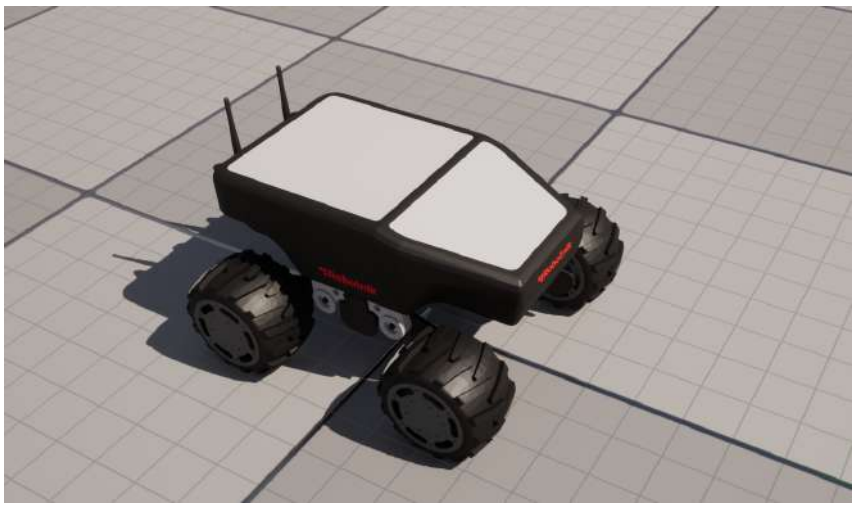


FIGURE 9 – Robot Robotnik sur UE5

2.3 ROS

L'étape d'importation d'un robot avec l'ensemble de ses composants et de ses liaisons a été accomplie avec succès. Il nous reste à présent à parvenir à le contrôler depuis ROS. Il est à noter que le plugin URoboSim contient du code permettant cette interaction, mais il est orienté vers ROS1. Toutefois, dans le contexte de notre projet, l'utilisation de ROS2 est l'objectif visé, et l'enseignement à l'ENSTA-Bretagne se concentre désormais sur ROS2 plutôt que sur ROS1. La première option qui s'offre à nous serait de modifier le plugin existant pour le rendre compatible avec ROS2. Cependant, cette approche impliquerait une refonte substantielle du plugin, étant donné que ROS2 diffère considérablement de ROS1. Cette tâche serait d'autant plus complexe du fait de notre niveau d'expérience relativement limité dans le domaine de ROS2, et encore moins dans celui d'UE5. C'est pourquoi nous explorons des solutions alternatives pour permettre la communication via ROS2 dans UE5. Celle que nous avons identifiée et retenue est l'utilisation du plugin **RapyutaSimulationPlugins** [5].

2.3.1 RapyutaSimulationPlugins

L'élément marquant de ce plugin, qui le distingue des autres plugins dédiés à l'intégration de ROS dans UE5, est qu'il repose sur ROS2 pour UE5 - précisément notre objectif. Ce plugin comprend les classes d'objets nécessaires à la mise en œuvre de robots capables de communiquer avec ROS2. Il intègre également un robot de démonstration, construit en utilisant les classes propres au plugin et le langage C++. Ce dernier point est particulièrement important, car cela correspond à la méthode que nous avons abordée précédemment. De plus, ce plugin facilite la gestion des namespaces. Chaque robot possède un namespace dédié, ce qui s'avère extrêmement utile pour la simulation multi-robots, une fonctionnalité qui peut être très pertinente dans certains scénarios.

À présent, la question qui se pose est celle de la compatibilité entre le plugin URoboSim et le plugin RapyutaSimulationPlugins en ce qui concerne l'importation automatique de robots. Bien qu'un rapprochement ait été tenté entre les deux, il est devenu évident que la communication via ROS2 avec un robot importé (c'est-à-dire non construit manuellement en C++) n'est pas une option viable. Tout d'abord, en apportant des ajustements mineurs au code source du plugin URoboSim, nous avons réussi à éviter certains conflits de dénomination de variables. Ensuite, nous avons dû faire en sorte que le robot construit utilise la classe de RapyutaSimulationPlugins. En effet, un robot construit automatiquement avec URoboSim serait enfant de la classe *AActor*, propre à UE5. Par conséquent, nous avons dû substituer la classe *AActor* par la classe *ARRBaseRobot*, qui correspond à la classe parente des robots dans RapyutaSimulationPlugins. Grâce à ces ajustements rapides, il n'y a pas eu de problèmes de compatibilité entre les deux plugins. De plus, cela a permis à notre robot construit à partir d'un fichier SDF de bénéficier notamment d'un namespace sur ROS2. Cependant, un obstacle demeure. Notre robot n'est pas en mesure de communiquer avec ROS2, car les éditeurs ou les abonnés aux topics ne sont pas définis. En effet, les messages publiés sur les topics sont configurés au moment de la construction du robot et dépendent des choix de l'utilisateur. Par conséquent, il n'est pas possible de configurer un éditeur ou un abonné après avoir importé un robot avec URoboSim. Cette limitation signifie que l'ajout de capteurs qui communiquent avec ROS n'est pas possible. Dans le domaine de la recherche en robotique, cette limitation est contraignante.

N'ayant pas trouvé d'autres solutions pour automatiser l'importation de robots à partir de

fichiers SDF ou URDF tout en permettant une communication via ROS2, la décision a été prise d'utiliser malgré tout le plugin RapyutaSimulationPlugins et de reconstruire notre robot manuellement. Grâce à cette approche, il devient possible d'ajouter des capteurs selon nos besoins. Il est important de noter que ce plugin présente une particularité : il est uniquement compatible avec le système d'exploitation Linux. Cependant, étant donné que l'utilisation d'UE5 sur Linux n'est pas des plus simples, cette situation a constitué un volet intéressant du stage que nous aborderons brièvement.

2.3.2 Unreal Engine sur Linux

Dans cette section, nous allons discuter des spécificités et des défis liés à l'utilisation d'UE5 sur Linux, en mettant particulièrement l'accent sur Ubuntu 22.04. Tout d'abord, l'installation du logiciel n'a pas été aussi simple que ce que le guide pour linux [6] laissait entendre. Les étapes d'installation ainsi que les problèmes rencontrés sont détaillés dans l'annexe 3.2.3 du rapport. Une fois que l'on a acquis une compréhension approfondie du fonctionnement des projets UE5, des plugins, ainsi que des commandes requises, et que l'on est au courant des éventuels problèmes, il est tout aussi aisé d'être efficace sur Linux que sur Windows. Cependant, il est important de noter que des limitations subsistent, étant donné que certains plugins ne sont tout simplement pas compatibles avec Linux. Cela ne s'applique pas aux plugins URoboSim et RapyutaSimulationPlugins, mais il en va autrement pour des plugins tels que Landscaping, qui seront abordés ultérieurement. De plus, une grande partie des objets normalement téléchargeables depuis la **Marketplace** [7] ne sont pas non plus compatibles avec Linux.

3 Créer une scène

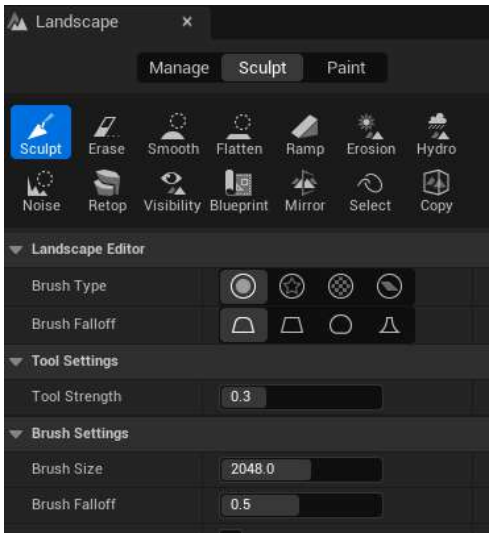
Nous considérons dans cette partie que nous avons un robot fonctionnel dans UE5. Pour entraîner le robot en simulation, il reste à concevoir un ou plusieurs environnements virtuels dans lesquels le robot se déplacera. L'objectif de cette partie est de construire une ou plusieurs scènes sur UE5 les plus réalistes possibles, incluant différentes sortes d'obstacles tels que des bâtiments, des arbres, etc. Ainsi, le robot devra se déplacer en évitant les obstacles grâce à ses capteurs (lidar ou caméra). C'est dans cette optique que nous concevrons les scènes. Il faut également que la conception des scènes soit relativement efficace, c'est-à-dire que nous favoriserons les solutions utilisant des plugins et les fonctions avancées d'UE5 pour rendre la conception plus automatique. L'objectif initial était de concevoir deux scènes : une avec un environnement rural/forestier et l'autre avec un environnement urbain.

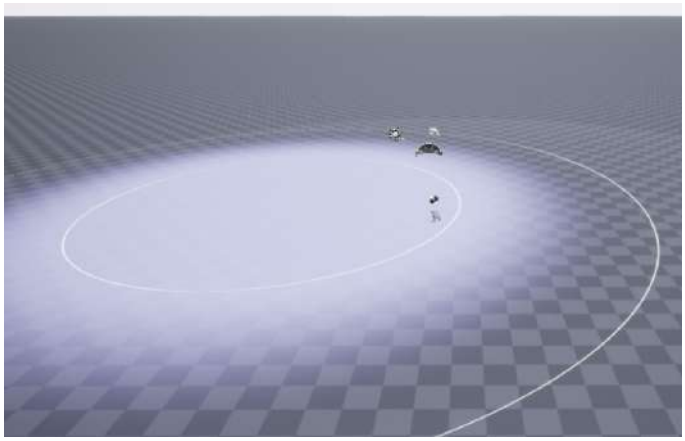
3.1 Environnement personnalisé

Tout d'abord, soyons créatifs. Nous pouvons utiliser les outils d'UE5 pour créer des reliefs et de la végétation selon notre bon vouloir.

3.1.1 Landscapes

Un sol auquel peut être affecté un relief est appelé *Landscape* sur UE5. C'est une grille plus ou moins grossière composée d'éléments carrés. Chaque angle de ces carrés a une hauteur, et la hauteur à l'intérieur des carrés est interpolée en fonction de celle de leurs angles. La précision du relief dépend donc de la taille des éléments carrés. Le relief peut être modifié manuellement avec un pinceau ou une brosse (*Brush* dans UE5). Elle permet d'ajouter de la hauteur, de l'érosion, d'aplatir, de lisser, et cela plus ou moins précisément grâce aux caractéristiques du pinceau (voir figure 10). Les actions de la brosse modifient la hauteur des angles de chaque carré. C'est une manière rapide de créer du relief, mais il restera peu réaliste.





Source : *https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/Landscape/QuickStart/*

FIGURE 10 – Caractéristiques et possibilités du pinceau à relief

Une autre manière de créer un relief est de l'importer. Un fichier de relief sur UE5 est une simple image en noir et blanc. L'image MNT (Modèle Numérique de Terrain) du lac de Guerledan générée lors du projet C++ a donc été utilisée comme fichier de relief. Après quelques modifications du programme du projet C++ et des paramétrages sur UE5, voici le relief du lac de Guerledan sur UE5 :

FIGURE 11 – Image topologique du lac de Guelédan

FIGURE 12 – Relief du lac de Guerlédan sur UE5

Plus tard, nous avons découvert le site *tangrams.github.io* [8] , qui offre la topologie du monde en noir et blanc. Cependant, il manque de précision si l'on veut de petites portions de terrain.

3.1.2 Végétation

Afin de concevoir une scène réaliste, nous nous intéressons ici à la végétation. Cela touche aux matériaux que l'on affecte au sol ainsi qu'aux éléments susceptibles de collision tels que des arbres ou des rochers. Avec la figure 13, nous voyons cette végétation sur un sol sans relief.

Les matériaux de surface ne sont pas en trois dimensions, même s'il semble le contraire. C'est-à-dire qu'un brin d'herbe ne dépassera pas du sol pour des matériaux de type "pelouse". Il existe des matériaux de base déjà présents dans un projet Unreal Engine lors de sa création. Cependant, si nous les utilisons sur une grande surface, une apparence élémentaire se répète dans les deux directions, ce qui n'est pas réaliste. Il est donc nécessaire, d'une part, de choisir un matériau de qualité, et d'autre part, de le modifier suivant son utilisation.
D'abord, pour choisir un matériau de qualité, il est conseillé d'utiliser la bibliothèque Quixel qui est un plugin sur UE5 offrant une multitude d'objets et de surfaces différentes issus de la réalité. En fait, ces objets ont été scannés puis convertis en éléments utilisables sur UE5. Une fois un matériau de cette bibliothèque choisi (par exemple de la pelouse), nous devons le modifier jusqu'à ce qu'il convienne. Parmi toutes les possibilités de transformation du matériau, nous pouvons mélanger plusieurs matériaux entre eux avec une prédominance pour l'un ajustable. Nous pouvons ajuster la taille apparente du matériau (un brin d'herbe apparaîtra plus grand). Cette taille est même ajustable en fonction de la distance à l'observateur pour accentuer les effets

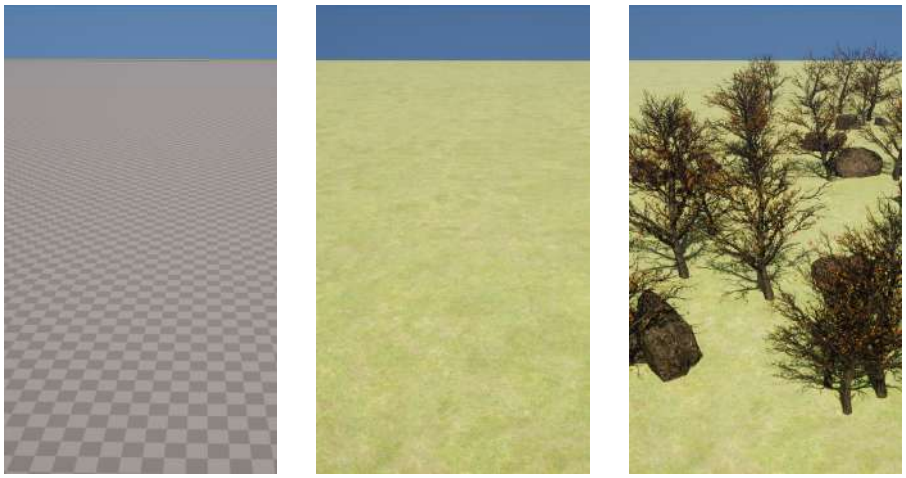


FIGURE 13 – Objets naturels sur le matériau « grass »

de proximité ou d'éloignement.
Finalement, après ces modifications, le matériau peut être assez réaliste. Celui utilisé sur la figure 13 possède les caractéristiques décrites juste avant. Il n'y a pas de motifs qui se répètent, il n'est pas monochrome car il y a un mélange de deux matériaux de gazon différents. L'observateur est trop loin pour voir les brins d'herbe, mais ils apparaîtraient s'il se rapprochait..

Comme dit précédemment, nous pouvons aussi piocher dans la bibliothèque **Quixel** des objets en 3 dimensions. Mais dans le cas des arbres, il n'y en a pas. Cependant, nous pouvons regarder dans la **Marketplace** d'UE5 pour trouver des objets en plus. De cette manière, nous avons accès à d'autres objets, en particuliers des arbres et des rochers. Généralement, ce sont des packs d'un certain type d'environnement qui sont mis à disposition. De cette manière nous pouvons ajouter à notre scène plusieurs espèces d'arbres, de plantes et plusieurs forme et taille de cailloux et de rochers.
Pour placer ces éléments sur un *Landscape*, nous utilisons l'outil *Foliage*. Ce dernier permet de répartir les objets sélectionnés selon une densité de probabilité. Cet aspect aléatoire affecte aussi la taille des éléments placés. Nous obtenons alors très rapidement une surface rempli de différentes variétés d'arbres et d'autres éléments voulu dans la scène.
Cet outil semble efficace pour construire une scène avec une forêt mais pas pour un environnement urbain. Si l'on veut ajouter des bâtiments ou des routes, nous devons les positionner un à un ce qui est laborieux. Or nous cherchons des méthodes les plus automatiques possibles.

Comme dit précédemment, nous pouvons aussi piocher dans la bibliothèque **Quixel** des objets en 3 dimensions. Mais dans le cas des arbres, il n'y en a pas. Cependant, nous pouvons regarder dans la **Marketplace** d'UE5 pour trouver des objets en plus. De cette manière, nous avons accès à d'autres objets, en particulier des arbres et des rochers. Généralement, ce sont des packs d'un certain type d'environnement qui sont mis à disposition. De cette manière, nous pouvons ajouter à notre scène plusieurs espèces d'arbres, de plantes et plusieurs formes et tailles de cailloux et de rochers.
Pour placer ces éléments sur un *Landscape*, nous utilisons l'outil *Foliage*. Ce dernier permet de répartir les objets sélectionnés selon une densité de probabilité. Cet aspect aléatoire affecte aussi la taille des éléments placés. Nous obtenons alors très rapidement une surface remplie de

différentes variétés d'arbres et d'autres éléments voulus dans la scène.

Cet outil semble efficace pour construire une scène avec une forêt, mais pas pour un environnement urbain. Si l'on veut ajouter des bâtiments ou des routes, nous devons les positionner un à un, ce qui est laborieux. Or, nous cherchons des méthodes les plus automatiques possibles.

3.2 Environnement géoréférencé

L'objectif reste de concevoir un environnement proche de la réalité donc nous nous intéressons à d'autres méthodes pour se rapprocher de cet objectif. En effet, il existe des plugins pour automatiser certaines parties de la construction d'une scène sur UE5 en utilisant de vraies données de terrains.

3.2.1 Plugin Landscaping

Le plugin **Landscaping** [9] permet d'importer des données de terrain et de les inclure sur UE5. Les données de terrain proviennent de zones géographiques sélectionnées. Détaillons le fonctionnement de ce plugin. Il faut deux types de fichiers de données : l'un est un DTM (Digital Terrain Models) qui nous donne le relief d'une zone géographique définie, et l'autre est un GIS (Geographic Information System) qui nous donne la position de tout ce qui constitue une zone géographique, par exemple la position d'un parc, de maisons, de routes ou encore d'une rivière.

Il est d'abord nécessaire d'importer des fichiers DTM afin d'obtenir un relief. Le relief est du type *Landscape*, donc il est modifiable. La documentation du plugin [10] offre quelques fichiers DTM que nous utiliserons pour construire une scène. Une fois le fichier DTM choisi, il est possible de préciser la zone géographique à générer via l'outil *Mapbox* du plugin (voir la figure 14). Cependant, pour toute partie de la zone sélectionnée par l'outil *Mapbox* qui est extérieure à la zone géographique du fichier DTM, aucune surface ne sera générée. En pratique, nous ne pouvons que rétrécir la zone par défaut.

FIGURE 14 – Outil Mapbox

Ensuite, il faut importer les données GIS. Ces données sont appelées *Vectordata* par le plugin.

Elles sont accessibles sur *geofabrik.de* [11]. Plusieurs fichiers en *.shp* (pour shapefiles) contiennent différentes informations de terrain sur un pays ou une région du monde. Par exemple, le tutoriel du plugin nous fournit des fichiers DTM correspondant à des zones rectangulaires en Autriche (pour le relief) et nous fournit aussi des fichiers de *Vectordata* en *.shp* contenant chacun des informations sur les routes d'Autriche, ses bâtiments, ses parcs, etc. Nous pouvons alors choisir les informations à importer dans notre scène. Lorsque nous importons les fichiers *.shp*, les éléments choisis sont représentés par des lignes ou *splines* sur UE5. Il reste donc à affecter un objet physique à ces lignes, comme par exemple une portion de mur pour créer des bâtiments. Dans ce cas, les bâtiments n'ont pas de toit, mais il est peut-être possible de créer des portions de mur plus complexes pour implémenter un toit aux bâtiments. La figure 15 montre la scène de test créée avec le plugin **Landscaping**.

FIGURE 15 – Des bâtiments dans une scène créée avec le plugin Landscaping

Le plugin offre une série de matériaux de sol différents que nous pouvons affecter aux éléments présents dans les fichiers de *Vectordata*. Par exemple, nous pouvons attribuer une apparence herbeuse pour les parcs et plutôt terreuse pour les champs. De plus, le matériau par défaut s'adapte au gradient du relief : nous remarquons la texture rocheuse sur les fortes pentes. Finalement, ce plugin offre une méthode de construction de scène assez automatique à partir de vraies données de terrain. Cependant, à l'exception du sol, il est difficile de construire les éléments de la scène. La documentation du plugin décrit d'autres possibilités, telles que l'importation d'imagerie satellite. Le plugin reste relativement complexe à utiliser, mais avec une maîtrise approfondie, construire un environnement complet et réaliste semble réalisable, comme le montre la vidéo du lien suivant : *https://www.youtube.com/watch?v=cb-rNlVVaPg* [12]

3.2.2 Plugin OpenStreetMap

Le plugin OpenStreetMap [13] permet d'importer une zone urbaine. Il utilise les données OSM (pour OpenStreetMap) en libre accès qui contiennent des informations sur les bâtiments. Elles renseignent sur la hauteur, la forme des bâtiments, les matériaux utilisés et même leurs heures d'ouverture. Le plugin ne se sert que de la hauteur, de l'adresse et de la forme des bâtiments d'une zone géographique pour les générer sur UE5. Les données OSM proviennent du site *openstreetmap.org* [14]. Nous observons sur la figure 16 que les données sur les routes

sont aussi importées. L'utilisation de ce plugin est très simple : sélectionner la zone à importer à partir du site *openstreetmap.org* [14], exporter puis importer le fichier (en *.osm*) dans le projet UE5 courant.

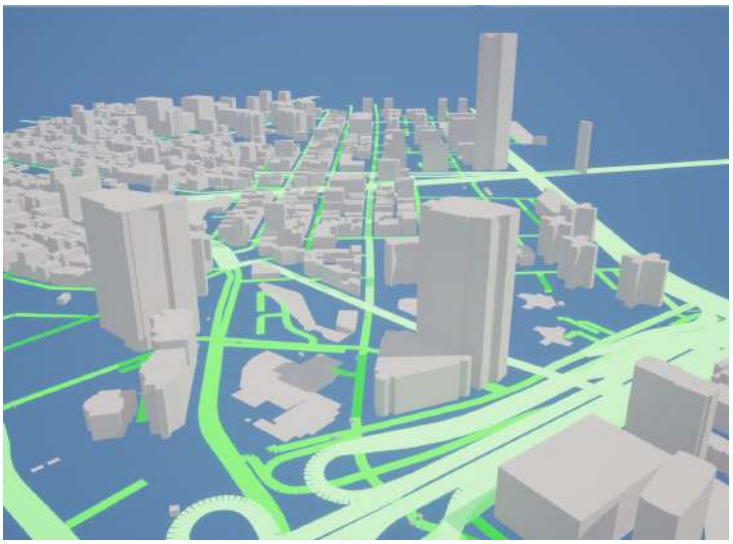


FIGURE 16 – New York sur UE5 avec OpenStreetMap

Bien que les bâtiments et le positionnement des routes soient automatiques, deux problèmes se posent. Le manque de données sur la hauteur des bâtiments est le premier. En effet, les grandes villes des États-Unis semblent bien fonctionner, le rendu sur UE5 est satisfaisant, mais les villes comme Bruxelles donnent très peu de bâtiments en 3 dimensions. Les milieux ruraux sont simplement plats. Le deuxième problème est la platitude du résultat. Il est plus intéressant pour le robot de circuler sur un sol variable, comme dans la réalité. Le résultat que donne ce plugin ne semble pas non plus modifiable.

Finalement, le plugin **OpenStreetMap** nous imposerait un environnement sans relief, le plugin **Landscaping** offre un relief intéressant mais il donne des bâtiments incomplets et identiques. Le plugin **Cesium** permet de palier à quelques problèmes.

3.2.3 Plugin Cesium

Le plugin **Cesium** [15] est rigoureusement appelé **CesiumForUnreal** car ce plugin a aussi été développé sur d'autres logiciel comme Unity. Il permet d'importer le monde entier dans le logiciel. Cela paraît assez exagéré mais c'est presque ce qu'il fait. En terminant le tutoriel de départ (cesium.com/learn/unreal/unreal-quickstart/), nous avons déjà un monde avec du relief et des bâtiments en 3D ainsi qu'un sol réaliste. Ce plugin est donc plus facile d'utilisation que le plugin *Landscaping*. Lorsque nous ajoutons le « monde » à la scène, les reliefs sont présents et le matériau du sol est basé sur des images satellite, d'où le réalisme. Ce « monde » s'étend et ne semble pas avoir de limites. Il n'y a pas de zone géographique sélectionnée qui délimite la partie du monde dans laquelle nous voulons simuler. Les bâtiments peuvent être ajouter seulement grâce à l'option OSM. Nous notons que le plugin Cesium utilise aussi les données OSM pour générer des bâtiments en 3D.

FIGURE 17 – Scène UE5 créée avec Cesium

Conclusion

Le stage a été très instructif, les résultats sont satisfaisants et pourtant, il reste un sentiment d'inaccomplissement. En effet, même si les objectifs principaux sont réalisés, des améliorations et des perfectionnements sont toujours possibles. Ce sentiment est d'autant plus présent que l'on s'attache au projet. En résumé, l'importation d'un robot sur Unreal Engine 5 a été entièrement réalisée pour la partie mécanique et physique du robot. En revanche, la communication ROS2 n'a pas pu être établie avec le robot importé malgré plusieurs tentatives. De plus, concevoir un environnement de simulation réaliste et automatique reste compliqué et chronophage malgré le travail que nous épargnent certains plugins. Pour estimer précisément la réussite du stage, il aurait fallu un cahier des charges techniques, mais nous aurions perdu le sens de ces quatre mois. L'objectif principal du stage était la découverte d'un nouveau logiciel, du travail de recherche en robotique et du travail en équipe. Le sujet complet n'était en fait que le résultat idéal à garder à l'esprit. La recherche en robotique semble consister en l'amélioration continue d'une solution jusqu'à l'épuisement des idées.

Références

[1] RobotnikAutomation. (s. d.-b). *Summit_Xl_common/summit_xl_description at Melodic-master · RobotnikAutomation/summit_xl_common*. GitHub. https://github.com/RobotnikAutomation/summit_xl_common/tree/melodic-master/summit_xl_description 5

[2] AGX Dynamics for Unreal in Code Plugins - UE Marketplace. (s. d.). Unreal Engine. https://www.unrealengine.com/marketplace/en-US/product/agx-dynamics-for-unreal 12

[3] Urobosim. (s. d.). *GitHub - Urobosim/URoboSim at Dev*. GitHub. https://github.com/urobosim/URoboSim/tree/dev 12

[4] Tasts-Robots. (s. d.). *Upkie_description/URDf/upkie.URDf at main · Tasts-robots/upkie_description*. GitHub. https://github.com/tasts-robots/upkie_description/blob/main/urdf/upkie.urdf 12

[5] Rapyuta-Robotics. (s. d.). *GitHub - rapyuta-robotics/RapyutaSimulationPlugins*. GitHub. https://github.com/rapyuta-robotics/RapyutaSimulationPlugins 14

[6] Installing Unreal Engine. (s. d.). *Unreal Engine 5.1 Documentation*. https://docs.unrealengine.com/5.1/en-US/installing-unreal-engine/ 15, 28

[7] Unreal Engine Marketplace | store of UE assets for games and 3D rendering. (s. d.). *Unreal Engine*. https://www.unrealengine.com/marketplace/en-US/store 15

[8] Tangram Heightmapper. (s. d.). https://tangrams.github.io/heightmapper/ 17

[9] Ludic Drive. (2023, 24 mai). *Landscaping - GIS data importer for Unreal engine. Landscaping - Unreal Engine Plugin*. https://landscaping.ludicdrive.com/ 19

[10] Landscaping for Unreal Engine. (s. d.-b). https://jorop.github.io/landscaping-docs/ 19

[11] Geofabrik download server. (s. d.). https://download.geofabrik.de/ 20

[12] Ludic Drive Art. (2022, 11 octobre). UE5 level design with Real World GIS Data | Free Asset Download [Vidéo]. YouTube. https://www.youtube.com/watch?v=cb-rNlVVaPg 20

[13] Ue4plugins. (s. d.). *GitHub - UE4Plugins/StreetMap : Import OpenStreetMap data into unreal engine*. GitHub. https://github.com/ue4plugins/StreetMap 20

[14] OpenStreetMap. (s. d.). *OpenStreetMap*. https://openstreetmap.org/ 20, 21

[15] Cesium for Unreal. (2015, 30 juin). *Cesium*. https://cesium.com/platform/cesium-for-unreal/ 21

[16] Code-Iai. (s. d.). *ConvertMeshToFBX/ConvertMeshObject.py at Master · Code-IAI/ConvertMeshToFBX*. GitHub. https://github.com/code-iai/ConvertMeshToFBX/blob/master/ConvertMeshObject.py 25

[17] Unreal Engine on GitHub. (s. d.). *Unreal Engine*. https://www.unrealengine.com/en-US/ue-on-github 28

[18] Download the latest official NVIDIA drivers. (s. d.). https://www.nvidia.com/download/index.aspx 28

Table des figures

Annexes

Annexe A

Script Blender de conversion STL ou DAE vers FBX

source : *https://github.com/code-iai/ConvertMeshToFBX/blob/master/ConvertMeshObject.py* [16]

```
#!/usr/bin/env python
import os
import bpy

# loops over all subfolder
CONVERT_DIR = "/path/to/meshes/"


def file_iter(path, ext):
    for dirpath, dirnames, filenames in os.walk(path):
        for filename in filenames:
            print(filename)
            extension = os.path.splitext(filename)[1]
            if extension.lower().endswith(ext):
                yield os.path.join(dirpath, filename)



def reset_blend():
    bpy.ops.wm.read_factory_settings(use_empty=True)

def convert_recursive(base_path):
    for filepath_src in file_iter(base_path, ".stl"):
        filepath_dst = os.path.splitext(filepath_src)[0] + ".fbx"
        print("Converting %r -> %r" % (filepath_src, filepath_dst))

        #reset_blend()
        #select all objects in scene
        for obj in bpy.context.scene.objects:
                obj.select_set(True)
        #delete all selected objects
        try:
            bpy.ops.object.delete()
        except Exception as e:
            print("Error during object deletion: ",e)

        try:
            print(obj.type)
            for obj in bpy.context.scene.objects:
                if obj.type != 'MESH':
                    obj.select_set(True)
                else:
                    obj.select_set(False)
```

```
            bpy.ops.object.delete()
            bpy.ops.import_mesh.stl(filepath=filepath_src)
            bpy.ops.export_scene.fbx(filepath=filepath_dst)
        except Exception as e:
            print("Error during stl conversion: ",e)


    for filepath_src in file_iter(base_path, ".dae"):
        filepath_dst = os.path.splitext(filepath_src)[0] + ".fbx"

        print("Converting %r -> %r" % (filepath_src, filepath_dst))

        for obj in bpy.context.scene.objects:
            try:
                print(obj)
                obj.select_set(True)
                #else:
               #     obj.select_set(False)
            except Exception as e:
                print("Error during deleting object ", e)
                break
        try:
            bpy.ops.object.delete()
            print("Import ", filepath_src)
            bpy.ops.wm.collada_import(filepath=filepath_src,
                                      import_units=True)
        except Exception as e:
            print("Could not import ", filepath_src, "Error: ", e)
            continue


        try:
            print("Export ", filepath_dst)
            bpy.ops.export_scene.fbx(filepath=filepath_dst)
        except Exception as e:
            print("Could not export ", filepath_dst, "Error: ", e)
            continue

if __name__ == "__main__":
    print("start")
    convert_recursive(CONVERT_DIR)
```

Annexe B

Aide à la gestion de crash d'UE5

Lorsque le logiciel crash, il faut supprimer le dossier **Binaries** puis recompiler le projet. La compilation peut se faire à partir du fichier en .sln sur Windows. Si le logiciel crash toujours, nous pouvons supprimer le dossier **Intermediate** puis « générer les fichiers de projet » avant de recompiler.

- Clic droit sur l'executable du projet -> « Generate Visual Studio project file »
- Ouvrir le fichier en .sln
- Clic droit sur le projet dans l'explorateur de fichier -> build
- Relancer le projet.

Une des erreurs classique est l'oubli de l'ajout d'un module nécessaire à notre code dans le fichier build.cs d'un projet ou d'un plugin.

```
using UnrealBuildTool;
using System.IO;

public class URoboSim : ModuleRules
{
  public URoboSim(ReadOnlyTargetRules Target) : base(Target)
  {
    PublicIncludePaths.Add(Path.Combine(ModuleDirectory, "Public"));
    PrivateIncludePaths.Add(Path.Combine(ModuleDirectory, "Private"));

    PublicDependencyModuleNames.AddRange(
      new string[]
      {
        "Module1",
        "Module2",
      }
    );

    PrivateDependencyModuleNames.AddRange(
      new string[]
      {
        "Module3",
      }
    );
  }
}
```

Annexe C

Installer Unreal Engine 5 sur Ubuntu

La méthode qui suit à été faite pendant le stage mais n'est pas la seule possible.

1. aller sur cette page : *https://www.unrealengine.com/en-US/linux* [6] et se connecter si besion.
2. Télécharger la version voulue du logiciel et le plugin « Bridge » (Optionnel sur Ubuntu. C'est un plugin présent par défaut sur windows. Il est utile pour créer des environnements réalistes.)

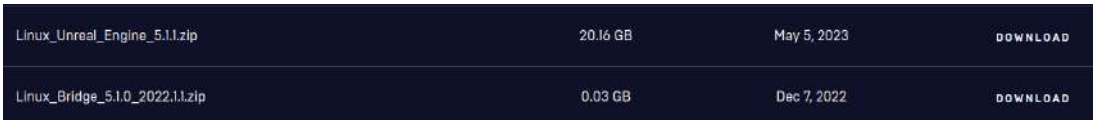
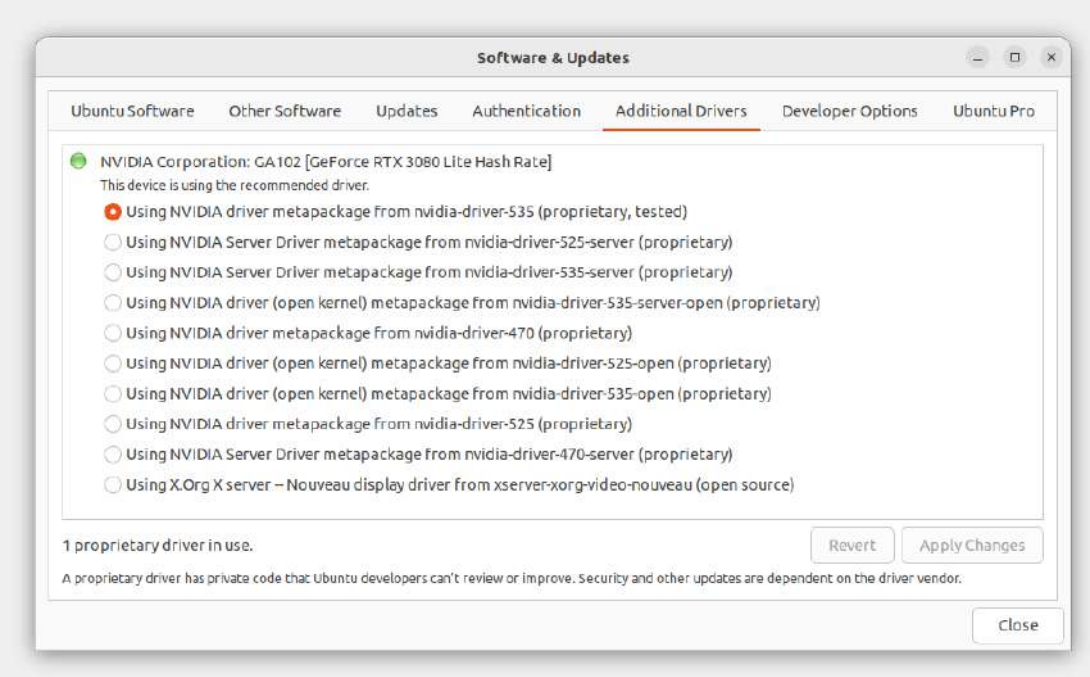


FIGURE 18 – UE5 à télécharger

3. Dézipper les 2 fichiers
4. Lors du stage, il manquait 2 fichiers dans */path/to/UE5/Engine/Build/BatchFiles/Linux/* : **BuildThirdParty.sh** et **SetupToochain.sh**. Récupérer-les sur Github : *https://github.com/EpicGames/*
   (a) Connecter son compte Github avec son Compte EpicGames en suivant la démarche ici : *https://www.unrealengine.com/en-US/ue-on-github* [17]
   (b) Choisir la branche Github correspondant à la version de Unreal Engine précédemment téléchargée.
   (c) Récuppérer les 2 fichiers depuis *https://github.com/EpicGames/UnrealEngine/tree/release/Engine/Bu* et les mettre dans */path/to/UE5/Engine/ Build/BatchFiles/Linux/*.
5. Changer les permissions pour les fichiers dans les dossiers */path/to/UE5/Engine/Binaries*, */path/to/UE5/Engine/Build* et */path/to/UE5/Engine/Extras* avec `chmod -R +x`
6. Essayer de lancer UE5 avec : `cd ./path/to/UE5/Engine/Binaries/Linux/UnrealEditor`
7. Si UE5 se lance, ignorer la suite. Sinon, un message d'erreur du type « No Vulkan device found » devrait apparaître. Dans ce cas il faut changer le pilote graphique. Le changement de pilote graphic à été fait sur une machine équipé d'un GPU Nvidia. Donc la démarche peut être différente pour d'autres GPUs.
8. Aller dans **Additional Drivers** depuis le menu Ubuntu.
9. Sélectioner un nouveau pilote. Pour savoir quel pilote choisir, aller sur *https://www.nvidia.com/download/i* [18] et entrer les caractéristiques GPU. Il faut choisir la version la plus proche de celle conseillée par ce site. Sur l'image ci-dessous, le pilote sélectionné fonctionne.



10. Appliquer les changements et redémarer.
11. Lancer UE5 avec `cd ./path/to/UE5/Engine/Binaries/Linux/UnrealEditor`

Annexe D : Rapport d'évaluation du stage

ENSTA Bretagne

RAPPORT D'EVALUATION
ASSESSMENT REPORT

Merci de retourner ce rapport par courrier ou par voie électronique en fin du stage à :
At the end of the internship, please return this report via mail or email to:

ENSTA Bretagne – Bureau des stages - 2 rue François Verny - 29806 BREST cedex 9 – FRANCE
☎ 00.33 (0) 2.98.34.87.70 / stages@ensta-bretagne.fr

I - ORGANISME / *HOST ORGANISATION*

NOM / *Name* Royal Military Academy of Belgium

Adresse / *Address* Avenue de la Renaissance 30, 1000 Brussels, Belgium

Tél / *Phone (including country and area code)* +32 470 35 80 47

Nom du superviseur / *Name of internship supervisor*
LE FLECHER Emile
Fonction / *Function* Researcher

Adresse e-mail / *E-mail address* emile.lefleche@mil.be

Nom du stagiaire accueilli / *Name of intern* Etienne Roussel

II - EVALUATION / *ASSESSMENT*

Veuillez attribuer une note, en encerclant la lettre appropriée, pour chacune des caractéristiques suivantes. Cette note devra se situer entre **A (très bien)** et **F (très faible)**
*Please attribute a mark from **A (excellent)** to **F (very weak)**.*

MISSION / *TASK*

- La mission de départ a-t-elle été remplie ? (A) B C D E F
  Was the initial contract carried out to your satisfaction?

- Manquait-il au stagiaire des connaissances ? ☐ oui/*yes* ☒ non/*no*
  Was the intern lacking skills?

  Si oui, lesquelles ? / *If so, which skills?* _____

ESPRIT D'EQUIPE / *TEAM SPIRIT*

- Le stagiaire s'est-il bien intégré dans l'organisme d'accueil (disponible, sérieux, s'est adapté au travail en groupe) / *Did the intern easily integrate the host organisation? (flexible, conscientious, adapted to team work)*
  (A) B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* Etienne worked in close collaboration with another researcher. The collaboration between them worked very well as seen in the simulation delivered.

COMPORTEMENT AU TRAVAIL / *BEHAVIOUR TOWARDS WORK*

Le comportement du stagiaire était-il conforme à vos attentes (Ponctuel, ordonné, respectueux, soucieux de participer et d'acquérir de nouvelles connaissances) ?

Did the intern live up to expectations? (Punctual, methodical, responsive to management instructions, attentive to quality, concerned with acquiring new skills)?

(A) B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* Etienne has shown a good balance between Autonomy and team work, asking questions when necessary and proposing its own solutions

INITIATIVE – AUTONOMIE / *INITIATIVE – AUTONOMY*

Le stagiaire s'est –il rapidement adapté à de nouvelles situations ? (A) B C D E F
(Proposition de solutions aux problèmes rencontrés, autonomie dans le travail, etc.)

Did the intern adapt well to new situations? A B C D E F
(eg. suggested solutions to problems encountered, demonstrated autonomy in his/her job, etc.)

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* _____

_____

CULTUREL – COMMUNICATION / *CULTURAL – COMMUNICATION*

Le stagiaire était-il ouvert, d'une manière générale, à la communication ? A (B) C D E F
Was the intern open to listening and expressing himself /herself?

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* _____

_____

OPINION GLOBALE / *OVERALL ASSESSMENT*

❖ La valeur technique du stagiaire était : Overall, it was a pleasure to (A) B C D E F
Please evaluate the technical skills of the intern:
work with Etienne for all the skills mentionned above. He would be very welcomed in the Lab future.

III - PARTENARIAT FUTUR / *FUTURE PARTNERSHIP*

❖ Etes-vous prêt à accueillir un autre stagiaire l'an prochain ?

Would you be willing to host another intern next year? ☒ oui/*yes* ☐ non/*no*

Fait à Brussels, le 21/08/2023
In _____, on _____

Signature Entreprise _____ Signature stagiaire
Company stamp _____ *Intern's signature*

ECOLE ROYALE MILITAIRE
Département de Mécanique
Secrétariat
Avenue de la renaissance, 30
B-1000 BRUXELLES
Tél. : 32 (0)2/441.40.97
Fax : 32 (0)2/441.41.00

Merci pour votre coopération
We thank you very much for your cooperation