



Haute école d'ingénierie et d'architecture Fribourg  
Hochschule für Technik und Architektur Freiburg

Filière Informatique  
Projet complémentaire  
Semestre de printemps 2017

---

# Lego-I

---

Owen McGill

---

Mandants : iCoSys

iCoSys  
Institute of Complex Systems

SeSi

SeSi  
Sustainable Engineering  
Systems Institute

Superviseurs : Pierre Kuonen  
Nicolas Rouvé  
Beat Wolf

---

Fribourg, Juin 2017

**Hes·so**  
Haute Ecole Spécialisée  
de Suisse occidentale  
Fachhochschule Westschweiz

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	SmartFactory . . . . .	3
1.2	LEGO Digital Designer . . . . .	3
1.3	But du projet . . . . .	3
1.4	Structure du projet . . . . .	3
<b>2</b>	<b>Analyse</b>	<b>5</b>
2.1	Ordre des briques dans les fichiers . . . . .	5
2.2	Positionnement des briques . . . . .	7
2.3	Origine des briques . . . . .	7
2.4	Taille des briques . . . . .	8
2.5	Format intermédiaire . . . . .	10
2.5.1	Déplacements . . . . .	10
2.6	Quaternion . . . . .	11
2.7	Platines . . . . .	11
<b>3</b>	<b>Implémentation</b>	<b>13</b>
3.1	Langage de développement . . . . .	13
3.2	Fonctions principales . . . . .	13
3.2.1	convert . . . . .	13
3.2.2	parse . . . . .	14
3.2.3	construct . . . . .	15
3.3	Programme complet . . . . .	15
<b>4</b>	<b>Conclusion</b>	<b>17</b>
4.1	Prochaines étapes nécessaire . . . . .	17
4.1.1	Construire sur plusieurs niveaux . . . . .	17
4.1.2	Construire avec des briques non-horizontales . . . . .	17
4.1.3	Généré automatiquement la platine de départ . . . . .	18
4.2	Idées d'amélioration . . . . .	18
4.2.1	Serveur de conversion . . . . .	18
4.2.2	Utilisation de QR codes . . . . .	18
4.2.3	Modélisation en VR/AR . . . . .	19
4.3	Avis personnel . . . . .	19
<b>A</b>	<b>Annexes</b>	<b>24</b>
A.1	Code source . . . . .	24
A.1.1	convert(matrix, origin) . . . . .	24
A.1.2	convertV2(matrix, origin) . . . . .	24
A.1.3	parse(file) . . . . .	25
A.1.4	construct(starting_plate, build, output) . . . . .	25
A.1.5	Classe Lego . . . . .	26
A.2	Versions des logiciels/librairie . . . . .	28
A.3	Manuel d'utilisation . . . . .	28
A.3.1	Installation des librairies . . . . .	28

A.3.2 Exécution du programme . . . . .	28
<b>B Structure de la clé usb</b>	<b>30</b>

# 1 Introduction

L'institut Sustainable engineering Systems institute (SeSi) s'est récemment doté d'une ligne de montage robotisée SmartFactory de l'entreprise Festo permettant d'assembler des pièces. Une telle chaîne pourrait, par exemple, être utilisée pour assembler les éléments d'un Smartphone. Nous désirons utiliser cette ligne d'assemblage dans le cadre d'expositions, comme par exemple les journées "porte ouverte", pour d'une part présenter les compétences de l'école dans le domaine de l'industrie 4.0 et d'autre part, pour attirer des étudiants dans notre école. La réalisation de ce dernier objectif suppose que nos démonstrations attirent les jeunes gens en y intégrant un aspect ludique.

Nous proposons donc d'utiliser cette chaîne d'assemblage pour monter des structures en LEGO, qui auront au préalable été définies par l'utilisateur.

Dans ce chapitre nous allons rapidement passer en revue les différentes applications et machines qui sont impliquées dans ce projet et aussi le but de ce projet.

## 1.1 SmartFactory

La SmartFactory est une usine Cyber-Physique[1] construite par Festo avec des divers modules possible, un bras robot six axes d'ABB, un système de convoyeur pour le transport de pièce entre les différents modules et plus.

## 1.2 LEGO Digital Designer

LEGO Digital Designer (LDD) est une application mise à disposition par le groupe LEGO pour créer des modèles LEGO sur un ordinateur avec toutes les différentes pièces produites par LEGO. Ces modèles peuvent ensuite être partagés sur LEGO.com, et il est aussi possible de télécharger des modèles créés par d'autres utilisateurs depuis ce site.

## 1.3 But du projet

Le but de ce projet est d'étudier la faisabilité de convertir les fichiers générés par LDD dans un format permettant de contrôler le bras robot six axes ABB dans la SmartFactory pour construire les modèles créés dans LDD.

## 1.4 Structure du projet

Ce projet est structuré de la manière suivante :

- Analyse : Nous analyserons les différentes parties du fichier généré par LDD et du format intermédiaire que nous générerons pour contrôler la ligne d'assemblage.
- Implémentation : Nous discuterons de l'implémentation qui a été faite, et les raisons derrière les choix liés à l'implémentation.
- Conclusion : Nous conclurons ce rapport par les difficultés rencontrées et les étapes nécessaires pour finir ce projet et d'avoir un système que nous pour-

rons ensuite utiliser afin d'attirer des étudiants à l'Haute école d'ingénierie et d'architecture de Fribourg

Dans ce chapitre nous avons passé en revue le but de ce projet ainsi que les outils qui seront utilisé et la structure du rapport.

## 2 Analyse

Le programme LEGO Digital Designer génère des fichiers LEGO Exchange Format (LXF) qui sont des archives ZIP contenant une image représentant le modèle et un fichier LEGO Exchange Format Markup Language (LXFML) qui est un type de XML. Nous allons nous concentrer sur les fichiers LXFML et les analyser, pour ensuite savoir comment les convertir dans un format approprié pour l'usine SmartFactory, et aussi voir les autres éléments nécessaires à ce projet.

### 2.1 Ordre des briques dans les fichiers

L'ordre des briques dans les fichiers LXFML joue un rôle important dans l'ordre de construction d'un modèle LEGO. Il n'est pas possible de construire un modèle LEGO dans un ordre quelconque, les modèles LEGO doivent être construit de bas en haut, puisque les briques s'emboîtent avec les tétons qui se trouvent sur le dessus des briques dans les espaces vide se trouvant en dessous des briques.

Pour cette raison il est important d'analyser l'ordre des briques dans les fichiers LXFML. Pour cette analyse nous avons construit plusieurs modèles LEGO dans LDD et ensuite analysé si l'ordre dans les fichiers était soit l'ordre de construction du modèle ou l'ordre dans lequel les pièces ont été placé dans LDD.

Nous avons découvert que les fichiers LXFML non pas d'ordre précise qu'on ait pu trouvé et qu'il n'y a pas d'ordre utile pour nous. La figure ?? est un extrait d'un fichier LXFML et la figure 2 une image du modèle avec une numérotation indiquant l'ordre de placement.

FIGURE 1 – Extrait d'un fichier LXFML avec ordre de placement des briques

---

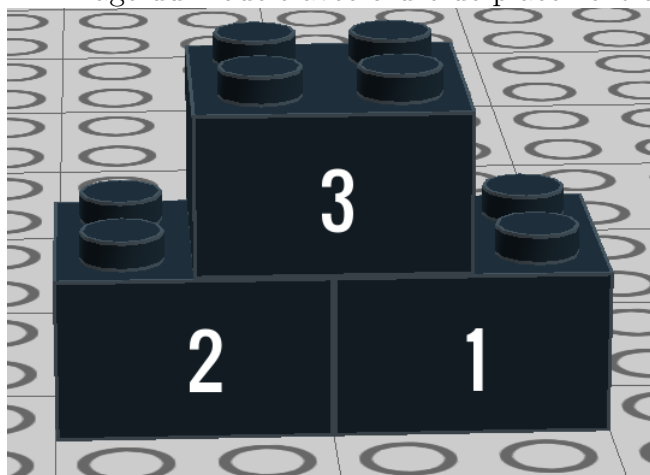
```

<Bricks cameraRef="0">
  <!-- Brique 2 -->
  <Brick refID="0" designID="3003" itemNos="300326">
    <Part refID="0" designID="3003" materials="26,0,0"
      decoration="0,0">
      <Bone refID="0"
        transformation="1,0,0,0,1,0,0,0,1,-0.4,0,-0.4">
      </Bone>
    </Part>
  </Brick>
  <!-- Brique 3 -->
  <Brick refID="1" designID="3003" itemNos="300326">
    <Part refID="1" designID="3003" materials="26,0,0"
      decoration="0,0">
      <Bone refID="1"
        transformation="1,0,0,0,1,0,0,0,1,-0.4,0.96,0.4">
      </Bone>
    </Part>
  </Brick>
  <!-- Brique 1 -->
  <Brick refID="2" designID="3003" itemNos="300326">
    <Part refID="2" designID="3003" materials="26,0,0"
      decoration="0,0">
      <Bone refID="2"
        transformation="1,0,0,0,1,0,0,0,1,-0.4,0,1.2">
      </Bone>
    </Part>
  </Brick>
</Bricks>

```

---

FIGURE 2 – Image du modèle avec ordre de placement des briques



Dû au fait que les fichiers ne possèdent pas un ordre logique, nous avons décidé que pour ce projet nous examinerons seulement des modèles ayant une hauteur d'une brique. Nous avons pourtant déjà envisagée quelques solutions pour résoudre le problème de l'ordre des pièces dans l'axe Z, une solution étant l'algorithme du Peintre [2], un algorithme utilisé dans le domaine d'infographie (Computer Graphics) pour résoudre un problème semblable.

## 2.2 Positionnement des briques

Le positionnement des briques est l'information la plus importante pour ce projet, car sans comprendre la position des briques dans les fichiers LXFML, nous ne pourrions pas indiquer au robot où prendre les briques depuis la platine de départ et où les placer pour assembler le modèle.

La position des briques est caractérisée par une matrice de transformation affine [3], qui contient une matrice de rotation  $3 \times 3$  et un vecteur  $3 \times 1$  et complétée par une ligne  $4 \times 1$  contenant  $[0 \ 0 \ 0 \ 1]$ .

L'information de positionnement se trouve dans l'attribut *transformation* de l'élément *Bone* pour chaque brique dans les fichiers LXFML, la matrice est représentée par une liste de 12 nombres à point flottant.

L'ordre des numéros dans la liste est colonne X de rotation, colonne Y de rotation, colonne Z de rotation et vecteur de position, chaque partie est composée de 3 numéros. La table 1 est la représentation sous forme de matrice de transformation affine.

TABLE 1 – Index de la liste de transformation sous forme de matrice de transformation affine

transformation[0]	transformation[3]	transformation[6]	transformation[9]
transformation[1]	transformation[4]	transformation[7]	transformation[10]
transformation[2]	transformation[5]	transformation[8]	transformation[11]
0	0	0	1

La position des briques est donc une position absolue dépendant de l'origine de la plaque virtuelle de LDD, cette position devra être convertie et adaptée pour le robot, quelque chose qui sera vu plus tard dans ce rapport.

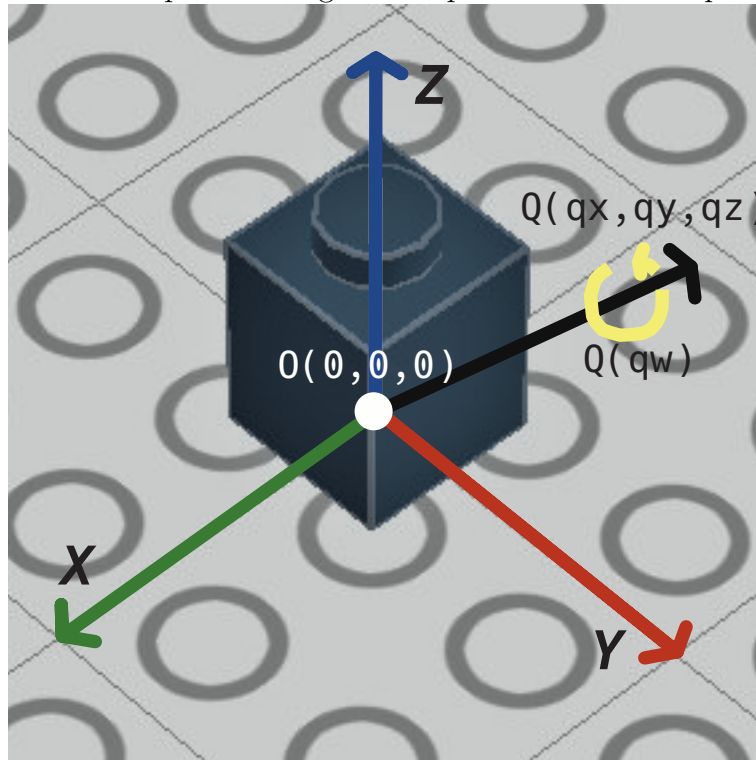
## 2.3 Origine des briques

L'origine des briques joue un rôle important dans le positionnement des briques et aussi pour les positions qui seront transmises au robot. La raison que l'origine est si importante est qu'il faut savoir où sur la pièce la position fait référence.

Dans le cas des fichiers LXFML, l'origine se trouve toujours en-dessous du téton se situant dans le coin inférieur de gauche en regardant dans la direction  $-X$ . Nous avons trouvé cette information en éditant manuellement des fichiers LXFML pour mettre la position à  $(0, 0, 0)$  car ce n'est pas possible de placer une brique à l'origine  $(0, 0, 0)$  du monde LDD, comme visible dans la figure [3]. Sur la figure on voit aussi la représentation d'un quaternion, qui sera discuté plus tard.



FIGURE 3 – Brique avec origine et représentation d'un quaternion

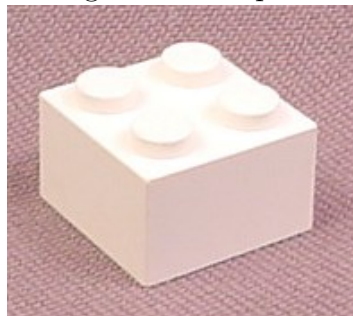


Du fait que l'origine des pièces se trouve à cet endroit nous n'aurons pas la nécessité à faire des grands changements aux positions contenues dans les fichiers LXFML sauf l'ajustement pour la taille.

## 2.4 Taille des briques

La taille des briques est important pour connaître l'échelle des pièces entre LDD et les briques physiques, ceci car il faudra adapté les positions dans les fichiers LXFML au monde physique.

La figure 4 montre une briques LEGO  $2 \times 2$  que nous avons mesuré, les dimensions étant  $16.0mm \times 16.0mm \times 9.6mm$  et l'espacement entre 2 tétons est de  $8.0mm$ .

FIGURE 4 – Image d'une brique LEGO  $2 \times 2$  [4]

La figure 5 est un extrait du fichier LXFML et la figure 6 une image illustrant le

modèle contenant 2 briques LEGO  $2 \times 2$ , l'une à côté de l'autre utilisé pour trouver l'espacement entre les deux pièces. Cette espacement est de 1.6, en divisant par deux, nous trouvons l'espacement entre deux tétons, nous avons 0.8 qui est dix fois plus petit que la taille réelle donc nous avons maintenant l'échelle, qui est de 1: 10.

FIGURE 5 – Extrait d'un fichier LXFML pour trouver l'échelle des pièces

---

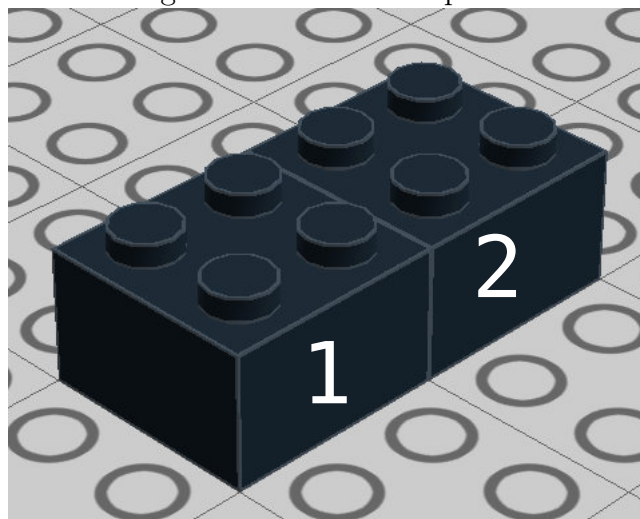
```

<Bricks cameraRef="0">
  <!-- Brique 2 -->
  <Brick refID="0" designID="3003" itemNos="300326">
    <Part refID="0" designID="3003" materials="26,0,0"
      decoration="0,0">
      <Bone refID="0" transformation="1,0,0,0,1,0,0,0,1,-1.2,0,1.2">
      </Bone>
    </Part>
  </Brick>
  <!-- Brique 1 -->
  <Brick refID="1" designID="3003" itemNos="300326">
    <Part refID="1" designID="3003" materials="26,0,0"
      decoration="0,0">
      <Bone refID="1" transformation="1,0,0,0,1,0,0,0,1,-1.2,0,-0.4">
      </Bone>
    </Part>
  </Brick>
</Bricks>

```

---

FIGURE 6 – Image du modèle utilisé pour trouver l'échelle



## 2.5 Format intermédiaire

Nous avons décidé de générer un format intermédiaire qui sera utilisé pour contrôler le robot. Ce format est basé sur le travail réalisé par Lucas Andrey pour son Travail Pratique Individuel (TPI) à l'École des Métiers Fribourg (EMF) [5].

Le format intermédiaire est composé d'un en-tête, suivi par des déplacements. La syntaxe des déplacements est composée du mot *movePiece* et deux positions, ceux-ci sont séparées par une virgule et terminé par un point-virgule.

### 2.5.1 Déplacements

Une position est composé de quatre éléments :

- Une position 3-dimensionnel, composée de 3 nombres à virgule dans l'ordre [X,Y,Z]
- Un quaternion, ceci composé de 4 nombres à virgule dans l'ordre [qx,qy,qz,qw]
- Les données de configuration des axes du robot, normalement toujours la même chose
- Les données de configuration des positions des axes externes, pas utilisé sur cette machine

La figure 7 comporte la syntaxe du format intermédiaire.

FIGURE 7 – Syntaxe du format intermédiaire

---

```
'movePiece' pos, pos;
```

pos = '['posXYZ, posOrientation, configurationData, externalJoints']'

posXYZ : Position en 3D, X, Y, Z. Z dirigé vers le haut.

posXYZ = '['float, float, float']'

posOrientation : Rotation définis avec un quaternion, l'ordre des composants étant (qx,qy,qz,qw).

posOrientation = '['float, float, float, float']'

configurationData : Configuration des axes du robot, normalement toujours la même chose.

configurationData = '['float, float, float, float']'

externalJoints : Configuration des positions des axes externes, pas utilisé sur cette machine.

externalJoints = '['float, float, float, float, float, float']'

---

Cette syntaxe fonctionne seulement si la procédure *movePiece* à été défini, dans le cas de la machine SmartFactory de l'HEIA-FR, ceci est déjà fait donc

nous avons pas dû le faire. Dans le cas échéant et que nous voudrions utiliser ce format intermédiaire sur un autre robot nous devrions créer cette procédure, la procédure étant créée par Lucas Andrey pour son TPI[5] sur l'usine SmartFactory de l'HEIA-FR.

Dans la figure 8 nous pouvons voir un exemple d'un déplacement.

FIGURE 8 – Exemple du format intermédiaire

---

```
movePiece
[[8,40,8],[0.0,0.0,1.0,0.0],[-2,-1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
[[72,8,8],[0.0,0.0,1.0,0.0],[-2,-1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

---

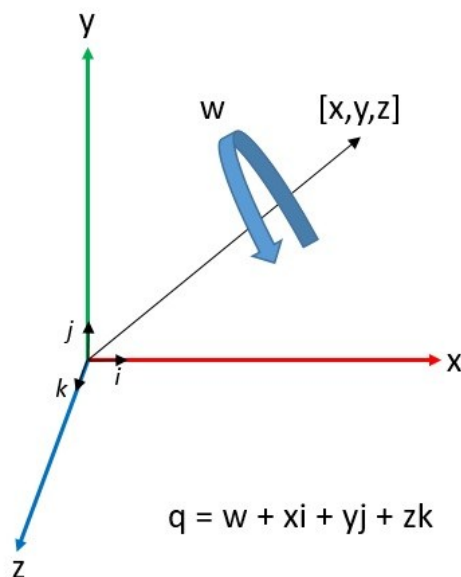
## 2.6 Quaternion

Un quaternion[6] est un système de nombre qui étend les nombres complexes. Ils sont aussi utilisés pour représenter des orientations/rotations en 3-dimensions et possède quelques avantages par rapport aux matrices de transformation :

- Nécessite seulement 4 scalaire au lieu de 9 pour une matrice
- Permet les opérations d'additions, soustractions et multiplications
- Empêche le blocage de cardan[7]

Un quaternion est caractérisé par 4 scalaires, 3 représentant un vecteur et le 4e un angle, une représentation est visible dans la figure 9.

FIGURE 9 – Représentation d'un quaternion



## 2.7 Platines

Les platines que nous utiliserons dans ce projet sont construit en mousse époxy et ont des tétons plus petits de  $0.2mm$  pour faciliter le décrochage des briques.

Les platines sont de taille  $8 \times 16$  tétons, nous devons donc déplacer l'origine de  $(-8, -4)$  pour que l'origine se situe dans le coin en bas à gauche de la platine.

Nous avons analysé les fichiers LXFML, le format intermédiaire que nous utilisons, vue rapidement les quaternions avec leurs avantages et les platines qui sont utilisées dans l'usine SmartFactory. Le prochain chapitre nous discuterons de l'implémentation des moyens nécessaires à la génération du fichier au format intermédiaire.

## 3 Implémentation

L'implémentation du programme de conversion est composé de plusieurs éléments. En premier nous allons voir le langage de programmation choisis pour le projet, suivi par les différentes parties de ce programme.

### 3.1 Langage de développement

Le langage de programmation choisi pour ce projet est le langage Python[8], Python à été choisi parmi deux langages : Python et Java.

Le raisonnement derrière le choix de Python fût en premier lieu motivé par la rapidité de développement, ceci puisque le Python est un langage interprété et non compilé comme le Java et possède aussi un interpréteur qui permet de prototyper rapidement des idées et méthodes pour le programme.

### 3.2 Fonctions principales

Le programme est composé de 3 fonctions principales :

- `convert` : Convertit une matrice de transformation affine 3-dimensionnel et la valeur du déplacement de l'origine dans une position du format intermédiaire
- `parse` : Parse un fichier LXFML et retourne une table associative des briques dans le fichier et leur positions
- `construct` : Construit le fichier du format intermédiaire à partir de 2 table associative qui proviennent de `parse`

#### 3.2.1 `convert`

Cette fonction convertit la matrice de transformation affine d'une brique dans la position du format intermédiaire, le code source est visible dans l'annexe A.1.1.

La plus grande difficulté fût d'abord de transformer la matrice de rotation se trouvant dans la matrice de transformation affine dans un quaternion représentant la rotation de la pièce. La première méthode était de convertir nous même la matrice de rotation dans un quaternion en utilisant un algorithme[9], visible dans la figure 10.

FIGURE 10 – Algorithme de conversion de matrice à quaternion

---

```

tr = matrix(0,0) + matrix(1,1) + matrix(2,2)

if tr > 0:
    s = math.sqrt(tr+1.0) * 2
    qw = 0.25 * s
    qx = (matrix(2,1) - matrix(1,2)) / s
    qy = (matrix(0,2) - matrix(2,0)) / s
    qz = (matrix(1,0) - matrix(0,1)) / s
elif (matrix(0,0) > matrix(1,1)) & (matrix(0,0) > matrix(2,2)):
    s = math.sqrt(1.0 + matrix(0,0) - matrix(1,1) - matrix(2,2)) * 2
    qw = (matrix(2,1) - matrix(1,2)) / s
    qx = 0.25 * s
    qy = (matrix(0,1) - matrix(1,0)) / s
    qz = (matrix(0,2) - matrix(2,0)) / s
elif matrix(1,1) > matrix(2,2):
    s = math.sqrt(1.0 + matrix(1,1) - matrix(0,0) - matrix(2,2)) * 2
    qw = (matrix(0,2) - matrix(2,0)) / s
    qx = (matrix(0,1) + matrix(1,0)) / s
    qy = 0.25 * s
    qz = (matrix(1,2) + matrix(2,1)) / s
else:
    s = math.sqrt(1.0 + matrix(2,2) - matrix(0,0) - matrix(1,1)) * 2
    qw = (matrix(1,0) - matrix(0,1)) / s
    qx = (matrix(0,2) + matrix(2,0)) / s
    qy = (matrix(1,2) + matrix(2,1)) / s
    qz = 0.25 * s

```

---

Après des tests nous avons découvert qu'il fallait encore multiplié ce quaternion par un quaternion représentant un vecteur dirigé vers le haut  $(0,0,1)$  avec une rotation nul. Pour nous faciliter les opérations avec les quaternions qui ne sont pas les plus évidents à comprendre, nous avons décidé d'utiliser une librairie Python pour les quaternions, `PyQuaternion`[10].

Cette librairie nous aident à convertir les matrices de rotation dans des quaternions et ensuite faire des opérations dessus.

La dernière étape est de changé l'échelle de la position et de changer l'origine, l'échelle nous l'avons découverte dans la section "Analyse/Taille de briques"[2.4]. Le code source pour la version finale de `convert` est visible dans l'annexe[A.1.2].

### 3.2.2 parse

Cette fonction lit et extraits les positions des briques dans les fichiers LXFML, l'extraction des données XML ce fait avec le module "`xml.etree.ElementTree`"[11] de la librairie standard de Python.

Nous cherchons d'abord tous les enfants de *Bricks*, ensuite le *designID* de chaque brique et nous finissons avec les valeurs se trouvant dans l'attribut *transformation*

de chaque brique. Nous stockons toute cette information dans une table associative, dictionnaire en Python, avec les *designID* comme clés et les valeurs des listes contenant les positions de chaque brique du type *designID*. Le code source de parse est visible dans l'annexe[A.1.3]

### 3.2.3 construct

Cette fonction construit et écrit le fichier du format intermédiaire en utilisant les autres fonctions au-dessus. Pour cela nous utilisons d'abord la fonction parse[3.2.2] sur les deux fichiers passé en paramètres, nous ouvrons ensuite un flux de sortie pour le fichier final et écrit l'en-tête.

Nous prenons ensuite les listes des positions et l'identifiant d'un type de brique et cherchons dans la table associative le même identifiant et nous prenons une des positions, nous convertissons, avec une fonction d'aide utilisant convert[3.2.1], les deux positions et les écrivons dans le fichier. Nous répétons ce processus jusqu'à ce que toutes les briques dans le deuxième fichiers soient traitées, la figure 11 illustre cet algorithme.

FIGURE 11 – Algorithme de recherche des briques

---

```
length = 2nd_file.length

while length > 0:
    brick = 2nd_file.popitem()
    brick_id = brick[0]
    bricks = brick[1]
    no_bricks = bricks.length

    for i in range(no_bricks):
        end_brick = bricks[i]
        start_brick = 1st_file.get(brick_id).pop()
        f.write(self.convert_all(start_brick, end_brick) + "\n")
    length -= 1
```

---

Le code source de cette fonction est visible dans l'annexe [A.1.4].

## 3.3 Programme complet

Nous avons décidé de regrouper toutes les fonctions mentionnées en-dessus dans une classe Python.

La classe est instancié en passant l'en-tête voulu pour le fichier et le nombres de tétons depuis le centre de la platine jusqu'au coin gauche du bas, dans le format  $[-X, -Y]$ .

Nous pouvons ensuite construire le fichier en appelant la méthode construct[3.2.3] avec les paramètres suivantes :

starting\_plate : le fichier LXFML de la platine de base



build : le fichier LXFML du modèle à construire

output : le nom du fichier du format intermédiaire

Le code source du programme est visible dans l'annexe [A.1.5].

Nous avons vu les différentes fonctions nécessaires à la conversion d'un fichier LXFML dans le format intermédiaire et le langage de développement utilisé pour ce projet. Le prochain chapitre conclura ce rapport en évoquant les prochaines étapes nécessaires pour avoir un système complet et des idées d'améliorations.

## 4 Conclusion

Nous avons donc trouvé que le projet est faisable et avons aussi un premier prototype qui fonctionne mais seulement pour des modèles d'une brique de haut et qui utilise des pièces prédéfinies.

Pour rendre ce projet plus utile et d'atteindre notre but pour ce projet nous devons accomplir quelques étapes supplémentaires et nous discuterons aussi d'idées d'amélioration possibles.

### 4.1 Prochaines étapes nécessaire

Nous pouvons envisager que les prochaines étapes soient de permettre de construire des modèles sur plusieurs niveaux, avec des briques non-horizontales et des platines de départ généré par notre programme.

#### 4.1.1 Construire sur plusieurs niveaux

Pour le moment nous pouvons seulement construire des modèles d'une brique de haut mais pour compléter ce projet nous devons pouvoir construire sur plusieurs niveaux.

Pour accomplir cela nous devons modifier le programme et trouvé des solutions pour ordonner les pièces dans l'ordre de l'axe  $Z$ . Une des solutions qui a déjà été envisagé est d'utiliser une variation sur l'algorithme du Peintre [2].

L'algorithme du Peintre[12] est comme suit :

FIGURE 12 – L'algorithme du Peintre

---

```
Repeat while not all objects placed {  
    Find object with the lowest value of Z  
    Place the object  
    Remove object from list  
}
```

---

Cet algorithme est une solution plutôt simple et nous devons le tester pour être sûr que celui-ci fonctionne pour nous, dans le cas échéant nous devons trouver une autre solution.

#### 4.1.2 Construire avec des briques non-horizontales

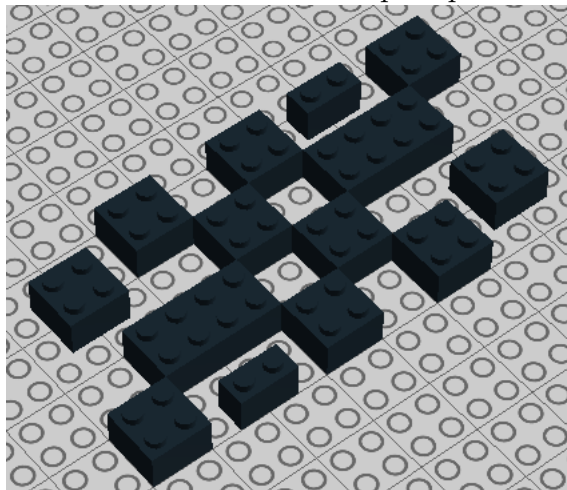
Nous permettons pour ce moment seulement des modèles avec des briques qui se posent horizontalement mais pour avoir un programme plus robuste nous devons aussi permettre des briques non-horizontales.

Pour accomplir cela nous devons passer plus de temps à utiliser l'usine Smart-Factory et le bras robot six axes pour trouver les limitations de ceux-ci. Une fois que les limitations découvertes nous devons trouver des solutions adéquates.

### 4.1.3 Généré automatiquement la platine de départ

Nous construisons tous nos modèles à partir d'une platine de départ prédéfinie, visible dans la figure 13, mais pour rendre le programme plus robuste et utile nous devrions aussi générer la platine en même temps que le format intermédiaire.

FIGURE 13 – Platine de départ prédéfinie



Pour accomplir cela nous devons trouver un algorithme de remplissage de la platine de départ et générer un fichier LXFML pour pouvoir visualiser la platine et la préparer pour l'usine.

## 4.2 Idées d'amélioration

Nous avons aussi pensé à quelques idées qui permettraient d'améliorer le projet. Quelques idées sont expliquées en-dessous.

### 4.2.1 Serveur de conversion

Une idée serait d'avoir un serveur de conversion où des personnes pourront télécharger leur modèles LEGO créés avec LDD et ensuite recevoir les fichiers nécessaires pour les mettre sur l'usine pour construire leurs modèles.

Une idée plus poussée serait que ce serveur soit directement relié à l'usine et que la construction se lancera automatiquement dès que la conversion soit achevée et que la personne puisse ensuite par la suite venir là chercher.

### 4.2.2 Utilisation de QR codes

Cette idée se repose sur l'idée d'un serveur de conversion, après la conversion achevée l'utilisateur reçoit un QR code qui leur permettrait ensuite d'aller à l'usine et faire scanner ce code pour faire construire leur modèle.

### 4.2.3 Modélisation en VR/AR

La dernière idée que nous avons est de pouvoir créer le modèle en réalité augmentée ou en réalité virtuelle et ensuite voir notre modèle construit en vrai dans l'usine.

Pour la modélisation en réalité augmentée, nous pourrions envisager d'utiliser un casque Microsoft HoloLens[12] ou un casque Meta[13], ces deux casques étant les meilleurs disponibles sur le marché en ce moment.

Pour la modélisation en réalité virtuelle, nous pourrions envisager d'utiliser un casque Oculus Rift[14] ou un casque HTC Vive[15], ces deux casques sont déjà très utilisés par le grand public pour jouer à des jeux vidéo.

## 4.3 Avis personnel

Ce projet fût très intéressant et éducative, j'ai appris des nouvelles notions et en même temps pu utiliser des connaissances acquises dans d'autres cours.

Je serais aussi très intéressé de poursuivre plus ce projet et voir jusqu'où nous pourrions aller

## Déclaration d'honneur

Je, soussigné, Owen McGill, déclare sur l'honneur que le travail rendu est le fruit d'un travail personnel. Je certifie ne pas avoir eu recours au plagiat ou à toutes autres formes de fraudes. Toutes les sources d'information utilisées et les citations d'auteur ont été clairement mentionnées.

## Table des figures

1	Extrait d'un fichier LXFML avec ordre de placement des briques . .	6
2	Image du modèle avec ordre de placement des briques . . . . .	6
3	Brique avec origine et représentation d'un quaternion . . . . .	8
4	Image d'une brique LEGO $2 \times 2$ [4] . . . . .	8
5	Extrait d'un fichier LXFML pour trouver l'échelle des pièces . . . .	9
6	Image du modèle utilisé pour trouver l'échelle . . . . .	9
7	Syntaxe du format intermédiaire . . . . .	10
8	Exemple du format intermédiaire . . . . .	11
9	Représentation d'un quaternion . . . . .	11
10	Algorithme de conversion de matrice à quaternion . . . . .	14
11	Algorithme de recherche des briques . . . . .	15
12	L'algorithme du Peintre . . . . .	17
13	Platine de départ prédéfinie . . . . .	18

## Liste des tableaux

1	Index de la liste de transformation sous forme de matrice de transformation affine . . . . .	7
---	--	---

## Références

- [1] FESTO. *CP Factory l'usine Cyber-physique*. URL : <http://www.festo-didactic.com/int-fr/learning-systems/usine-ecole,systemes-cim-fms/cp-factory/cp-factory-1-usine-cyber-physique.htm?fbid=aW50LmZyLjU1Ny4xNi4xOC4xMjkzLjUzNzY4> (visité le 26/06/2017).
- [2] WIKIPEDIA. *Algorithme du peintre*. URL : [https://fr.wikipedia.org/wiki/Algorithme\\_du\\_peintre](https://fr.wikipedia.org/wiki/Algorithme_du_peintre) (visité le 21/06/2017).
- [3] FORUM. *Understanding LDD's LXFML Schema*. URL : <http://www.eurobricks.com/forum/index.php?/forums/topic/92863-understanding-ldds-lxfml-schema/> (visité le 21/06/2017).
- [4] *Lego 3003 White 2x2 Brick*. URL : <http://ronsrescuedtreasures.com/Lego-3003-White-2x2-Brick-P2621725.aspx> (visité le 26/06/2017).
- [5] Lucas ANDREY. *Programmation d'un bras robot six axes*. Rapp. tech. Ecole des Métiers Fribourg, 2016.
- [6] WIKIPEDIA. *Quaternion*. URL : <https://en.wikipedia.org/wiki/Quaternion> (visité le 21/06/2017).
- [7] WIKIPEDIA. *Blocage de cardan*. URL : [https://fr.wikipedia.org/wiki/Blocage\\_de\\_cardan](https://fr.wikipedia.org/wiki/Blocage_de_cardan) (visité le 26/06/2017).
- [8] PYTHON. *Python programming language*. URL : <https://www.python.org/> (visité le 26/06/2017).

- [9] Martin John BAKER. *Maths - Conversion Matrix to Quaternion*. URL : <http://www.euclideanspace.com/maths/geometry/rotations/conversions/matrixToQuaternion/> (visité le 26/06/2017).
- [10] Kieran WYNN. *PyQuaternion*. URL : <https://kieranwynn.github.io/pyquaternion/> (visité le 21/06/2017).
- [11] PYTHON. *The ElementTree XML API*. URL : <https://docs.python.org/3/library/xml.etree.elementtree.html> (visité le 21/06/2017).
- [12] MICROSOFT. *Microsoft Hololens*. URL : <https://www.microsoft.com/en-us/hololens> (visité le 26/06/2017).
- [13] META. *Meta 2*. URL : <https://www.metavision.com/> (visité le 26/06/2017).
- [14] OCULUS. *Oculus Rift*. URL : <https://www.oculus.com/rift/> (visité le 26/06/2017).
- [15] HTC. *HTC Vive*. URL : <https://www.vive.com/eu/> (visité le 26/06/2017).

## Acronymes

**LDD** LEGO Digital Designer. 3, 5, 7–9, 18, 22, *Glossaire* : LEGO Digital Designer

**LXF** LEGO Exchange Format. 5, 22, *Glossaire* : LEGO Exchange Format

**LXFML** LEGO Exchange Format Markup Language. 5–9, 12–16, 18, 21, 22, *Glossaire* : LEGO Exchange Format Markup Language

**TPI** Travail Pratique Individuel. 10, 11, 22, *Glossaire* : Travail Pratique Individuel

## Glossaire

**Java** Java est un langage de programmation compilé, créé par Sun Microsystems. 22

**LEGO Digital Designer** Application mis a disposition par le groupe LEGO pour créer des modèles LEGO. 3, 5, 22

**LEGO Exchange Format** Fichier comportant une image et un fichier LXFML généré par LDD. 5, 22

**LEGO Exchange Format Markup Language** Fichier XML généré par LDD. 5, 22

**Python** Python est un langage de programmation interprété, créé par Guido van Rossum [8]. 13–15, 22

**Quaternion** Un quaternion est un système de nombre pour représenter une orientation/rotation dans l'espace d'un objet. 22

**SmartFactory** Usine Cyber-Physique construite par Festo. 3, 11, 12, 22

**Travail Pratique Individuel** Rapport technique de l'Ecole des Métiers FribourgTPI. 10, 22



## A Annexes

### A.1 Code source

#### A.1.1 convert(matrix, origin)

---

```

def convert(self, matrix, origin):
    x, y, z = matrix[11], matrix[9], matrix[10]
    tr = matrix[0] + matrix[4] + matrix[8]
    origin_x, origin_y = origin[0], origin[1]

    if tr > 0:
        s = math.sqrt(tr+1.0) * 2
        qw = 0.25 * s
        qx = (matrix[5] - matrix[7]) / s
        qy = (matrix[6] - matrix[2]) / s
        qz = (matrix[1] - matrix[3]) / s
    elif (matrix[0] > matrix[4]) & (matrix[0] > matrix[8]):
        s = math.sqrt(1.0 + matrix[0] - matrix[4] - matrix[8]) * 2
        qw = (matrix[5] - matrix[7]) / s
        qx = 0.25 * s
        qy = (matrix[3] - matrix[1]) / s
        qz = (matrix[6] - matrix[2]) / s
    elif matrix[4] > matrix[8]:
        s = math.sqrt(1.0 + matrix[4] - matrix[0] - matrix[8]) * 2
        qw = (matrix[6] - matrix[2]) / s
        qx = (matrix[3] + matrix[1]) / s
        qy = 0.25 * s
        qz = (matrix[7] + matrix[5]) / s
    else:
        s = math.sqrt(1.0 + matrix[8] - matrix[0] - matrix[4]) * 2
        qw = (matrix[1] - matrix[3]) / s
        qx = (matrix[6] + matrix[2]) / s
        qy = (matrix[7] + matrix[5]) / s
        qz = 0.25 * s

    new_x = round(round(x, 1) * self.scaling - origin_x - self.offset_x)
    new_y = round(round(y, 1) * self.scaling - origin_y - self.offset_y)
    new_z = round(round(z, 1) * self.scaling + self.offset_z)

    return str(f"[[{new_x},{new_y},{new_z}],
[{qx},{qy},{qz},{qw}],
{self.confData},{self.extJoints}]")

```

---

#### A.1.2 convertV2(matrix, origin)

---

```

def convert(self, matrix, origin):
    origin_x, origin_y = origin[0], origin[1]

```

```

x, y, z = matrix[11], matrix[9], matrix[10]

rot = array([(matrix[0], matrix[3], matrix[6]),
             (matrix[1], matrix[4], matrix[7]),
             (matrix[2], matrix[5], matrix[8])])

q = Quaternion(matrix=rot)
quat = Quaternion(0, 0, 0, 1)
q = q * quat
q = q.elements

new_x = round(round(x, 1) * self.scaling - origin_x - self.offset_x)
new_y = round(round(y, 1) * self.scaling - origin_y - self.offset_y)
new_z = round(round(z, 1) * self.scaling + self.offset_z)

return str(f"[{new_x},{new_y},{new_z}],
           [{q[1]},{q[2]},{q[3]},{q[0]}],
           {self.confData},{self.extJoints}]")

```

---

### A.1.3 parse(file)

---

```

def parse(file):
    tree = xml.etree.ElementTree.parse(file)
    root = tree.getroot()
    bricks = {}
    for child in root.find('Bricks'):
        for part in child:
            part_id = part.attrib.get('designID')
            for bone in part:
                matrix = bone.attrib.get('transformation').split(',')
                matrix = list(map(float, matrix))
                if part_id not in bricks:
                    matrices = [matrix]
                else:
                    matrices = bricks[part_id]
                    matrices.append(matrix)
            bricks[part_id] = matrices

```

---

### A.1.4 construct(starting\_plate, build, output)

---

```

def construct(self, starting_plate, build, output):
    start = self.parse(starting_plate)
    end = self.parse(build)

    f = open(output, 'w')

```

```

f.write(self.header + "\n")

length = len(end)

while length > 0:
    brick_list = end.popitem()
    brick_id = brick_list[0]
    bricks = brick_list[1]
    no_bricks = len(bricks)

    for i in range(no_bricks):
        end_brick = bricks[i]
        start_brick = start.get(brick_id).pop()
        f.write(self.convert_all(start_brick, end_brick) + "\n")
    length -= 1
f.close()

```

---

### A.1.5 Classe Lego

---

```

import xml.etree.ElementTree
from numpy import array
from pyquaternion import Quaternion

class Lego:

    extJoints = "[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]"
    confData = "[-2,-1,1,0]"
    scaling = 10
    offset_x = 12
    offset_y = 4
    offset_z = 8
    translation_scaling = 8

    def __init__(self, header, translation):
        self.header = header
        self.translation = translation

    def convert_all(self, start, end):
        start = self.convert(start, self.translation)
        end = self.convert(end, self.translation)
        return str(f"movePiece {start},{end};")

    def convert(self, matrix, origin):
        origin_x, origin_y = origin[0], origin[1]
        x, y, z = matrix[11], matrix[9], matrix[10]
        rot = array([(matrix[0], matrix[3], matrix[6]),
                     (matrix[1], matrix[4], matrix[7]),

```

---

```

        (matrix[2], matrix[5], matrix[8]))
    quat_rotation = Quaternion(matrix=rot)
    quat_orientation = Quaternion(0, 0, 0, 1)

    quat_result = quat_rotation * quat_orientation

    quat_elements = quat_result.elements

    new_x = round(round(x, 1) * self.scaling - origin_x *
                  self.translation_scaling - self.offset_x)
    new_y = round(round(y, 1) * self.scaling - origin_y *
                  self.translation_scaling - self.offset_y)
    new_z = round(round(z, 1) * self.scaling + self.offset_z)

    return str(f"[[{new_x},{new_y},{new_z}], "
              f"[{quat_elements[1]},{quat_elements[2]},{quat_elements[3]},{quat_elements[0]}], "
              f"{self.confData},{self.extJoints}]")

    @staticmethod
    def parse(file):
        tree = xml.etree.ElementTree.parse(file)
        root = tree.getroot()
        bricks = {}
        for child in root.find('Bricks'):
            for part in child:
                part_id = part.attrib.get('designID')
                for bone in part:
                    matrix = bone.attrib.get('transformation').split(',')
                    matrix = list(map(float, matrix))
                    if part_id not in bricks:
                        matrices = [matrix]
                    else:
                        matrices = bricks[part_id]
                        matrices.append(matrix)
                    bricks[part_id] = matrices

        return bricks

    def construct(self, starting_plate, build, output):
        start = self.parse(starting_plate)
        end = self.parse(build)

        f = open(output, 'w')

        f.write(self.header + "\n")

        length = len(end)

```

```
while length > 0:
    brick = end.popitem()
    brick_id = brick[0]
    bricks = brick[1]
    no_bricks = len(bricks)

    for i in range(no_bricks):
        end_brick = bricks[i]
        start_brick = start.get(brick_id).pop()
        f.write(self.convert_all(start_brick, end_brick) + "\n")
    length -= 1
f.close()
```

---

## A.2 Versions des logiciels/librairie

Python : Version 3.6.1  
numpy : Version 1.13.0  
pip : Version 9.0.1  
pyquaternion : Version 0.9.0  
setuptools : Version 18.0.1  
wheel : Version 0.24.0

## A.3 Manuel d'utilisation

Pour utiliser ce programme il faut avoir Python version 3.6 installé sur la machine.

### A.3.1 Installation des librairies

Vérifier si *pip* est installé sur votre système, en exécutant

---

```
$ pip3 --version
```

---

Si la commande vous retourne des erreurs, installer *pip* comme indiqué sur le site <https://pip.pypa.io/en/stable/installing/>.

Une fois que vous avez *pip* installé, vous pourrez exécuter

---

```
$ pip3 install pyquaternion
```

---

Après l'installation de PyQuaternion, vous aurez toutes les dépendances du programme.

### A.3.2 Exécution du programme

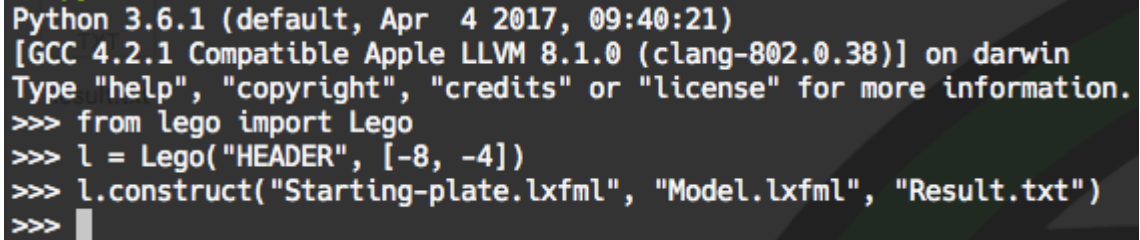
Maintenant pour utiliser le programme il faut lancer Python version 3.6 avec les dépendances.

---

```
$ python3
```

---

Un exemple de l'exécution du programme se trouve en-dessous.



```
Python 3.6.1 (default, Apr  4 2017, 09:40:21)
[GCC 4.2.1 Compatible Apple LLVM 8.1.0 (clang-802.0.38)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from lego import Lego
>>> l = Lego("HEADER", [-8, -4])
>>> l.construct("Starting-plate.lxfml", "Model.lxfml", "Result.txt")
>>> █
```

Les étapes sont les suivantes :

- Importer la classe Lego
- Instancier la classe avec l'en-tête et la translation depuis le centre vers le coin gauche du bas
- Exécuter la méthode construct sur l'instance de la classe avec les fichiers nécessaires
  - Le fichier LXFML de la platine de départ
  - Le fichier LXFML du modèle
  - Le nom du fichier sortant

## B Structure de la clé usb

La structure de la clé USB est la suivante :

- Code : Dossier contenant le code source du programme
  - lego.py
- Modeles : Dossier contenant les modèles utilisé durant le projet
  - Brick ordering : Dossier contenant les modèles utilisé pour identifier l'ordre des briques
  - Brick positioning : Dossier contenant les modèles utilisé pour identifier le positionnement et l'origine
  - Coordinate system : Dossier contenant les modèles pour identifier le système de coordonnées
  - Report : Dossier contenant les modèles pour la rédaction du rapport
  - Test cases : Dossier contenant les modèles des test cases
- Ordre du jour : Dossier contenant les Ordre du jours
  - Ordre du jour-30.05.17.pdf
  - Ordre du jour-06.06.17.pdf
  - Ordre du jour-13.06.17.pdf
  - Ordre du jour-20.06.17.pdf
- PV : Dossier contenant les PV
  - PV-19.05.17.pdf
  - PV-30.05.17.pdf
  - PV-06.06.17.pdf
  - PV-13.06.17.pdf
  - PV-20.06.17.pdf
- Rapport : Dossier contenant tous les fichiers du rapport
  - images : Dossier contenant les images du rapport
  - tex : Dossier contenant les sous-fichiers LaTeX du rapport
    - glossary.tex
    - lego.py
    - Report.bib
    - Report.tex
    - Readme.tex
    - Manuel d'utilisation.tex
- Planning.pdf
- Rapport.pdf
- Manuel d'utilisation.tex