



## PROJET LONG N7 SESSION 2020

RAPPORT DE STAGE

---

# TER atelier flexible

---



Team SALLAG : Promo 2020

Steve CROCE

Anthony FAVIER

Lucas VEIT

Lucie RICART

Antonin MESSIOUX

Guillaume AUFRAY-AMEN

*Tuteur : MME NGUEVEU SANDRA U.*

ET M. BRIAND CYRIL

27 Janvier 2020 — 6 Mars 2020

## Remerciement

Le projet long est un exercice nous rapprochant de plus en plus de nos conditions de travail en sortie d'école. Il nous est d'un grand secours pour se tenir prêt à notre insertion dans le monde de l'entreprise.

Mais réaliser un projet qui nous plaît est également une motivation supplémentaire de travail.

Ainsi, nous souhaitons remercier Mme Ngueveu Sandra U. ainsi que M. Briand Cyril pour être restés à l'écoute et disponibles pendant ces six semaines. Grâce à leur aide, nous avons pu orienter nos travaux vers ce qui nous tenait à coeur sans perdre leur soutien.

Nous tenons également à remercier les équipes de l'AIP-PRIMECA Toulouse pour leur présence et leur amabilité tout au long de notre séjour dans les locaux.

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Présentation du sujet</b>	<b>4</b>
2.1	Environnement . . . . .	4
2.1.1	Maquette de l'AIP-PRIMECA . . . . .	4
2.1.2	Simulation . . . . .	5
2.2	Prise en main des outils . . . . .	6
2.2.1	ROS : Robot Operating System . . . . .	6
2.2.2	Simulation 3D : CoppeliaSim . . . . .	7
2.3	Objectifs et organisation du travail . . . . .	7
<b>3</b>	<b>Développement et optimisation de la simulation</b>	<b>8</b>
3.1	Simplification des modèles dynamiques . . . . .	8
3.2	Modélisation des produits sur V-rep . . . . .	9
3.2.1	Produits physiques . . . . .	9
3.2.2	Produits non-physiques . . . . .	11
<b>4</b>	<b>Développement et optimisation du code</b>	<b>13</b>
4.1	Nettoyage du travail effectué précédemment . . . . .	13
4.2	Suppression des intelligences . . . . .	13
4.3	Ajout de "manager" . . . . .	14
<b>5</b>	<b>Migration de ROS Indigo vers ROS Kinetic</b>	<b>15</b>
5.1	Imitation des services avec des topics . . . . .	15
5.2	Changement de l'interface graphique . . . . .	15
<b>6</b>	<b>Mise en place du checker</b>	<b>17</b>
6.1	Sujet . . . . .	17
6.2	Le fichier Config . . . . .	17
6.3	Le fichier log . . . . .	18
6.4	Le Checker . . . . .	19
<b>7</b>	<b>Conclusion</b>	<b>21</b>

# 1 Introduction

Notre 3ème année du cycle ingénieur à l'ENSEEIH en option CDISC (Commande, Diagnostique et Informatique des Systèmes Critiques), nous donne l'opportunité de réaliser un stage de 6 semaines pour se consacrer à un "Projet Long". Nous avons choisi de le réaliser à l'AIP-PRIMECA, répondant à une demande de préparation d'un nouveau sujet du TER Atelier Flexible pour les élèves des prochaines années.

Nous avons donc travaillé sur la simulation de la maquette présente à l'AIP, constituée d'un circuit monorail sur lequel circule des navettes pouvant transporter des produits. Ce circuit est entouré de 4 robots pouvant effectuer des tâches sur chaque produit acheminé par les navettes. La simulation que nous proposons devra donc permettre à l'étudiant d'implémenter un réseau de Petri afin de d'assurer la gestion de cet environnement (qui peut s'apparenter à un environnement de production). Pour cela, nous avons repris le travail laissé par les anciens stagiaires et groupes de projet long.

## 2 Présentation du sujet

Dans un premier temps, nous allons introduire le sujet et l'environnement de travail afin de comprendre au mieux nos démarches.

### 2.1 Environnement

#### 2.1.1 Maquette de l'AIP-PRIMECA

Au sein des locaux de l'AIP se trouve la maquette décrite précédemment. On y trouve donc un circuit monorail, des navettes, des robots, des capteurs de position et des aiguillages.

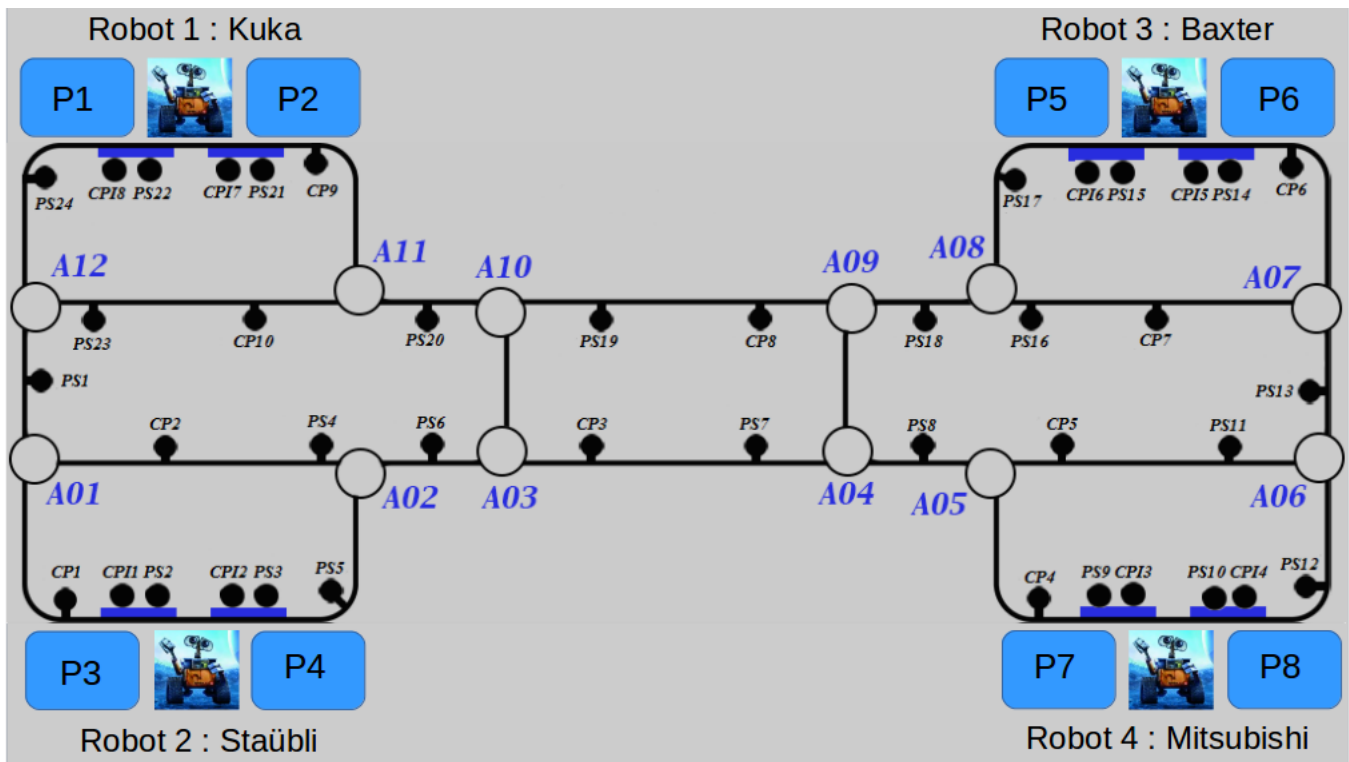


FIGURE 1 – Schéma de la Maquette

Les navettes sont alimentées par le monorail, elles avancent donc automatiquement et possèdent un capteur de proximité frontal qui permet d'éviter les collisions.

Les capteurs (PS et CP sur le schéma) permettent de connaître partiellement les positions des navettes et ainsi aident à l'orientation des aiguillages. En plus de pouvoir détecter les navettes, les Points de Stop (PS) sont les seuls endroits où l'on peut programmer leur arrêt.

On retrouve également les aiguillages noté AXX, permettant d'orienter les navettes, et les postes de travail notés PX, où certaines tâches pourront être effectuées sur les produit.

Au cours de notre stage, nous avons passé un certain temps auprès d'autres équipes manipulant la maquette. Ainsi, nous avons pu en extraire toutes les informations dont nous avons besoin pour nous assurer que la simulation était proche de la réalité. Néanmoins, nous n'avons jamais directement programmé les véritables automates, puisque nous étions concentrés sur l'amélioration de la simulation en vue de la proposer en tant que sujet de TER à l'ENSEEIH.

### 2.1.2 Simulation

Après avoir observé la maquette, nous découvrons la simulation. Les élèves des années précédentes ont obtenu une simulation très proche en apparence à la maquette. Voici la fenêtre de l'interface TER précédente :

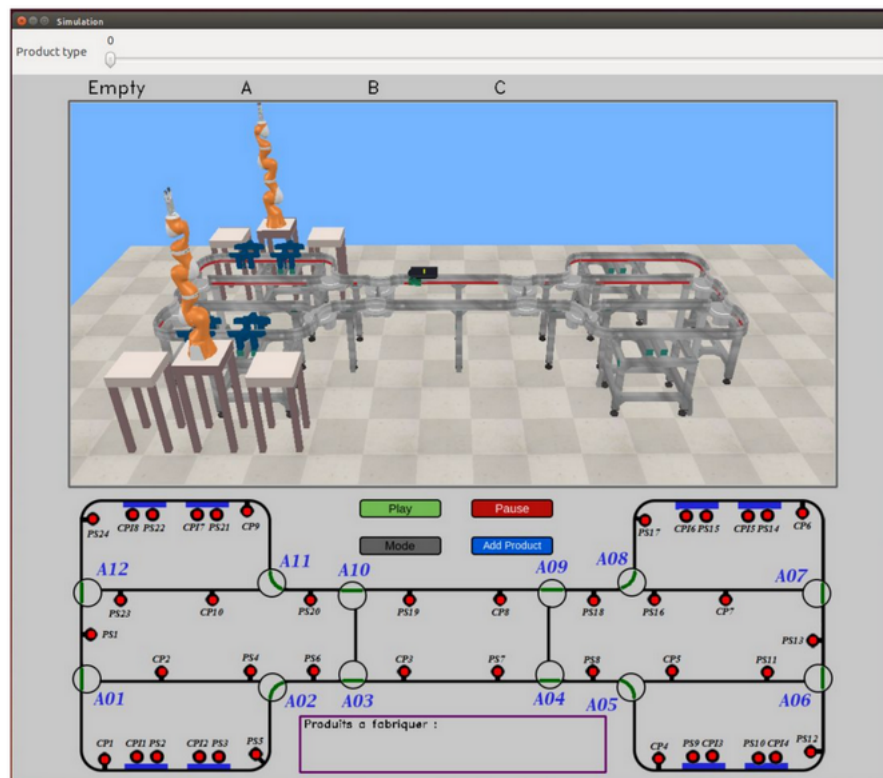


FIGURE 2 – Interface 2019

On y observe donc bien une modélisation 3D de la maquette, accompagnée d'un schéma du circuit monorail et des capteurs en dessous.

Lors du lancement de la simulation un certain nombre de navettes apparaissent. Grâce au bouton "Add product", nous pouvons colorier un des 4 postes pour simuler l'apparition d'un produit.

Les aiguillages, les robots et les "stop" des capteurs sont à gérer via le réseau de Petri, grâce à des fonctions haut niveau dont la documentation nous a été fournie.

Jusque là, la simulation ne permet pas aux robots de manipuler/déplacer des objets entre les navettes et les postes de travail. Un système de coloration de navette et de poste est utilisé pour modéliser ces

déplacements d'objet.

## 2.2 Prise en main des outils

Afin de réaliser/continuer ce projet, nous devons maîtriser des outils spécifiques : le middleware ROS(Robot Operating System) et le logiciel de modélisation 3D V-Rep (puis CoppeliaSim).

### 2.2.1 ROS : Robot Operating System

Le middleware ROS est un ensemble de bibliothèques et d'outils permettant de développer des logiciels pour la robotique. C'est un méta-système d'exploitation qui peut fonctionner sur un ou plusieurs ordinateurs et qui fournit plusieurs fonctionnalités telles que l'abstraction du matériel, le contrôle des périphériques de bas niveau, la mise en œuvre de fonctionnalités couramment utilisées, la transmission de messages entre les processus et la gestion des packages installés.

ROS offre une architecture souple de communication inter-processus et inter-machine. Les processus ROS sont appelés des nodes et chaque node peut communiquer avec d'autres via des topics. La connexion entre les nodes est gérée par un master et suit le processus suivant :

- Une première node avertit le master qu'elle a une donnée à partager
- Une deuxième node avertit le master qu'elle souhaite avoir accès à une donnée
- Une connexion entre les deux nodes est créée
- La première node peut envoyer des données à la seconde

Une node qui publie des données est appelée un publisher et une node qui souscrit à des données est appelée un subscriber. Une node peut être à la fois publisher et subscriber. Les messages envoyés sur les topics sont pour la plupart standardisés, ce qui rend le système extrêmement flexible.

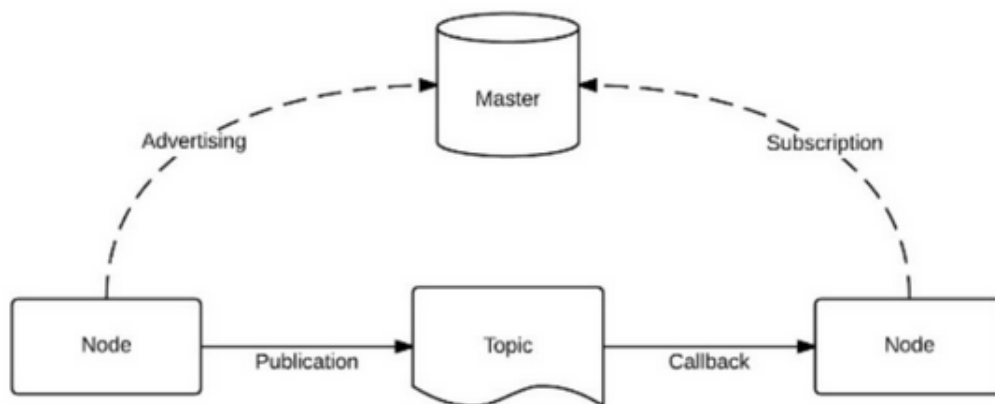


FIGURE 3 – schemaROS

ROS permet une communication inter-machine, des nodes s'exécutant sur des machines distinctes, mais ayant connaissance du même master, peuvent communiquer de manière transparente pour l'utilisateur.

Il existe également des messages asynchrones dans ROS : les services. Les services sont accessibles directement entre les nodes sans passer par les topics ; la node ayant besoin d'un service particulier va faire la demande au master qui va la mettre en relation directement avec la node correspondante.

On remarque donc que plusieurs nodes distinctes peuvent être gérées par un même master, et chaque node peut être associée à un robot. Dans un système on aura donc une ou plusieurs nodes (représentant les différents robots/éléments) gérés par un seul master et pouvant communiquer simplement entre eux. D'où l'intérêt d'utiliser un tel système dans notre chaîne de production, de plus que ROS est open source.

### 2.2.2 Simulation 3D : Coppeliasim

En ce qui concerne le logiciel de simulation 3D, nos prédécesseurs ont utilisé V-rep, mais la nouvelle version s'appelle Coppeliasim. Ce logiciel est dédié à la simulation intégrant des robots développée par la compagnie Coppelia Robotics. Il est intéressant d'utiliser ce logiciel puisque les bras robots que nous utilisons sur la maquette sont déjà modélisés au sein de la simulation. De plus, il existe un plugin interfaçant Coppeliasim et ROS ce qui permet de contrôler les objets simulés en déclarant simplement des Subscribers écoutant les topics de commande.

## 2.3 Objectifs et organisation du travail

Pour avoir une communication optimale entre notre équipe et nos encadrants (nos clients), nous avons mis en place une méthode de fonctionnement agile. Pour cela nous avons utilisé le site Trello, permettant de définir l'ensemble des objectifs à court et long terme (semaine par semaine) et de se tenir informés tous ensemble de l'avancée du projet en y postant les grandes lignes ou la documentation de notre travail. De plus, nous avons rendez-vous tous les lundis matins afin de discuter de notre projet.

En ce qui concerne le code ROS, nous avons utilisé la plateforme Github (celle-ci étant une nouveauté pour certains d'entre nous). Elle permet de travailler sur le même code en simultané sur différents PC, elle enregistre toutes les modifications effectuées par chaque membre, et ainsi permet les retours en arrière en cas de besoin.

Le but principal de ce projet est donc d'obtenir une simulation dans laquelle l'élève pourra implémenter son réseau de Petri. Le réseau de Petri devra donc répondre à un scénario imposé par l'encadrant(e).

Les objectifs définis pour cette année sont donc :

- Migrer l'ancienne version de ROS (Indigo) vers la nouvelle version (Kinetic)
- Modéliser la saisie d'objet dans la simulation Coppeliasim
- Optimiser la simulation et le code
- Supprimer les intelligences données à certains éléments de la maquette (ces intelligences ne sont pas censées exister et doivent être créées par les étudiants via le réseau de Petri)
- Mettre en place un Checker, signalant les incohérences générées par l'étudiant lors du test de son réseau de Petri.
- Mettre en place une documentation de notre travail
- Mettre en place un sujet pour les étudiants ainsi qu'une liste des fonctions de haut niveau



### 3 Développement et optimisation de la simulation

#### 3.1 Simplification des modèles dynamiques

Lors du lancement de la simulation sur V-rep, nous avons remarqué un message d'erreur spécifiant que des objets "non-purs" et "non-convexes" simulés dynamiquement pouvaient ralentir la simulation. En utilisant le mode de visualisation dynamique, nous nous sommes aperçus que les rails et les navettes étaient constitués d'objets complexes (une multitude de polygones accolés) qui génèrent des calculs de collisions trop important. En vert sur les images ci-dessous, on peut apercevoir chacun de ces polygones.

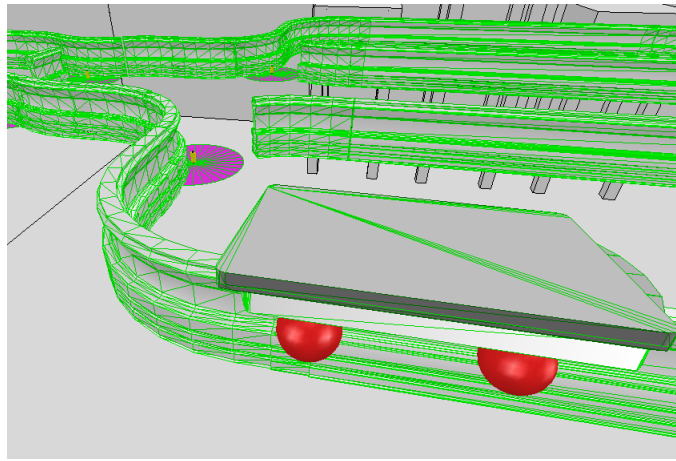


FIGURE 4 – Rails et navettes non-convexes et non-purs

Nous avons donc modifié ces objets afin de les rendre plus simples, en ajoutant uniquement des formes pures tels que des cylindres et des cubes. Ces nouvelles formes se superposent avec les anciennes.

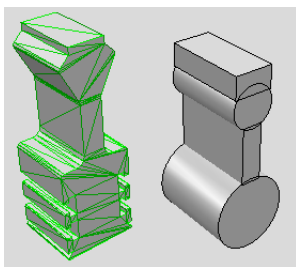


FIGURE 5 – Ancien (à gauche) et nouveau (à droite) profil de rail

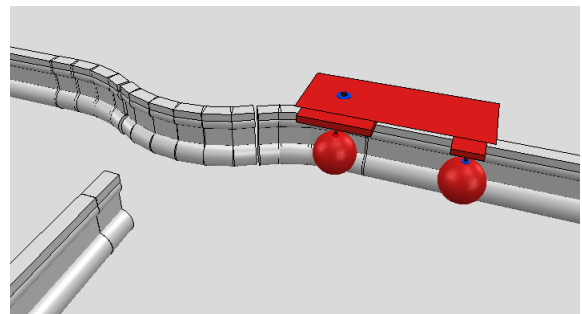


FIGURE 6 – Rails et navette purs

Après modification des rails et des navettes, nous obtenons un gain de performance en simulation. De plus, nous avons gardé les anciens modèles de rails et de navette, mais nous simulons dynamiquement que les formes simples. Ainsi, cette modification n'affecte pas le visuel de la simulation.

## 3.2 Modélisation des produits sur V-rep

Dans les anciens projets longs, les étudiants coloraient la plateforme des navettes et des postes d'une certaine couleur pour simuler chaque produit. Cette méthode est intéressante mais pas assez proche de la réalité. De plus, les successions de tâches sur un même produit n'étaient pas très visibles.



FIGURE 7 – Navette avec un produit F

Notre objectif est de créer des produits plus proches de la réalité, des blocs de produits, qui pourront être manipulés par les robots. Ainsi, notre méthode permettrait de superposer plusieurs blocs de couleurs, tandis que l'ancienne méthode ne permettait de colorer la plateforme de la navette que d'une seule couleur et qui, malgré un système de dégradé pas forcément très lisible, ne permettait de voir que la dernière tâche effectuée sur le produit. Ainsi, impossible de voir si l'on a effectué une mauvaise tâche sur un produit.

Nous nous sommes d'abord tournés vers des produits soumis aux propriétés physiques, et donc avec lesquels les robots pourraient interagir. Cependant, nous nous sommes rendus compte que cette méthode est faisable mais prendrait plus de temps de développement que prévu, en plus d'occasionner davantage de calculs pour le simulateur. Nous avons donc choisis de créer des blocs non-physiques.

### 3.2.1 Produits physiques

Dans cette première approche nous avons pensé à modéliser la physique des produits, qui nous faciliterait beaucoup de choses dans la programmation de ROS. En effet, en colorant les plateformes des navettes on doit connaître leur identité et donc savoir à tout instant quelle navette se trouve à quel poste afin de colorer la bonne. En modélisant la physique des produits on n'aurait plus à se soucier de cela, car on n'aurait pas à modifier des couleurs lors d'un déplacement de produits.

Nous devons choisir un produit qui ait une forme permettant de les empiler entre eux. Nous avons donc dans un premier temps opté pour la forme suivante :

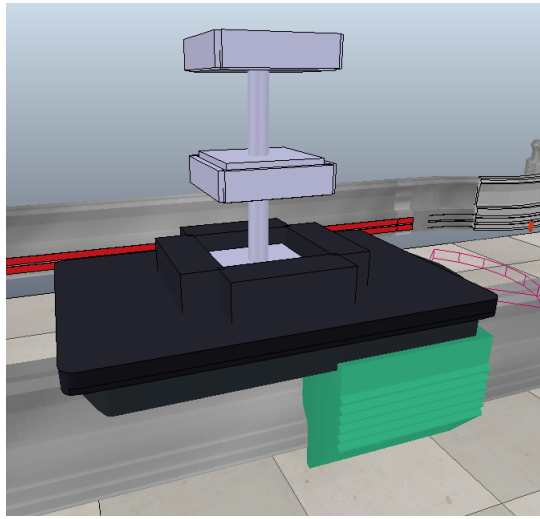


FIGURE 8 – 2 produits physiques empilés

Après plusieurs essais en simulation, nous constatons que l'empilement de ces produits est stable, mais que l'empilement glisse sur la surface de la navette pendant son déplacement. La première idée qui nous ait venue est d'ajouter des cales sur les navettes afin de stabiliser les produits, comme on peut le voir sur l'image. Cependant, l'équipe qui a mené ce projet long l'année dernière avait déjà tenté cela sans succès. Le produit traverse la cale et le robot ne parvient ensuite pas à extraire le produit de la navette.

Bien que cet empilement de produits soit stable, il est difficile par la suite de faire prendre l'empilement complet par un robot. Nous avons donc dû imaginer une autre solution d'empilement. Cette solution doit permettre au robot d'empiler facilement les produits sur la navette, mais aussi de récupérer l'empilement final facilement. Nous avons donc opté pour des produits en forme d'anneaux qui s'empilent sur un mât :

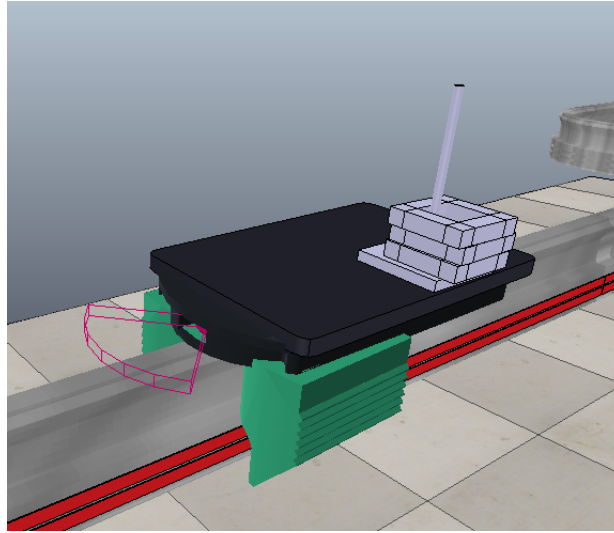


FIGURE 9 – 3 produits physiques empilés sur le mât

Ici nous n'avons pas mis de cale, le mât qui supporte l'empilement des produits est donc instable. Notre approche pour palier ce problème est d'ajouter des "dummy", ce sont des points de connections sur V-Rep qui permettent de lier des objets entre eux. Cette solution est intéressante et pourra être testée par une équipe de projet long ultérieure, mais nous ne nous sommes pas attardés sur cette approche par manque de temps. Nous avons opté pour le modèle non-physique des produits afin de pouvoir avancer rapidement dans le projet.

### 3.2.2 Produits non-physiques

Nous avons choisi de modéliser les produits sans leur attribuer de contrainte physique. Les produits sont donc simplement des cubes non-physiques soumis à aucune force.

Ainsi, un empilement de 6 cubes est présent sur chaque poste et chaque navette dès le début de la simulation. Il est néanmoins entièrement transparent et donc invisible. Notre approche consiste simplement à rendre visible et colorer au bon moment chaque étage de cet empilement pour simuler un déplacement d'objet. Le cube à la base représente le type de produit et les cubes suivants représentent les tâches effectuées sur les postes. Ceci permet de voir facilement l'historique des tâches effectuées sur un produit. Lors du lancement d'une tâche, le cube apparaît d'abord en semi-transparence et se colorie entièrement à la fin de la tâche. Il est possible d'augmenter le nombre de cubes. Une explication détaillée pour augmenter le nombre de cubes maximal se trouve dans la documentation jointe à ce rapport.

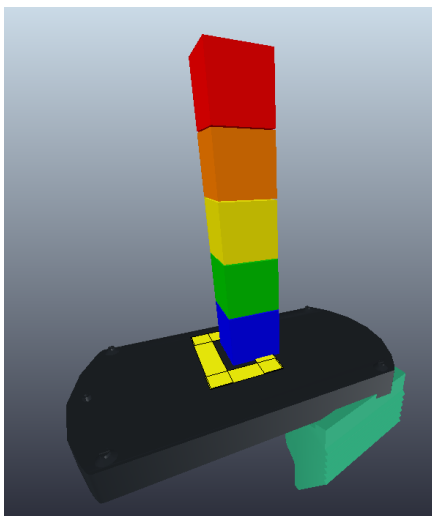


FIGURE 10 – Navette avec empilement de cubes

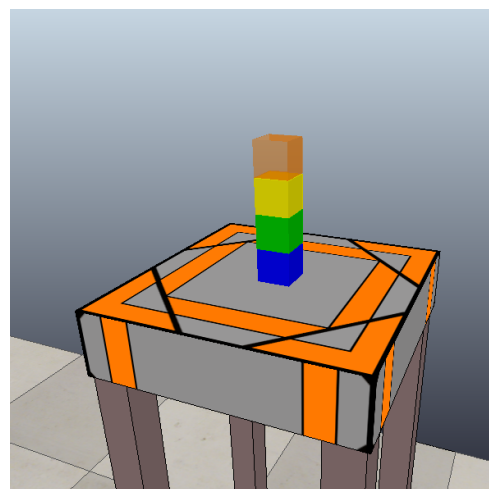


FIGURE 11 – Tâche en cours sur un poste

En utilisant cette méthode, nous devons donc savoir de quelle navette il s'agit pour pouvoir colorer son empilement. Dans ce but, une nouvelle node a été créée : le Shuttle Manager. Son fonctionnement sera détaillé dans la partie développement du code, mais elle permet de fournir à tout instant aux robots les positions des navettes sur le circuit.

## 4 Développement et optimisation du code

En parallèle de l'optimisation de la simulation, nous nous sommes appropriés le code ROS afin de pouvoir y développer les améliorations souhaitées et de le rendre aussi proche que possible du cahier des charges.

### 4.1 Nettoyage du travail effectué précédemment

Dans un premier temps, nous avons tenté de repartir du code développé lors du dernier projet long, en 2019. Un lien github nous avait été fourni.

Malheureusement, nous avons eu de grandes difficultés à réutiliser ce code, pour des raisons indépendantes de notre volonté. En effet, la compilation sous ROS des codes sources entraîne l'apparition de deux dossiers propres à ROS : build/ et devel/. Ces dossiers compilés permettent ensuite l'exécution de toutes les nodes mais sont **uniques à chaque machine** (nom de session, d'ordinateur, chemin d'accès etc.).

L'archive 2019 était très peu réutilisable car ces dossiers build/ et devel/ étaient eux aussi sauvegardés et nécessaires. Il est dangereux de travailler avec ces fichiers dans son espace de travail, car ROS ne fait qu'ajouter ou modifier des compilations partielles, et ne supprime jamais de morceaux compilés d'anciennes sources.

Nous avons donc récupéré le travail de la session 2018, dans l'espoir de pouvoir y incorporer peu à peu les modifications de 2019, mais ce dernier ne compilait pas du premier coup. En effet, il a fallu que nous réglions un problème d'interdépendance entre package, dûe une nouvelle fois à des travaux réalisés à moitié sur les dossiers build/ et devel/ précompilés.

Après une étude du code et du rapport 2019, nous avons décidé de ne pas reprendre de code 2019 mais uniquement certaines idées telle que l'accélération de ROS en enlevant des boucles infinies et le fait de rajouter un produit sur un poste et non sur une navette.

Nous avons ensuite parcouru les codes de chaque package, et avons remarqué que certains codes étaient assez répétitifs. En effet, on retrouvait plusieurs fois le même code pour chaque robot, chaque poste, chaque aiguillage. Nous les avons donc fusionnés en un seul programme en utilisant l'intérêt du C++ : l'orienté objet.

Avec ces changements, le projet est maintenant moins lourd et plus clair à la lecture.

### 4.2 Suppression des intelligences

Après une étude du travail effectué précédemment, nous avons vu que les éléments de la maquette en simulation avaient une intelligence et communiquaient entre eux ce qui n'est pas possible dans la réalité. En effet, un produit connaissait son lieu de destination et le communiquait aux aiguillages, ainsi les aiguillages se tournaient automatiquement pour satisfaire cette destination.

Nous voulons que l'élève oriente lui-même les aiguillages grâce à son réseau de Petri. Nous avons donc supprimé cette intelligence et créé des fonctions de haut niveau permettant de commander les aiguillages (ces fonctions sont détaillées dans le fichier annexe concernant les fonctions de haut niveau).

De même, plusieurs packages (Tâches et Scheduler) avaient accès à beaucoup trop d'informations. Nous pensons qu'une simulation doit simplement obéir à son utilisateur, et donc l'étudiant, sans juger la

pertinence des actions demandées par celui-ci. Après mûre réflexion, nous avons simplement supprimé ces packages.

### 4.3 Ajout de "manager"

L'intelligence des navettes étant supprimée, nous ne connaissons plus la localisation des navettes dans le circuit, or pour que la simulation 3D se passe bien il faut savoir en couche basse quelle navette se trouve devant un robot qui souhaite effectuer une opération. Pour cela, on a créé un "Shuttle Manager", ce dernier a pour but de suivre chaque navette en observant l'état des capteurs et l'état des aiguillages.

Ainsi chaque navette est suivie et permet le bon fonctionnement des déplacements de produits dans la simulation. Évidemment ce suivi de navette est interne au programme, et l'élève n'y aura pas accès.

Ce suivi se base sur un principe de prédiction, lorsque qu'une navette passe par un capteur, on enregistre sa position sur une zone "inter-capteur" et on l'attend à la prochaine zone. S'il y a un aiguillage dans cette zone inter-capteur, alors on enregistre l'état de cet aiguillage avant l'entrée de la navette dans celle-ci pour définir le prochain capteur de sa destination. Lorsque 2 navettes sont sur la même zone "inter-capteur", alors la première navette qui entre est la première qui en sort.

Cependant ce système a ses limites : si l'orientation de l'aiguillage est trop tardive (changement d'orientation après que la navette ait dépassé le capteur précédent), il est possible qu'il y ait une erreur de suivi de navette. Il ne faut donc jamais passer le capteur précédent un aiguillage pendant que ce dernier est encore en mouvement. Si cette erreur se produit, la dépose de produit d'un robot sera erronée (la mauvaise navette recevra le produit).

## 5 Migration de ROS Indigo vers ROS Kinetic

### 5.1 Imitation des services avec des topics

Lors du passage vers Kinetic, les services du package `vrep_common` et du plugin ROS de V-Rep permettant de faire le lien entre le simulateur et le projet ne sont plus utilisables. Ces services permettaient de bouger les robots, de demander la position d'un robot, de demander des handle, le temps de simulation, de colorer des objets... . Nous avons 2 solutions : rajouter nous même les services dans le plugin et le recompiler, ou remplacer ces services par des topics. Nous avons choisi la deuxième option, c'est à dire imiter le fonctionnement des services avec des topics : on publie sur un topic qui est écouté par V-rep, on attend jusqu'à avoir une réponse de V-rep dans un topic de retour. Dans V-rep, un callback est déclenché par le premier message, il effectue les actions voulues avant d'envoyer la réponse attendue par le côté ROS. Il fallait bien faire attention à ce que chaque node ait ses propres topics (par exemple pour chaque robot, des topics différents doivent être créés). En effet, il faut absolument éviter de lancer 2 appels simultanés d'un même service. Afin de valider le bon fonctionnement de cette implémentation, nous avons réalisé des tests unitaires en écrivant directement sur les topics écoutés par V-rep (à l'aide de `rostopic pub`) et en affichant la réponse de V-rep dans le topic de retour (à l'aide de `rostopic echo`).

Du côté ROS, la structure utilisée est la suivante :

```
pub = noeud.advertise<type_msg>(nom_topic,100);
sub = noeud.subscribe(nom_topic_retour,100,&Callback);
[...]
pub.publish(msg);
while(!rep && ros::ok())
{
    ros::spinOnce();
    loop_rate->sleep();
}
rep = false;
var = value;
[...]
void Callback(type_msg_retour msg)
{
    value = msg->data;
    rep = true;
}
```

Les variables `rep` et `value` étant des variables globales ou des attributs si on travaille dans une classe.

### 5.2 Changement de l'interface graphique

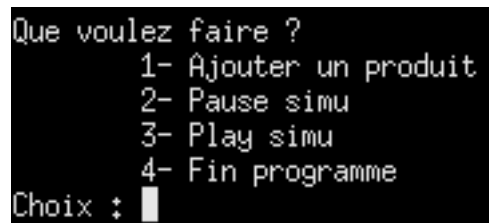
Le passage à Kinetic a également entraîné un conflit de version avec OpenCV, la bibliothèque graphique qui avait été utilisée pour réaliser l'interface homme-machine avec la simulation. Nous avons choisi dans un premier temps, de faire une nouvelle interface rapide en ligne de commande car nous



n'avions pas le temps de reprendre l'ancienne interface et de conserver uniquement la fenêtre de visualisation de la simulation. Nous avons donc perdu l'interface affichant l'état des capteurs et actionneurs de la maquette en temps réel qui avait été faite les années précédentes.

De plus, nous avons constaté que la caméra sur la simulation permettant de récupérer l'image de celle-ci consommait beaucoup de ressources et entraînait donc un ralentissement de la simulation. En effet, sur les 100ms de calcul de Coppelia par boucle, 40ms sont entièrement consacré au flux vidéo. Après discussion, nous avons choisi de ne plus utiliser la caméra mais d'ouvrir en mode fenêtré le logiciel de simulation. Nous avons trouvé un paramètre permettant d'interdire la sauvegarde de la scène et ainsi éviter que l'étudiant ne fasse des modifications et enregistre ensuite (Lock scene after next scene save). De plus, nous avons modifié les paramètres de lancement du logiciel afin de limiter les menus ouverts dans la fenêtre de simulation. Pour cela, nous avons modifié les lignes 47 et 48 du fichier `usrset.txt` se trouvant dans le répertoire `CoppeliaSim/system/`.

Maintenant, nous avons une fenêtre pour la simulation, une pour l'interface homme machine qui permet à l'utilisateur de choisir les actions qu'il souhaite faire. Nous avons aussi mis en place une fenêtre affichant les marquages actifs du réseau de Petri que l'étudiant aura programmé pour faciliter la visualisation du déroulement du Petri.



```
Que voulez faire ?
  1- Ajouter un produit
  2- Pause simu
  3- Play simu
  4- Fin programme
Choix : █
```

FIGURE 12 – Interface homme machine

Le code de l'ancienne interface homme-machine est encore présent dans le projet et pourra être adapté à la nouvelle version d'OpenCV afin de retrouver l'interface graphique qui avait été faite.

## 6 Mise en place du checker

Dans cette dernière partie nous allons présenter le développement du checker. Celui-ci sera indispensable à l'étudiant pour vérifier le bon déroulement de la simulation, et de vérifier si le cahier des charges a été respecté. Nous joindrons avec ce compte rendu une proposition de sujet pour les étudiants.

### 6.1 Sujet

Nous sommes donc partis du sujet proposé par la session 2019, puis nous avons rajouté/modifié certains éléments.

On y retrouve dans un premier temps une description de l'environnement de travail (de la cellule), afin que l'étudiant puisse comprendre ce qui va lui être demandé.

Il y a ensuite un plan du circuit monorail avec les emplacements des capteurs, ce schéma est essentiel pour que l'étudiant puisse connaître les numéros des capteurs qu'il utilisera dans son réseau de Petri, ainsi que les numéros des robots et des postes.

Puis, il y a le cahier des charges du TER. On y décrira les objectifs que l'étudiant devra atteindre ainsi que les outils qu'il devra utiliser. Il sera accompagné d'un scénario de production fourni par l'encadrant, et ce scénario devra être implémenté dans le fichier "ProductConfiguration.config" afin que le Checker puisse vérifier que l'étudiant ait bien respecté ce scénario.

Et enfin on retrouve la liste des fonctions de haut niveau nécessaires à la réalisation du réseau de Petri.

### 6.2 Le fichier Config

En plus du cahier des charges global, l'encadrant pourra définir un cahier des charges spécifique à chaque élève grâce au fichier "ProductConfiguration.config". Il permettra de rentrer une séquence de production à respecter.

**Start**

**2 : 1 4 : 4 5 : 1**

**6 : 7 6 5 : 3 6 3 : 1**

FIGURE 13 – Exemple de configuration

Dans cet exemple, l'élève devra produire un produit de type 2 sur lequel seront appliquées les tâches 1 puis 4 respectivement durant 4s et 5s, et devra produire 1 unité de ce type. On suit le même raisonnement pour chaque ligne de ce fichier Config.

Le Checker va ensuite vérifier le respect de ces consignes par le fichier log généré par la simulation.

### 6.3 Le fichier log

Le fichier log.txt est généré par ROS au cours de la simulation. ROS va écrire chaque action-clé du scénario dans le fichier Log (par exemple l'ajout d'une tâche sur un produit), puis le Checker va lire ligne par ligne ces informations afin d'en conclure si les règles de fabrication ont bien été respectées ou non.

Pour la lecture du log nous avons mis en place une convention. Lorsque le Checker lira une ligne, il va identifier le premier mot qui lui indiquera quels paramètres contient le reste de la ligne. Ainsi, le premier mot est un identificateur qui indiquera au checker comment lire la ligne et donc comment traiter les informations qui s'y trouvent. Les différents messages pouvant être écrit dans le log sont les suivants :

Mot identificateur	Paramètres	Signification
OperationPosteVide	n° poste	Erreur - opération sur un poste vide
EcrasementProduit	n° robot	Erreur - un produit est écrasé
OperationProduitPlein	n° poste	Erreur - opération sur un produit déjà plein
PerteNavette	n° tronçon	Erreur - une navette a été perdu
NewProduct	n° produit : temps	Info - un produit a été généré
TempoT	n° produit : n° tâche : durée	Info - une tâche a été faite sur un produit
Sortie	n° produit : tâche1 2 3 : durée 1 2 3 : temps	Info - un produit a été évacué

A titre d'exemple, si on fait apparaître un produit 3 sur n'importe quel poste au temps 12.3s, il sera écrit dans le log :

NewProduct : 34 : 12.3

Le checker lit le log ligne par ligne, donc quand il arrivera a cette ligne et qu'il identifiera le mot "NewProduct" il saura alors reconnaître les paramètres "n° produit" et "temps"

Il est important de noter que ROS identifie les numéros produits et de tâches d'un façon bien spécifique, sachant que les deux sont représentés par des cubes de couleur :

- Le produit est identifié avec un nombre à 2 chiffres. Le chiffre des dizaines indique le numéro du produit, et le chiffre des unités vaut '4' et indique que le cube est de type 'produit'
- La tâche est identifiée avec un nombre à 2 chiffres. Le chiffre des dizaine indique le numéro de la tâche, qui correspond également au numéro du poste où à lieu la tâche, tandis que le chiffre des unités indique l'état de la tâche. Il y a 2 états pour une tâche, soit la tâche est terminée donc visuellement le cube est plein et le chiffre des unités vaut '3', ou alors la tâche est en cours donc le cube est semi-transparent et le chiffre des unité vaut '2'

Sur une pile de 4 cubes, le produit est représenté par le premier cube du bas (n° produit de 1 à 6). Sur ce cube se trouve 3 cubes qui symbolisent les tâches qui ont été faites sur le produit (n° tâche de 1 à 8). Donc en respectant la syntaxe présenter juste avant, le numéro attribuer aux 4 cubes sont les suivants :

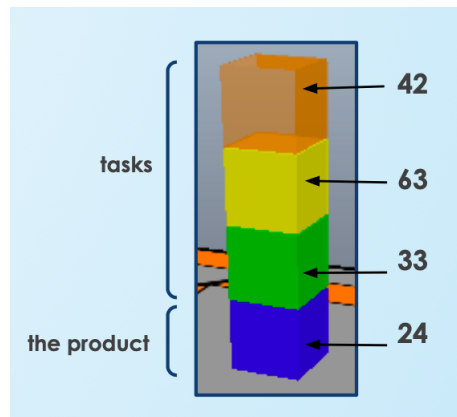


FIGURE 14 – 4 cubes et leur syntaxe correspondante

## 6.4 Le Checker

Pour aider l'étudiant à trouver ses erreurs, nous mettons en place un "Checker". Ce dernier consiste à écrire dans la console les erreurs commises dans le scénario créé par l'étudiant.

Ce Checker a été codé en python car ce langage propose des commandes plus simples en ce qui concerne la lecture de fichiers textes. Le checker lit donc les fichiers log et config, traite leur données en les comparant, puis affiche dans le terminal la validation de la séquence des tâches sur un produit et les erreurs s'il y en a :

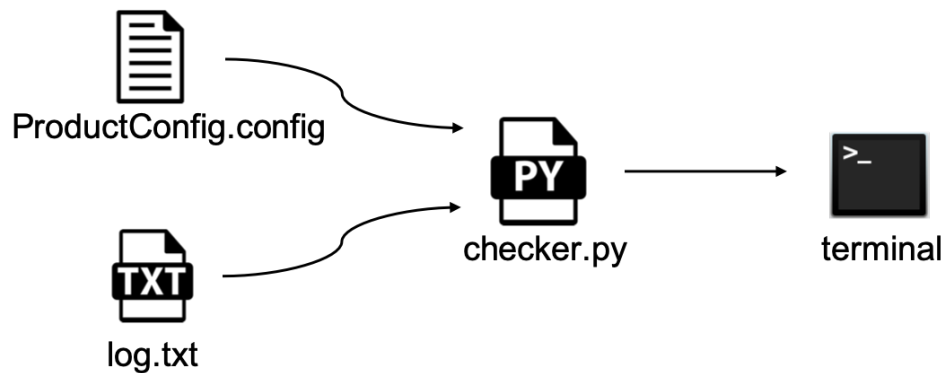


FIGURE 15 – Fonctionnement du checker

Notre Checker est donc en mesure de repérer les erreurs suivantes :

- Erreur de mauvaise déclaration dans le fichier .config :
  - n° produit n'est pas compris entre 1 et 6
  - destination du produit n'est pas entre 1 et 8 (poste)
  - nombre de tâches par produit supérieur à 5
  - le nombre de tâches est différent du nombre de durées de tâches

- Erreur dans la simulation par lecture du Log :

- opération sur un poste vide numéro X
- opération sur un produit qui a déjà les 5 tâches maximales (6 cubes empilés) sur le poste X
- produit écrasé par le robot X
- le type de produit ne finit pas par un '4'
- enchaînement des tâches d'un produit ne correspond pas à celui attendu d'après le fichier Config
- une des tâches n'étaient pas terminées lors de l'évacuation d'un produit
- une tâche ne finit pas par un '2' ou '3'
- évacuation commandée alors qu'il n'y a pas de produit sur le poste 3
- la durée d'une tâche n'est pas correcte car différente de celle attendu d'après fichier Config
- un produit qui n'est pas sensé apparaître est quand même apparu
- un produit attendu n'est pas apparu le nombre de fois désiré
- un produit attendu n'est pas sorti le nombre de fois désiré

En revanche, si aucun message d'erreur n'a été généré et que tout s'est bien passé, on affiche : le produit créé, le nombre de fois, la séquence de tâches, et le temps moyen qu'un produit a passé dans l'atelier de production.

## 7 Conclusion

Nous avons pu satisfaire la majeure partie de nos objectifs dans le temps imparti, et ainsi proposer un sujet opérationnel pour les étudiants. En effet notre travail pourra directement être utilisé pour le TER avec le sujet fourni et un scénario de production au choix.

Durant ce projet de 6 semaines, nous avons pu améliorer nos compétences de gestion du travail d'équipe ainsi que nos compétences en informatique (C++, Python, ROS, CoppeliaSim ...). Ce fût une expérience fort enrichissante en tant que futur ingénieur car nous avons pu nous consacrer à temps plein sur un projet technique pendant plusieurs semaines, ce qui nous prépare pour notre projet de fin d'étude et notre futur métier.

Bien que le projets ce soit bien réalisé nous avons rencontré quelques difficultés comme la prise en main des différents logiciels qui n'étaient pas connus par tous ou la réutilisation des travaux antérieurs qui n'a même pas été possible avec l'archive 2019.

Il reste encore quelques améliorations à faire, telles que l'interfaçage effectif avec la vraie maquette, l'ajout d'une interface de réseau de Petri générant automatiquement le code C++ (ou python) pour ROS ou encore une amélioration de la simulation afin de pouvoir réellement déplacer des objets physiques et se rapprocher de la maquette réelle. Il serait également souhaitable de paralléliser les tâches dans un/des nodes à part.