



Ecole Polytechnique de l'Université de Tours

Département Informatique

64 avenue Jean Portalis

37200 Tours, France

Tél. +33 (0)2 47 36 14 14

polytech.univ-tours.fr

Projet Recherche & Développement

2021-2022

Apprentissage de la préhension d'objets à l'aide d'un bras manipulateur



POLYTECH[®]
TOURS

Tuteurs académiques

Pierre GAUCHER

Nicolas MONMARCHE

Étudiant

Hanlu HU (DI5)

8 avril 2022



Liste des intervenants

Nom	Email	Qualité
Hanlu HU	hanlu.hu@etu.univ-tours.fr	Étudiant DI5
Pierre GAUCHER	pierre.gaucher@univ-tours.fr	Tuteur académique, Département Informatique
Nicolas MONMARCHE	nicolas.monmarche@univ-tours.fr	Tuteur académique, Département Informatique



Avertissement

Ce document a été rédigé par Hanlu HU susnommé l'auteur.

L'Ecole Polytechnique de l'Université de Tours est représentée par Pierre GAUCHER et Nicolas MONMARCHE susnommés les tuteurs académiques.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

L'auteur reconnaît assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

L'auteur atteste que les propos du document sont sincères et assume l'entière responsabilité de la véracité des propos.

L'auteur atteste ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

L'auteur atteste que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

L'auteur reconnaît qu'il ne peut diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable des tuteurs académiques et de l'entreprise.

L'auteur autorise l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.



Pour citer ce document

Hanlu HU, *Apprentissage de la préhension d'objets à l'aide d'un bras manipulateur*, Projet Recherche & Développement, Ecole Polytechnique de l'Université de Tours, Tours, France, 2021-2022.

```
@mastersthesis{
  author={HU, Hanlu},
  title={Apprentissage de la préhension d'objets à l'aide d'un bras manipulateur},
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université de Tours},
  address={Tours, France},
  year={2021-2022}
}
```

Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Table des figures	vi
1 Introduction	1
1 Enjeux et contexte.....	1
2 Objectifs.....	2
3 Hypothèses	2
4 Bases méthodologiques.....	2
2 Description générale	4
1 Environnement du projet	4
2 Caractéristiques des utilisateurs.....	4
3 Fonctionnalités du système	4
4 Structure générale du système.....	6
3 État de l'art	7
1 Modélisation de contacts, positions, forces et vitesses.....	7
Contacts	7
Positions	7
Forces et vitesses.....	8
2 Relation entre forces et vitesses.....	8

3	Evaluation de préhension	9
3.1	Métriques sur la matrice de préhension G	10
	Minimum des valeurs singulières de G	10
	Volume de l'ellipsoïde dans l'espace de moment	10
	L'indice d'isotropie de préhension	10
3.2	Métriques sur la relation géométrique des points de contact	10
	Distance entre le centre des points de contact et le centre de masse de l'objet	11
4	Analyse et conception	12
1	Analyse	12
2	Modélisation mathématique	12
2.1	Modélisation des contacts	12
2.2	Analyse de la préhension.....	13
2.2.1	Matrice de préhension	13
	Matrice de préhension partielle.....	14
	Matrice de préhension complète.....	14
3	Conception logicielle	14
3.1	Classe.....	14
	Classe Error	14
	Classe InputFormatError.....	15
	Classe LoadFile	15
	Classe Grasp	15
3.2	Interface	15
5	Mise en oeuvre	16
1	Outils et librairie utilisés.....	16
1.1	Numpy	16
1.2	HtmlTestRunner	16
1.3	Sphinx.....	16
2	IHM	17
3	Lecture de données	19
4	Calcul matriciel	20
4.1	Constructeur	20
4.2	Calculer le vecteur n	21
4.3	Calculer le vecteur t et o	21
4.4	Calculer la matrice de rotation.....	23
4.5	Calculer la matrice de préhension et ses valeurs singulières.....	24
4.6	Calculer les métriques.....	25
5	Analyse des résultats	26

5.1	Données d'entrée	26
5.2	Récupérer les résultats.....	27
5.3	Analyse	28
6	Bilan et conclusion	29
1	Bilan du semestre 9	29
2	Bilan du semestre 10.....	29
3	Bilan sur la qualité	30
4	Bilan auto-critique.....	30
	Annexes	31
A	Planification, gestion de projet	32
1	Evolution du projet	32
2	Description des tâches.....	33
	Tâche 1 : Analyser des besoins	33
	Tâche 2 : Rechercher des solutions existantes.....	33
	Tâche 3 : Vérifier la faisabilité	33
	Tâche 4 : Spécifier les besoins.....	33
	Tâche 5 : Rédaction des spécifications générales.....	34
	Tâche 6 : Rédaction des spécifications sur le modèle de préhension ..	34
	Tâche 7 : Rédaction des spécifications sur les métriques d'évaluation de préhenison.....	34
	Tâche 8 : Préparation de soutenance	34
	Tâche 9 : Soutenance S9.....	34
	Tâche 10 : Conception du modèle de préhension	34
	Tâche 11 : Mise en place de l'algorithme d'importation des données	34
	Tâche 12 : Mise en place de l'algorithme de calcul de la matrice de préhension	34
	Tâche 13 : Mise en place de l'algorithme de calcul des métriques	35
	Tâche 14 : Tester les algorithmes et les commenter	35
	Tâche 15 : Rédaction de rapport	35
	Tâche 16 : Soutenance S10	35
	Tâche 17 : Rendu des livrables finaux.....	35
B	Description des interfaces	36
1	Interfaces matérielles/logicielles	36
2	Interfaces homme/machine.....	36

C	Cahier de Spécifications	38
1	spécifications Fonctionnelles.....	38
	Fonctionnalités à développer.....	38
1.1	Définition de la fonction 1 : lire le fichier de données.....	38
	Description de la fonction 1 :.....	38
1.2	Définition de la fonction 2 :stocker une préhension.....	39
	Description de la fonction 2 :.....	39
1.3	Définition de la fonction 3 :calculer le vecteur n des deux points de contact.....	39
	Description de la fonction 3 :.....	39
1.4	Définition de la fonction 4 :calculer les valeurs t et o à partir d'un point et le vecteur n	39
	Description de la fonction 4 :.....	39
1.5	Définition de la fonction 5 :calculer la matrice de rotation.....	40
	Description de la fonction 5 :.....	40
1.6	Définition de la fonction 6 :calculer la matrice S	40
	Description de la fonction 6 :.....	40
1.7	Définition de la fonction 7 :calculer la matrice de préhension.....	40
	Description de la fonction 7 :.....	40
1.8	Définition de la fonction 8 :calculer les valeurs singulières de la matrice de préhension.....	41
	Description de la fonction 8 :.....	41
1.9	Définition de la fonction 9 :calculer Q_{MVS}	41
	Description de la fonction 9 :.....	41
1.10	Définition de la fonction 10 :calculer Q_{VEM}	41
	Description de la fonction 10 :.....	41
1.11	Définition de la fonction 11 :calculer Q_{IIP}	42
	Description de la fonction 11 :.....	42
1.12	Définition de la fonction 12 :calculer Q_{DCC}	42
	Description de la fonction 12 :.....	42
2	Spécifications non fonctionnelles.....	42
2.1	Contraintes de développement et conception.....	42
2.2	Contraintes de fonctionnement et d'exploitation.....	42
2.2.1	Performances.....	42
2.2.2	Capacités.....	43
2.2.3	Contrôlabilité.....	43
2.2.4	Sécurité.....	43
D	Guide d'installation des librairies	44
E	Guide d'utilisation	45

F	Cahier du développeur	46
1	Diagramme de classe.....	46
	Classe Error	46
	Classe InputFormatError.....	46
	Classe LoadFile	46
	Classe Grasp	46
2	Structure des fichiers utilisés.....	46
G	Cahier de test	48
1	Test avec unittest	48
2	Tests unitaires de la classe LoadFile.....	48
3	Tests unitaires de la classe Grasp.....	49
H	Bibliographie	50



Table des figures

1	Introduction	
1.1	Modèles de contact : a. Contact ponctuel sans friction b. Contact ponctuel avec frottement c. Contact mou.....	3
2	Description générale	
2.1	Diagramme de cas d'utilisation	5
2.2	Diagramme de séquence	5
3	État de l'art	
3.1	Relation entre la force de préhension et la vitesse.....	8
4	Analyse et conception	
4.1	Modélisation des contacts	13
4.2	Diagramme de classe.....	15
5	Mise en oeuvre	
5.1	Documentation	17
5.2	Données d'entrée	26
5.3	Les préhensions.....	27
5.4	Résultats	27
6	Bilan et conclusion	
6.1	Les tâches sur Trello	30

A Planification, gestion de projet	
A.1 Le diagramme de Gantt initial	32
A.2 Le diagramme de Gantt final	33
B Description des interfaces	
B.1 Interface du logiciel.....	37
E Guide d'utilisation	
E.1 Interface du logiciel.....	45
F Cahier du développeur	
F.1 Diagramme de classe.....	47
F.2 La structure du projet	47
G Cahier de test	
G.1 tests unitaires	48
G.2 rapport de tests	49

1

Introduction

Le but de ce document est de spécifier le Projet Recherche & Développement sur « Apprentissage de la préhension d'objets à l'aide d'un bras manipulateur ».

Le sujet a été proposé par deux enseignant chercheurs de l'équipe Reconnaissances des Formes & Analyse d'Images de Laboratoire Informatique de L'École Polytechnique de l'Université de Tours : M. Pierre GAUCHER et M. Nicolas MONMARCHE. Par conséquent, ils représentent le rôle MOA au sein de la réalisation de ce projet. Le rôle MOE est représenté par Hanlu HU, étudiant DI5 de Polytech Tours.

1 Enjeux et contexte

Nous souhaitons que les robots intelligents soient capables de détecter et d'interagir avec l'environnement dans le futur. Parmi toutes ses fonctionnalités, la préhension est la plus basique et la plus importante, parce qu'elle pourrait être très utile dans de nombreux scénarios.

Par exemple, les bras manipulateurs peuvent aider les travailleurs à effectuer des tâches laborieuses de sélection et de déplacement, et les robots domestiques peuvent aider les personnes handicapées ou âgées dans leur vie quotidienne. Par conséquent, l'étude de préhension des robots est à long terme un objectif dans le domaine de la vision par ordinateur et de la robotique.

Le processus de préhension d'un bras manipulateur est très complexe. En général, nous obtenons des informations sur la position et l'orientation de l'objet à saisir à partir d'une image, puis avec des traitements d'image nous essayons d'obtenir un ensemble de saisies réalisables. À l'aide du modèle dynamique du bras manipulateur, nous faisons en sorte que la pince se déplace vers la posture que nous calculons préalablement.

Dans différents contextes industriels, les bras manipulateurs ont des fonctions et des exigences différentes. C'est la raison pour laquelle nous avons une grande variété de modèles de bras manipulateurs et de pinces aussi. Pour ces différents types de bras et de pinces, les chercheurs ont également développé différents algorithmes. La clé du problème de préhension est donc de savoir comment :

1. analyser l'image afin d'obtenir un ensemble de points de contact ;
2. définir la meilleure préhension ;
3. manipuler le bras manipulateur.

Dans la plupart des études existantes, les chercheurs ont pu obtenir des postures réalisables par traitement d'images, puis vérifier la faisabilité de leurs algorithmes par un grand nombre d'essais de préhension. Cependant, les préhensions obtenues par ces algorithmes pourraient ne pas être réalisables en pratique en raison de la présence d'obstacles ou des performances insuffisantes du bras manipulateur.

Au contraire, nous pouvons chercher une méthode pour évaluer toutes les préhensions atteignables pour le bras manipulateur, et ainsi trouver la meilleure dans un ensemble limité de points de contact.

Dans ce Projet Recherche & Développement, nous allons étudier l'évaluation de préhensions sur le bras manipulateur à deux doigts. Dans notre cas, nous considérons qu'un ensemble de préhensions atteignables pour le manipulateur est fourni. A partir des points de contact, nous pourrions éventuellement définir la meilleur préhension.

2 Objectifs

L'objectif principal du travail décrit dans ce document est de répondre aux questions relatives à la sélection des points de contact. Dans une préhension, comment construire un modèle approprié pour évaluer la qualité et comment sélectionner des bonnes métriques de qualité à partir du modèle. Parmi un ensemble de préhensions atteignables par le bras manipulateur comment trouver la meilleur préhension ou l'ensemble de points de contact.

Tout d'abord, nous allons modéliser mathématiquement le processus de préhension. Le modèle représentera la relation entre le contact, la position, la force et la vitesse au cours du processus de préhension dans une matrice mathématique. C'est un modèle bien développé dans le domaine robotique. La compréhension nécessite une bonne base de l'analyse robotique et ainsi l'algèbre linéaire.

Ensuite, nous étudions sur des métriques d'évaluation de préhensions. La manière d'évaluer les différentes préhensions sur les points de contact avec les bonnes métriques est le but final du projet. Les métriques choisies doivent permettre de visualiser l'écart entre les différents crawls au niveau du modèle. Idéalement, à partir de ces métriques, nous pouvons éventuellement donner un score à chaque préhension ou l'ensemble de points de contact.

3 Hypothèses

Au cours du projet, nous supposons que les points de contact entre la pince et l'objet à saisir soient identiques et que il n'y ait pas de glissement pendant la préhension. Par conséquent, pour chaque préhension, nous le représentons par un ensemble de points de contact.

De plus, dans la modélisation de la préhension, nous considérons que pour une pince à deux doigts, les forces appliquées à chaque point de contact sont dans des directions opposées.

Une illustration plus détaillée sera développée dans les chapitres suivants.

4 Bases méthodologiques

La compréhension du contact entre le bras manipulateur et l'objet est essentielle à l'analyse de la préhension. Lorsque deux objets entrent en contact, chacun d'entre eux peut exercer une force au point de contact de l'une des trois manières suivantes [1] :

1. Contact ponctuel sans friction : la force appliquée est toujours normale à la frontière de contact ;

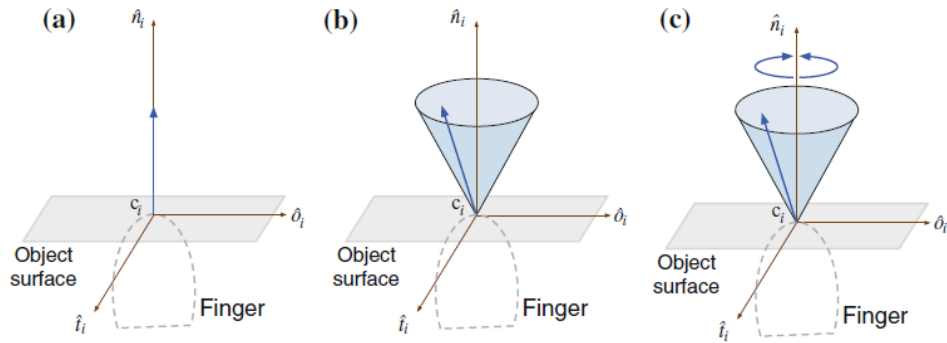


Figure 1.1 – *Modèles de contact : a. Contact ponctuel sans friction b. Contact ponctuel avec frottement c. Contact mou*

2. Contact ponctuel avec frottement : la force appliquée a une composante normale à la frontière de contact et peut en avoir une autre tangentielle à celle-ci. Plusieurs modèles ont été proposés pour représenter la friction, le plus courant étant la friction de Coulomb ;
3. Contact mou : il permet d'appliquer les mêmes forces que le contact ponctuel avec frottement, plus un moment autour de la direction normale à la frontière du contact. Ce modèle n'est valable que pour modéliser le contact entre un doigt souple et un objet rigide.

Pour une pince à deux doigts, nous considérons que la force exercée par le doigt est de type Contact ponctuel avec frottement. C'est le modèle que nous implémentons au cours du projet.

2

Description générale

1 Environnement du projet

Pendant la mise en œuvre du projet, les préhensions seront fournies sous forme des ensembles de points de contact. Au cours du projet, nous effectuerons des calculs matriciels complexes. Pour faciliter le développement du programme, nous utiliserons Python comme langage de programmation. Le programme sera développé dans un environnement Windows ou Linux.

2 Caractéristiques des utilisateurs

Afin d'évaluer plusieurs préhensions à l'aide des métriques de préhension choisies dans ce projet, l'utilisateur doit avoir une bonne connaissance des domaines de l'informatique et ainsi de la robotique. Lorsque le système est utilisé dans une situation pratique, l'utilisateur devrait pouvoir générer plusieurs ensembles de points de contact réalisables à partir des images et, après avoir utilisé le système d'évaluation développé dans ce projet, être capable d'effectuer des actions de préhension suivantes.

3 Fonctionnalités du système

Les préhensions seront fournies sous forme des ensembles de points de contact dans un fichier .txt. Par conséquent, le système prend un fichier contenant des ensembles de points de contact comme l'entrée, il fait des calculs matriciels. Eventuellement, le système nous donne la matrice de préhension. Selon les métriques que nous choisissons, le système calcule les valeurs singulières de la matrice de préhension et nous donne les valeurs des métriques. Le diagramme de cas d'utilisation est comme Figure 2.1. Etant un système simple, le diagramme de séquence est comme Figure 2.2.

Les différentes fonctionnalités du système :

- Prendre en entrée les préhensions et le centre de masse de l'objet
- Modéliser la préhension sur chaque point de contact
- Calculer la matrice de préhension partiel
- Calculer la matrice de préhension complète

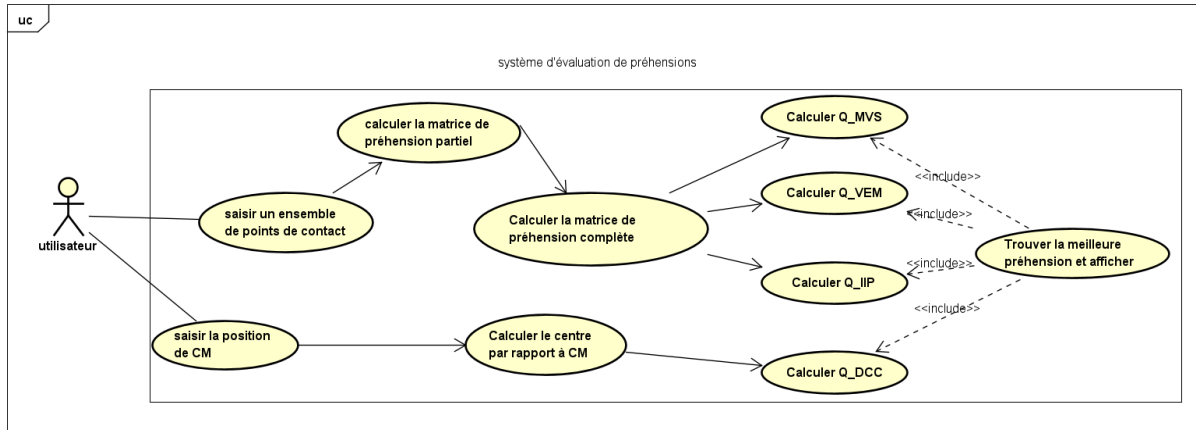


Figure 2.1 – Diagramme de cas d'utilisation

- Calculer les valeurs singulières de la matrice de préhension
- Calculer la métrique d'évaluation Q_{MVS}
- Calculer la métrique d'évaluation Q_{VEM}
- Calculer la métrique d'évaluation Q_{IIP}
- Calculer la métrique d'évaluation Q_{DCC}
- Trouver la meilleure préhension et afficher

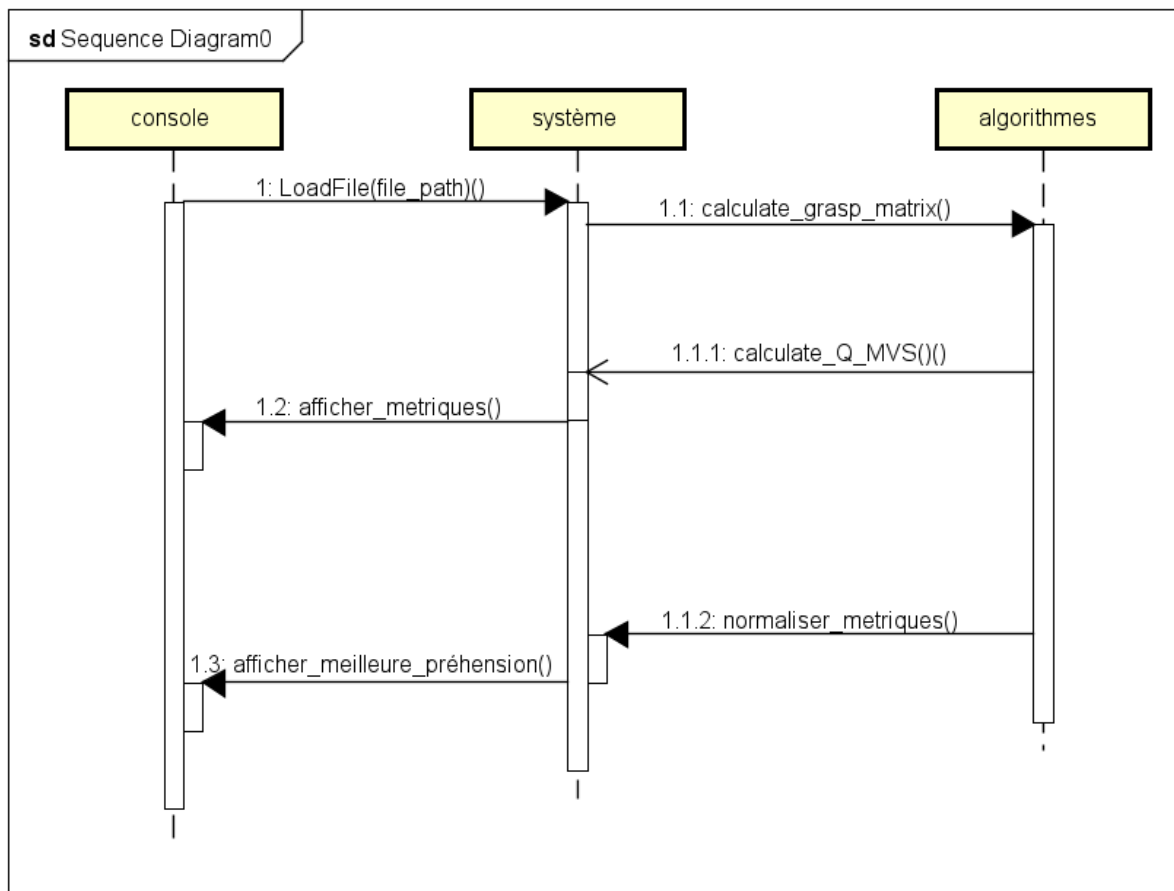


Figure 2.2 – Diagramme de séquence

4 Structure générale du système

Le système sera composé de plusieurs éléments pour répondre à toutes les fonctionnalités demandées :

1. Pour lire les préhensions et la position du centre de masse, nous les stockerons dans un fichier texte. L'utilisateur saisira dans le console l'emplacement du fichier.
Classe LoadFile → une class pour lire le fichier texte et transformer le contenu en structure de contact de point.
2. Pour garantir le format du fichier d'entrée, nous vérifions que le fichier a le format requis lors de la lecture des données.
Classe InputFormatError → une class pour définir l'exception à lever lors que le format du fichier d'entrée n'est pas le format standard.
3. Pour évaluer les préhension, nous allons calculer la matrice de préhension qui vient des matrices de préhension partielles.
Classe Grasp → une class pour tous les calculs matriciels sur la préhension.
4. Après avoir la matrice de préhension, nous allons calculer les métriques choisies, y compris les calculs des valeurs singulières d'une matrices et de la distance entre le centre des points de contact et le centre de masse de l'objet.
Classe Grasp → pour chaque object de type Grasp, après avoir la matrice de préhension, nous pouvons faire les calculs des métriques d'évaluations de préhensions.
5. Afin de faciliter la réception des entrées de l'utilisateur et la sortie du calcul, nous concevons l'interface interactive de la console dans le fichier main.py.
Fichier main.py → une interface pour recevoir le fichier et afficher les résultats.

3

État de l'art

La préhension et la manipulation à l'aide de pinces complexes, telles que les mains à plusieurs doigts, constituent un domaine de recherche actif en robotique. Le but ultime de la préhension est d'effectuer le mouvement des objets en présence de perturbations externes, y compris le poids propre de l'objet. L'analyse de préhension de robot est étroitement liée à la conception des pinces et à la conception du robot. D'abord, nous allons modéliser une préhension généralement.

1 Modélisation de contacts, positions, forces et vitesses

Nous pouvons modéliser la relation entre le contact, la position, la force et la vitesse au cours du processus de préhension à l'aide des études existantes sur l'analyse dynamique de bras manipulateur. Par exemple, Máximo A. Roa et Raúl Suárez [4] présentent le fameux modèle dans le domaine robotique.

Contacts

Dans la section Bases méthodologiques, nous avons introduit les trois types de contact entre la pince et l'objet à saisir :

1. Contact ponctuel sans friction ;
2. Contact ponctuel avec frottement ;
3. Contact mou.

Si nous utilisons r pour représenter le nombre de composantes indépendantes de moment appliquées à chaque point de contact, alors r dépend du type de contact : $r = 1$ pour les points de contact sans friction, $r = 2$ et $r = 3$ pour les contacts ponctuels avec frottement en deux et trois dimensions respectivement et $r = 4$ pour les contacts mous.

Positions

Si une force F_i est appliquée en un point de l'objet, un moment $\tau_i = p_i \times F_i$ est généré par rapport au centre de masse de l'objet.

Nous représentons la force et le moment dans un vecteur $\omega_i = (F_i, \tau_i/\rho)$, où le paramètre ρ est introduit en raison de la dimension de la force et le moment.

Nous pouvons calculer ρ de deux façons. Dans la première, ρ est la distance maximale entre le centre de masse de l'objet et tout point de l'objet. Cela limite le maximum du moment à celui de

la force, qui est donc généralement considéré comme inadéquat. Une autre approche considère que ρ est le rayon de giration de l'objet, qui a une signification physique en termes d'énergie, mais il est moins utilisée dans les calculs pratiques car il est plus complexe à calculer.

Par conséquent, pour 2D objets, la dimension d du vecteur ω est $d = 3$ et pour 3D objets, $d = 6$.

Forces et vitesses

Pour la vitesse, nous utilisons v pour représenter la vitesse linéaire par rapport au centre de masse de l'objet et w pour représenter la vitesse angulaire par rapport au centre de masse de l'objet. Les deux vitesses, nous les représentons dans un vecteur twist $\dot{x} = (v, w)^T$, $\dot{x} \in \mathbb{R}^d$.

Pour une pince, il est possible qu'elle se compose des doigts dont chacun contient m articulations. Ici m représente le nombre d'articulations de chaque doigt. La force f_i appliquée sur le doigt i vient du moment, représenté par T_{ij} , $j = 1, \dots, m$. Dans le cas où il y a n doigts, nous représentons le moment appliqué sur l'articulation de la pince dans un vecteur $T = [T_{1j}^T \dots T_{nj}^T]^T$, $T \in \mathbb{R}^{nm}$. Pareil, les vitesses des articulations $\dot{\theta}_{ij}$ pourront éventuellement être représentées dans un vecteur $\dot{\theta} = [\dot{\theta}_{1j}^T \dots \dot{\theta}_{nj}^T]^T$, $\dot{\theta} \in \mathbb{R}^{nm}$.

Pour représenter la force appliquée et la vitesse du doigt, nous pouvons introduire deux vecteurs f et v . Ici le vecteur $f = [f_{1k}^T \dots f_{nk}^T]^T$, $f \in \mathbb{R}^{nr}$ ($k = 1, \dots, r$) contient toutes les composantes de la force appliquée en points de contact, et le vecteur $v = [v_{1k}^T \dots v_{nk}^T]^T$, $v \in \mathbb{R}^{nr}$ ($k = 1, \dots, r$) contient toutes les composantes de la vitesse des doigts.

2 Relation entre forces et vitesses

Dans les études de Murray et al.[2], ils ont montré la relation entre la force et la vitesse dans un diagramme.

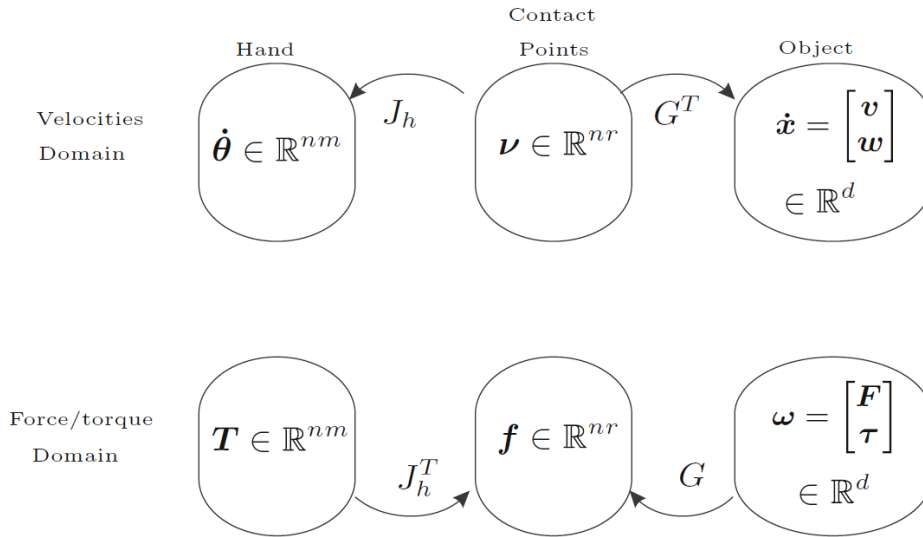


Figure 3.1 – Relation entre la force de préhension et la vitesse

Dans le diagramme, J_h représente la matrice Hand Jacobian, où $J_h = \text{diag}[J_1, \dots, J_n]$, $J_h \in \mathbb{R}^{nr \times nm}$. La matrice Hand Jacobian vient des matrices Hand Jacobian Partiel J_i , $i = 1, \dots, n$, $J_i \in \mathbb{R}^{r \times m}$, qui représentent la matrice Jacobian de chaque doigt.

Nous pouvons observer que la matrice Hand Jacobian met en relation la force f et le moment T , la vitesse v et la vitesse θ :

$$v = J_h \dot{\theta} \quad (1)$$

$$T = J_h^T f \quad (2)$$

De même, la matrice de préhension, représentée par $G, G \in \mathbb{R}^{d \times nr}$, met en relation entre la force f et le moment ω appliqué à l'objet, la vitesse v aux points de contact et le twist \dot{x} :

$$v = G^T \dot{x} \quad (3)$$

$$\omega = G f \quad (4)$$

Grâce à l'équation (1) et l'équation (3), nous pouvons éventuellement obtenir la relation entre la vitesse de doigts et la vitesse de l'objet :

$$J_h \dot{\theta} = G^T \dot{x} \quad (5)$$

L'équation (3) aussi nous permet d'obtenir la vitesse de l'objet à partir des vitesses des points de contact en faisant :

$$\dot{x} = (G^T)^+ v + N(G^T) v_0 \quad (6)$$

Dans l'équation (6), $(G^T)^+$ représente le pseudo-inverse de la matrice G^T , $N(G^T)$ forme une base pour la valeur null de la matrice G^T et v_0 est un vecteur arbitraire. Généralement, le pseudo-inverse n'est pas une matrice carrée.

Afin de appliquer du moment sur l'objet, il faut que $N(G^T)$ soit égale à 0 ou $rank(G)$ soit égale à d . Par conséquent, nous pouvons obtenir l'équation (6) comme $\dot{x} = (G^T)^+ v$.

Dans le domaine de la vitesse, la transformation directe de l'espace de haute dimension de l'articulation à l'espace de basse dimension de l'objet peut être obtenue à partir de la matrice Hand-object Jacobian H comme suit :

$$\dot{x} = H \dot{\theta} \quad (7)$$

où $H = (G^T)^+ J_h, H \in \mathbb{R}^{d \times nm}$.

3 Evaluation de préhension

Les métriques d'évaluation choisies pour la qualité de préhensions doivent tenir compte des propriétés de l'objet (forme, taille, poids), de la forme de friction ainsi que des conditions de fermeture de la force pour quantifier les préhensions.

Les métriques d'évaluation que nous avons sélectionnées peuvent être divisées en deux sous-groupes [5] :

1. propriétés algébriques de la matrice de préhension G ;
2. relations géométriques des points de contact.

3.1 Métriques sur la matrice de préhension G

Minimum des valeurs singulières de G

Etant une matrice de rang complet, la matrice de préhension G possède six valeurs singulières. Nous pouvons les calculer à partir des valeurs propres de GG^T . Lorsqu'une préhension est dans une configuration singulière, au moins une des valeurs singulières de G passe à zéro, et la préhension perd la capacité de résister à des moments externes dans au moins une direction.

Par conséquent, le minimum des valeurs singulières de la matrice de préhension G , $\sigma_{\min}(G)$, est une métrique de qualité qui indique dans quelle mesure la configuration de préhension est loin de tomber dans une configuration singulière.

$$Q_{MVS} = \sigma_{\min}(G) \quad (8)$$

Un grand $\sigma_{\min}(G)$ conduit à une meilleure préhension.

Volume de l'ellipsoïde dans l'espace de moment

Nous pouvons interpréter l'impact de l'équation (4) comme suit. L'équation (4) fait correspondre une sphère de rayon unitaire dans le domaine de force des points de contact à un ellipsoïde dans l'espace de moment. La contribution globale de toutes les forces des points de contact peut être considérée le volume de cet ellipsoïde.

$$Q_{VEM} = \sqrt{\det(GG^T)} = \sigma_1 \sigma_2 \dots \sigma_d \quad (9)$$

où $\sigma_1 \sigma_2 \dots \sigma_d$ représentent les valeurs singulières de la matrice G . À la différence de la métrique précédente, celle-ci considère toutes les valeurs singulières de manière égale, et pour la meilleur préhension, cette valeur doit être maximum. Mais Q_{VEM} ne permet pas de savoir si certains doigts contribuent plus que d'autres dans une préhension.

L'indice d'isotropie de préhension

Cette métrique recherche une contribution uniforme de la force au moment appliqué à l'objet. Elle tente d'obtenir une préhension isotrope où chaque force appliquée contribue de manière similaire à la force interne de l'objet.

$$Q_{IIP} = \frac{\sigma_{\min}(G)}{\sigma_{\max}(G)} \quad (10)$$

où $\sigma_{\min}(G)$ représente le minimum des valeurs singulières de G et $\sigma_{\max}(G)$ représente le maximum. Cet indice s'approche de 1 lorsque la préhension est isotrope (cas optimal), et tombe à zéro lorsque la préhension est proche d'une configuration singulière. Q_{IIP} indique si la préhension a un comportement équivalent dans n'importe quelle direction.

3.2 Métriques sur la relation géométrique des points de contact

Pour une pince à deux doigts, les métriques de cette partie sont très limitées. Car il n'y a que deux points de contact.

Distance entre le centre des points de contact et le centre de masse de l'objet

L'effet d'inertie et de gravitation sur la préhension est minimisé lorsque la distance entre le centre de masse de l'objet CM et le centre des points de contact C est minimale.

$$Q_{DCC} = Dist(CM, C) \quad (11)$$

Le calcul de la Q_{DCC} n'est plus facile que si le centre de masse est connu. Mais en pratique, il est difficile de connaître le centre de masse d'un objet. La densité de l'objet ou la forme de l'objet peut être inconnue.

4

Analyse et conception

1 Analyse

Dans les chapitres précédents, nous avons analysé des modèles de préhension, en complétant la tâche de modélisation de la relation entre la force et la vitesse dans la préhension par un bras Manipulateur. En outre, dans le modèle de préhension, nous avons trouvé la matrice de préhension qui est cruciale pour l'évaluation de la préhension.

D'une part, les valeurs singulières de la matrice de préhension sont importantes pour l'évaluation de la préhension et, d'autre part, il est tout aussi important de calculer la distance entre les points de contact et le centre de masse de l'objet.

Dans la section précédente, nous avons expliqué pourquoi les métriques sélectionnées peuvent évaluer la qualité de la préhension. Dans cette section, nous nous concentrons donc sur la modélisation sur les points de contact et de calculer la matrice de préhension partielle et ainsi obtenir la matrice de préhension complète.

2 Modélisation mathématique

Une préhension est définie comme un ensemble de contacts sur la surface de l'objet, dont le but est de contraindre les mouvements potentiels de l'objet en cas de perturbations externes.

L'analyse de la préhension consiste à déterminer si la préhension est stable en utilisant propriétés communes de clôture, étant donné un objet et un ensemble de points de contact. Ensuite, des métriques de qualité peuvent être évaluées afin de permettre au bras manipulateur de sélectionner la meilleure prise à exécuter.

2.1 Modélisation des contacts

Considérons un bras manipulateur en contact avec un objet rigide dont la position et l'orientation sont spécifiées par la localisation de l'origine d'un repère de coordonnées O fixé à l'objet et l'orientation de ce repère de coordonnées par rapport à un repère d'inertie W fixé dans le monde (Voir Figure 4.1).

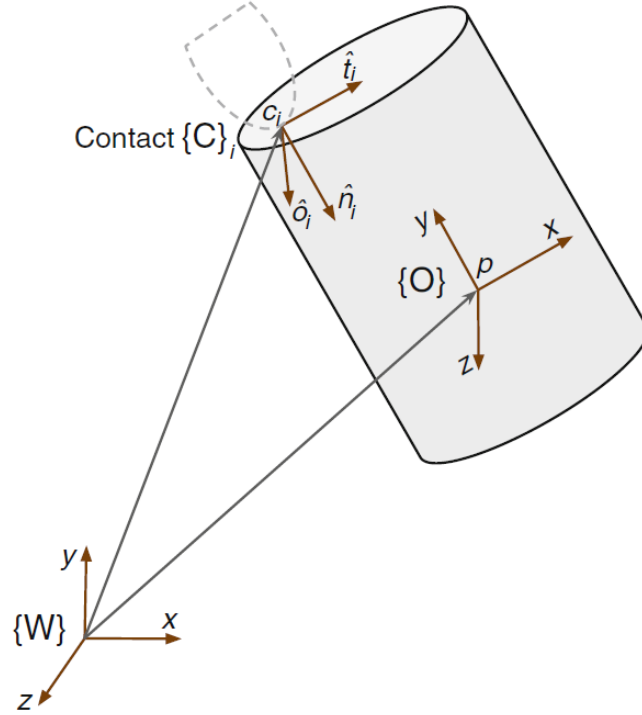


Figure 4.1 – Modélisation des contacts

Soit $p \in \mathbb{R}^3$ la position de l'objet et $c_i \in \mathbb{R}^3$ la position d'un point de contact i par rapport à W . A ce point de contact, nous définissons un repère C_i avec axes $\{\hat{n}_i, \hat{t}_i, \hat{o}_i\}$ où \hat{n}_i est le vecteur normale au plan tangent au point de contact et se pointe vers l'objet. Les deux autres vecteurs unitaires sont orthogonaux et se trouvent dans le plan tangent au point de contact.[3]

2.2 Analyse de la préhension

Une fois que le modèle de contact a été établi, il peut être utilisé pour étudier des tâches impliquant des contacts multiples. L'ensemble des points de contact définissant une préhension peut être analysé afin de tester la capacité de la préhension à résister aux perturbations et ses propriétés de dextérité. Comme cela sera présenté par la suite, les préhensions qui peuvent être maintenues pour toutes les perturbations possibles sont appelées préhensions de clôture. Cependant, il existe généralement plus d'une préhension qui remplit cette condition. C'est pourquoi de nombreuses métriques ont été proposées pour évaluer la qualité des préhensions sélectionnées et déterminer laquelle est la meilleure à exécuter.

2.2.1 Matrice de préhension

Il y a deux matrices qui sont souvent utilisées dans l'analyse de la préhension : la matrice de préhension G et la matrice jacobienne de la main J . Elles définissent les propriétés pertinentes de cinématique de la vitesse et de transmission de la force des contacts. Étant donné le grand nombre de variables nécessaires au calcul de la matrice jacobienne de la main J , nous ne nous intéressons et ne discutons ici que de la matrice de préhension G .

Matrice de préhension partielle

La transposée de la matrice de préhension partielle $\tilde{G}_i \in \mathbb{R}^{6 \times 6}$ met en relation la torsion de l'objet de W au repère de contact :

$$t_{i,obj} = \tilde{G}_i^T t \quad (1)$$

où t et $t_{i,obj}$ représentent la torsion d'objet liée à W et à C_i , respectivement.

\tilde{G}_i peut être calculé comme suit :

$$\tilde{G}_i = \begin{pmatrix} R_i & 0 \\ S(c_i - p) & R_i \end{pmatrix} \quad (2)$$

où $R_i \in \mathbb{R}^{3 \times 3}$ représente la matrice de rotation du repère de contact C_i par rapport à W , c_i la position du point de contact, p la position de l'objet.

$S(c_i - p)$ est la matrice de produit vectoriel, qui, étant donné un vecteur $r = [r_x, r_y, r_z]^T$, $S(r)$ est défini comme :

$$S(r) = \begin{pmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{pmatrix} \quad (3)$$

Matrice de préhension complète

La matrice de préhension complète $\tilde{G} \in \mathbb{R}^{6 \times 6n_c}$ est la combinaison des matrices de préhension partielle pour chacun des n_c points de contact.

$$\tilde{G}^T = \begin{pmatrix} \tilde{G}_1^T \\ \vdots \\ \tilde{G}_{n_c}^T \end{pmatrix} \quad (4)$$

Elle fait correspondre la torsion de l'objet de W à C :

$$t_{c,obj} = \tilde{G}^T t \quad (5)$$

où $t_{c,obj} \in \mathbb{R}^{6n_c}$ est un vecteur contenant toutes les torsions de l'objet dans le repère de contact :

$$t_{c,obj} = (t_{1,obj}^T \cdots t_{n_c,obj}^T)^T \quad (6)$$

3 Conception logicielle

3.1 Classe

La partie Conception logicielle est illustrée dans le diagramme de classe comme Figure 4.2

Classe Error

Classe de base pour les exceptions dans le module

Classe InputFormatError

Exception à lever pour les erreurs dans le format d'entrée

Classe LoadFile

Classe pour lire les données dans une liste de type Numpy. Elle contient un attribut pour stocker des objets Grasp.

Classe Grasp

Classe pour calculer la matrice de préhension et les métriques de préhension. Un objet Grasp est défini par deux points et le centre de masse de l'objet.

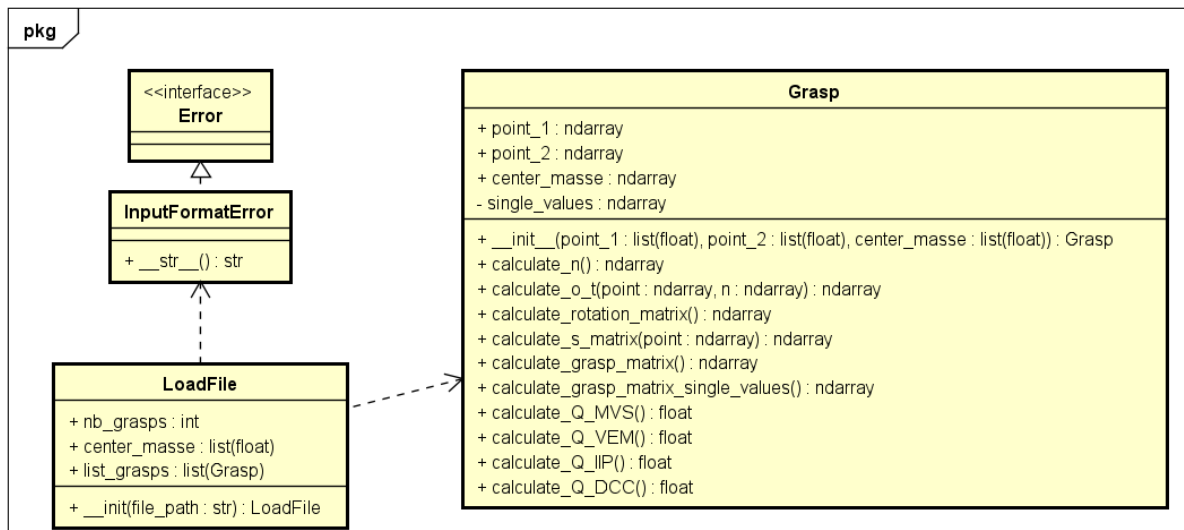


Figure 4.2 – Diagramme de classe

3.2 Interface

Un programme de console simple a été conçu pour recevoir l'entrée de l'utilisateur pour la commodité. Pour chaque fichier de données, le programme calculera la valeur de chacun des 4 métriques de préhension, puis normalisera le même métrique pour que les valeurs soient comprises entre 0 et 1. En calculant la somme des quatre métriques, nous pouvons obtenir un score pour chaque préhension. Par conséquent, le programme renvoie la meilleure préhension et son score.

5

Mise en oeuvre

Dans ce chapitre, on explique en détail la réalisation de l'algorithme proposé pour calculer les 4 métriques de qualité de préhension. On explique aussi le résultat obtenu sur un fichier de test.

1 Outils et librairie utilisés

1.1 Numpy

Dans les chapitres précédents, nous avons analysé la manière de calculer la matrice de préhension. Nous pouvons observer que beaucoup d'opérations matricielles sont effectuées dans ce processus. En ce qui concerne le type de données list à Python, sa structure n'est pas intuitive, ni facile à manipuler.

Par conséquent, dans notre mise en œuvre, nous utilisons principalement Numpy pour stocker les données de points, et il existe de nombreuses fonctions bien définies pour les opérations matricielles dans Numpy qui facilitent également le processus de calcul.

1.2 HtmlTestRunner

Pour la partie tests unitaires, nous avons principalement utilisé le module unittest dans python. Nous avons conçu des tests unitaires pour chaque fonction des deux classes les plus importantes du projet : LoadFile et Grasp et utilisé le module HtmlTestRunner qui permet la visualisation des résultats des tests unitaires. Une description plus détaillée se trouve dans les annexes suivantes.

1.3 Sphinx

Pour mieux maintenir le projet, nous avons utilisé Sphinx pour commenter notre code. Avec sa spécification de commentaire standard, nous pouvons présenter les commentaires complets comme une page .html. La documentation qui en résulte est très claire et esthétique.

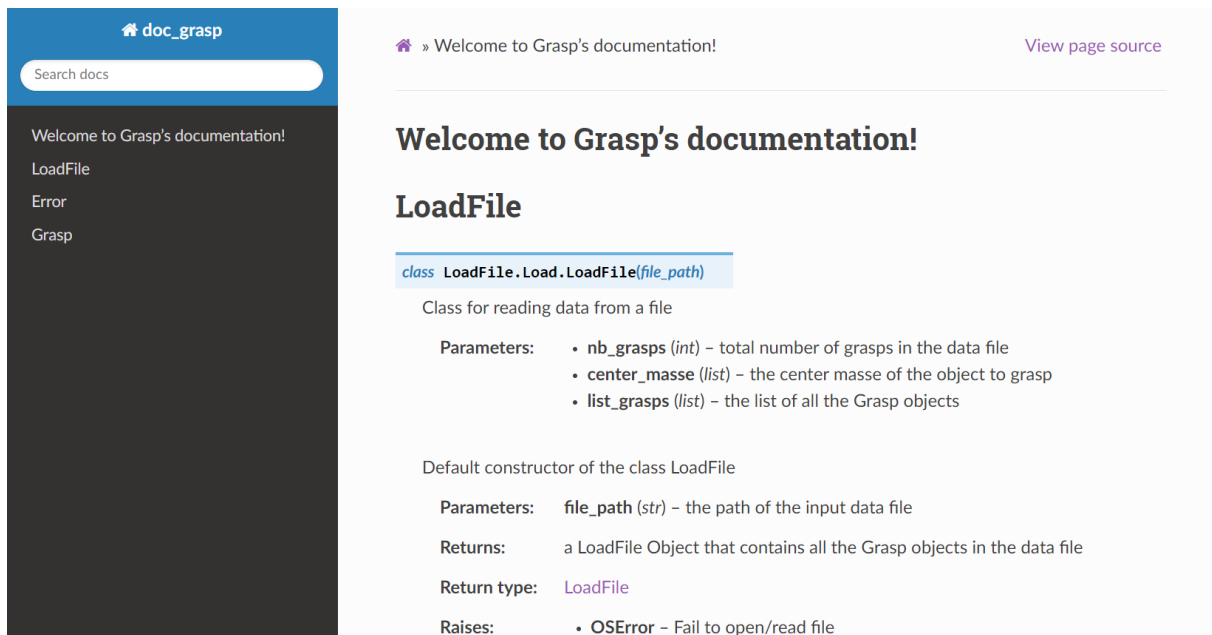


Figure 5.1 – Documentation

2 IHM

Pour mieux gérer les entrées utilisateur, nous avons compilé le fichier `main.py` avec `pyinstaller` pour générer un exécutable.

Lorsque l'utilisateur ouvre ce programme, celui-ci lui demande de saisir l'emplacement du fichier de données. Si le fichier n'existe pas ou ne peut pas être ouvert, l'interface du programme renvoie un message d'erreur et demande à nouveau à l'utilisateur d'entrer des données jusqu'à ce que le programme réussisse à ouvrir le fichier.

Après avoir ouvert le fichier de données, le programme vérifie que le fichier d'entrée a le format requis : d'abord le nombre de préhensions présentes dans le fichier, puis les coordonnées du centre de masse de l'objet et enfin les coordonnées 3D des deux points de contact pour chaque préhension. S'il y a un problème avec le format du fichier, ou si le type de données est erroné, le programme indiquera également que le fichier est dans le mauvais format et demandera à l'utilisateur de le saisir à nouveau.

Dans le programme, il d'abord génère un objet de type `LoadFile`, qui contient une liste de tous les objets du type `Grasp`. Ensuite, les 4 métriques sont calculées pour chaque objet du type `Grasp` dans cette liste. Chaque métrique est ensuite normalisée entre 0 et 1. En faisant la somme, nous pouvons obtenir un score pour chaque préhension, ainsi que la meilleure préhension dans ce fichier de données.

Le code est suivant :

```

1 from LoadFile.Load import *
2
3 # Read data from file
4 while True:
5     print(" *" * 40)
6     file_path = input("Import the data file: ")
7     try:
8         data = LoadFile(file_path)
9     except OSError:
10        # Handle the exception

```

```

11         print("Could not open/read file:", file_path)
12         continue
13     except ValueError:
14         # Handle the exception
15         print("Wrong Format: line 1")
16         continue
17     except InputFormatError:
18         # Handle the exception
19         print("Wrong Format!")
20         continue
21     except:
22         # Handle the exception
23         print("Wrong Format!")
24         continue
25     print("File " + file_path + " imported.")
26     break
27
28 # Calculate the 4 quality metrics of each grasp
29 list_grasp_qualities = []
30 for grasp in data.list_grasps:
31     list_grasp_qualities.append([
32         grasp.calculate_Q_MVS(),
33         grasp.calculate_Q_VEM(),
34         grasp.calculate_Q_IIP(),
35         grasp.calculate_Q_DCC()])
36
37 # Scale each column(quality metric) to 0 - 1, 1 means better than 0
38 list_grasp_qualities = np.array(list_grasp_qualities)
39 for i in range(4):
40     max_column_i = list_grasp_qualities[:, i].max()
41     min_column_i = list_grasp_qualities[:, i].min()
42     for j in range(len(list_grasp_qualities)):
43         list_grasp_qualities[j][i] =
44             (list_grasp_qualities[j][i] - min_column_i)/
45             (max_column_i - min_column_i)
46         # for the quality metric Q_DDC, reverse to 1 - 0
47         if i == 3:
48             list_grasp_qualities[j][i] =
49                 1 - list_grasp_qualities[j][i]
50
51 print("*" * 40)
52 print("4 quality metrics of each grasp")
53 for grasp_qualities in list_grasp_qualities:
54     print(grasp_qualities)
55
56 # sum up the four quality metrics and obtain the final score
57 # of the grasp
58 list_score = []
59 max_score = 0
60 max_score_index = 0
61 for i in range(len(list_grasp_qualities)):
62     list_score.append(sum(list_grasp_qualities[i]))
63     if sum(list_grasp_qualities[i]) > max_score:
64         max_score = sum(list_grasp_qualities[i])
65         max_score_index = i
66
67 print("*" * 40)
68 print("Score: " + " ".join(str(i.round(3)) for i in list_score))

```

```

69 # show the score of the best grasp and its index (start by 0)
70 print("best grasp score and index (start by 0): ")
71 print(max_score.round(3), max_score_index)
72
73 input("Press Enter to continue...")
74 }

```

3 Lecture de données

Comme nous l'avons mentionné précédemment, pour la lecture des données, nous générons un objet de type LoadFile. Dans l'implémentation, cet objet contient trois attributs : le nombre de préhensions, le centre de masse de l'objet et une liste de toutes les objets Grasp. La classe sont comme suivante.

```

1  import sys
2  sys.path.append("../")
3  from Grasp.Grasp import *
4  from Error.Error import *
5
6
7  class LoadFile:
8      """Class for reading data from a file
9
10         :param int nb_grasps: total number of grasps in the data file
11         :param list center_masse: the center masse of the object to grasp
12         :param list list_grasps: the list of all the Grasp objects
13     """
14
15     def __init__(self, file_path):
16         """Default constructor of the class LoadFile
17
18         :param str file_path: the path of the input data file
19
20         :returns: a LoadFile Object that contains all the Grasp objects
21         in the data file
22         :rtype: LoadFile
23
24         :raises OSError: Fail to open/read file
25         :raises ValueError: Fail to transform string to int/float
26         :raises InputFormatError: Wrong format of the file
27     """
28
29
30     # Open file and read
31     file = open(file_path, "r")
32
33     # Read the number of grasps in the file
34     self.nb_grasps = int(file.readline().strip())
35
36     # Read the center masse of the object
37     self.center_masse =
38     list(map(float, file.readline().strip().split()))
39     if len(self.center_masse) != 3:
40         raise InputFormatError()
41
42     # list of grasps

```

```

43     self.list_grasps = []
44     for i in range(self.nb_grasps):
45         # Read the two points in one row
46         list_xyz =
47             list(map(float, file.readline().strip().split()))
48         if len(list_xyz) != 6:
49             raise InputFormatError()
50         point_1 = list_xyz[:3]
51         point_2 = list_xyz[3:]
52         # Add the Grasp to the list
53         self.list_grasps.append(Grasp(point_1, point_2,
54             self.center_masse))
55     file.close()

```

4 Calcul matriciel

4.1 Constructeur

Chaque objet de type Grasp est constitué des coordonnées de deux points et des coordonnées du centre de masse de l'objet. Dans le constructeur, il prend des données de type list et stocke les coordonnées dans un Numpy array.

```

1  import numpy as np
2  np.seterr(all='raise')
3
4
5  class Grasp:
6      """Class for calculating the quality metrics of a grasp
7
8      :param ndarray point_1: contact point 1
9      :param ndarray point_2: contact point 2
10     :param ndarray center_masse: center masse of the object to grasp
11     :param ndarray __single_values: the single values of the grasp
12     matrix
13     """
14
15     # point -> [x, y, z]
16     # grasp -> [point_1, point_2, center_masse]
17     def __init__(self, point_1, point_2, center_masse):
18         """Default constructor of the class Grasp, convert the input list
19         object to Numpy array
20
21         :param ndarray point_1: contact point 1
22         :param ndarray point_2: contact point 2
23         :param ndarray center_masse: center masse of the object to
24         grasp
25
26         :returns: a Grasp Object
27         :rtype: Grasp
28         """
29
30     self.point_1 = np.array(point_1)
31     self.point_2 = np.array(point_2)
32     self.center_masse = np.array(center_masse)
33     self.__single_values = np.array([])

```

4.2 Calculer le vecteur n

Afin de calculer la matrice de préhension partielle sur chaque point de contact, il faut d'abord déterminer le repère sur le point de contact. Dans la section précédente sur les hypothèses, nous considérons que pour une pince à deux doigts, les forces appliquées à chaque point de contact sont dans des directions opposées. Par conséquent, nous pouvons obtenir le vecteur n par ici.

```

1  def calculate_n(self):
2      """Calculate the vector n by calculating the vector from point_1 to
3      point_2 and from point2 to point1
4
5      :returns: the corresponding vector n of point_1 and point_2
6      :rtype: ndarray
7      """
8
9      return self.point_2 - self.point_1, self.point_1 - self.point_2

```

4.3 Calculer le vecteur t et o

Avec le point de contact et le vecteur n , nous pouvons calculer un plan où se trouvent le vecteur t et le vecteur o . Finalement, nous pouvons établir un système linéaire avec 3 équations et 6 variables. Nous avons des solutions illimitées. Par conséquent, Nous pouvons d'abord fixer deux variables aléatoirement, puis une autre et éventuellement obtenir une solution.

Dans le calcul, il faut faire attention à l'exception ZeroDivision. Nous utilisons 3 structure try...except... afin d'éviter ce problème.

```

1  def calculate_o_t(self, point, n):
2      """Calculate a random set of vector o and vector t with the point
3      and its vector n
4
5      The three vector n, t, o is perpendicular to each other.
6
7      if we have the normal vector n = [A, B, C], the point
8      P = [px, py, pz]
9
10     point T and O are in the plane: Ax + By + Cz + D = 0,
11     with D = -A*px -B*py -C*pz
12
13     the vector P,T are perpendicular to the vector PO:
14
15     (tx - px)(ox - px) + (ty - py)(oy - py) + (tz - pz)(oz - pz)
16     = 0
17
18     Atx + Bty + Ctz + D = 0
19
20     Aox + Boy + Coz + D = 0
21
22     3 equations, 6 variables ==> illimited solutions
23
24     ox, oy = random integer ==> oz ==> point O
25
26     tx = random integer ==> 3 equations, 3 variables ==> point T
27
28     vector o = vector PO, vector t = vector PT
29

```



```

30     :param ndarray point: contact point
31     :param ndarray n: the vector n of the contact point
32     in this Grasp
33
34     :returns: the vector o and vector t of the contact point
35     with the vector n
36     :rtype: ndarray
37
38     :raises ZeroDivisionError: a number is divided by a zero
39     """
40
41     # randomly define ox and oy, we can calculate oz
42     ox = np.random.randint(50)
43     while ox == point[0]:
44         ox = np.random.randint(50)
45     oy = np.random.randint(50)
46     while oy == point[1]:
47         oy = np.random.randint(50)
48     try:
49         # exception: division by zero
50         oz = ((n * point).sum() - n[0] * ox - n[1] * oy) / n[2]
51     except:
52         oy = np.random.randint(50)
53         while oy == point[1]:
54             oy = np.random.randint(50)
55         oz = np.random.randint(50)
56         while oz == point[2]:
57             oz = np.random.randint(50)
58     try:
59         ox = ((n * point).sum() - n[1] * oy - n[2] * oz) / n[0]
60     except:
61         ox = np.random.randint(50)
62         while ox == point[0]:
63             ox = np.random.randint(50)
64         oz = np.random.randint(50)
65         while oz == point[2]:
66             oz = np.random.randint(50)
67         oy = ((n * point).sum() - n[0] * ox - n[2] * oz) / n[1]
68     # vector o = point_o - point
69     o = np.array([ox - point[0], oy - point[1], oz - point[2]])
70
71     # randomly define tx, we can build a linear system
72     with 3 equations, 3 variables
73     random_tx = np.random.randint(50)
74     while random_tx == point[0] or random_tx == ox:
75         random_tx = np.random.randint(50)
76     a = np.array([[n[0], n[1], n[2]],
77                  [ox - point[0], oy - point[1], oz - point[2]],
78                  [1, 0, 0]])
79     b = np.array(
80         [(n * point).sum(),
81          ox * point[0] + oy * point[1] + oz * point[2] -
82          (point * point).sum(),
83          random_tx])
84     try:
85         # exception: the determinant of the matrix a = 0
86         t = np.linalg.solve(a, b)
87     except:

```

```

88     random_ty = np.random.randint(50)
89     while random_ty == point[1] or random_ty == oy:
90         random_ty = np.random.randint(50)
91     a = np.array([[n[0], n[1], n[2]],
92                 [ox - point[0], oy - point[1], oz - point[2]],
93                 [0, 1, 0]])
94     b = np.array(
95         [(n * point).sum(),
96          ox * point[0] + oy * point[1] + oz * point[2]
97          - (point * point).sum(),
98          random_ty])
99     try:
100         t = np.linalg.solve(a, b)
101     except:
102         random_tz = np.random.randint(50)
103         while random_tz == point[2] or random_tz == oz:
104             random_tz = np.random.randint(50)
105         a = np.array([[n[0], n[1], n[2]],
106                     [ox - point[0], oy - point[1], oz - point[2]],
107                     [0, 0, 1]])
108         b = np.array(
109             [(n * point).sum(),
110              ox * point[0] + oy * point[1] + oz * point[2]
111              - (point * point).sum(),
112              random_tz])
113         t = np.linalg.solve(a, b)
114     t = t - point
115
116     # verify the direction of the vector t
117     # cross_product(a, b) = norm(a) * norm(b) * sin(theta) * vector_n
118     # sin(theta) = 1 or -1
119     n = n/np.linalg.norm(n)
120     cross_product_ot = np.cross(o, t)/
121     (np.linalg.norm(o) * np.linalg.norm(t))
122     # if sin(theta) == 1, wrong direction of the vector t
123     if np.array_equal(cross_product_ot, n):
124         return o, -t
125     else:
126         return o, t

```

4.4 Calculer la matrice de rotation

Après avoir le repère C sur le point de contact, nous pouvons obtenir la matrice de rotation en normalisant les trois vecteurs n , t , et o . De même, nous pouvons calculer la matrice S par sa définition.

```

1  def calculate_rotation_matrix(self):
2      """Calculate the rotation matrix of the two contact points
3
4      First, normalize the vectors n, t, o. Then calculate the
5      rotation matrix.
6
7      rotation matrix = [n.T, t.T, o.T]
8
9      :returns: 2 rotation matrix
10     :rtype: ndarray
11     """

```

```

12
13     n_1, n_2 = self.calculate_n()
14     o_1, t_1 = self.calculate_o_t(self.point_1, n_1)
15     o_2, t_2 = self.calculate_o_t(self.point_2, n_2)
16     r_1 = np.column_stack(((n_1/np.linalg.norm(n_1),
17     t_1/np.linalg.norm(t_1),
18     o_1/np.linalg.norm(o_1)))
19     r_2 = np.column_stack(((n_2/np.linalg.norm(n_2),
20     t_2/np.linalg.norm(t_2),
21     o_2/np.linalg.norm(o_2)))
22     return r_1, r_2
23
24 def calculate_s_matrix(self, point):
25     """Calculate the S Matrix of a contact point
26
27      $r = point - center\_masse$ 
28
29      $S(r) =$ 
30
31          $\begin{bmatrix} 0 & -rz & ry \\ rz & 0 & -rx \\ -ry & rx & 0 \end{bmatrix}$ 
32
33         :param ndarray point: contact point
34
35         :returns: the S matrix of the contact point
36         :rtype: ndarray
37     """
38
39     vector = point - self.center_masse
40     column_0 = np.array([0, vector[2], -vector[1]])
41     column_1 = np.array([-vector[2], 0, vector[0]])
42     column_2 = np.array([vector[1], -vector[0], 0])
43     s = np.column_stack((column_0, column_1, column_2))
44     return s
45
46
47
48

```

4.5 Calculer la matrice de préhension et ses valeurs singulières

En fonction de la définition de la matrice de préhension, nous pouvons facilement calculer la matrice de préhension complète et calculer ses valeurs singulières.

```

1  def calculate_grasp_matrix(self):
2      """Calculate the Grasp Matrix
3
4       $G(point) =$ 
5
6       $[R(point), 0]$ 
7
8       $[S(point - cm) * R(point), R(point)]$ 
9
10      $G = [G(point1), G(point2)]$ 
11
12     :returns: the Grasp matrix of the current grasp
13     :rtype: ndarray

```

```

14         """
15
16         r_1, r_2 = self.calculate_rotation_matrix()
17         s_1 = self.calculate_s_matrix(self.point_1)
18         s_2 = self.calculate_s_matrix(self.point_2)
19         g_1 = np.vstack((np.hstack((r_1, np.zeros((3, 3)))),
20         np.hstack((np.dot(s_1, r_1), r_1))))
21         g_2 = np.vstack((np.hstack((r_2, np.zeros((3, 3)))),
22         np.hstack((np.dot(s_2, r_2), r_2))))
23         g = np.hstack((g_1, g_2))
24         return g
25
26     def calculate_grasp_matrix_single_values(self):
27         """Calculate the single values of the Grasp Matrix
28
29         stock the results in the attribut self.__single_values
30
31         :returns: the single values of the Grasp Matrix
32         :rtype: ndarray
33         """
34
35         g = self.calculate_grasp_matrix()
36         u, s, vh = np.linalg.svd(g, full_matrices=True)
37         self.__single_values = s
38         return s

```

4.6 Calculer les métriques

Après avoir calculé les valeurs singulières de la matrice de préhension, nous avons pu calculer les 3 métriques de qualité qui leur sont associés, puis, en fonction de la position du centre de masse de l'objet, nous avons pu en calculer le quatrième.

```

1     def calculate_Q_MVS(self):
2         """Calculate the quality metric Q_MVS
3
4         Q_MVS = the minimum of the single values
5
6         :returns: the quality metric Q_MVS
7         :rtype: float
8         """
9
10        if self.__single_values.size == 0:
11            self.calculate_grasp_matrix_single_values()
12        return self.__single_values.min()
13
14    def calculate_Q_VEM(self):
15        """Calculate the quality metric Q_VEM
16
17        Q_VEM = s1*s2*s3...*sn
18
19        :returns: the quality metric Q_VEM
20        :rtype: float
21        """
22
23        if self.__single_values.size == 0:
24            self.calculate_grasp_matrix_single_values()

```

```

25         return np.prod(self.__single_values)
26
27     def calculate_Q_IIP(self):
28         """Calculate the quality metric Q_IIP
29
30         Q_IIP = min(s)/max(s)
31
32         :returns: the quality metric Q_IIP
33         :rtype: float
34         """
35
36         if self.__single_values.size == 0:
37             self.calculate_grasp_matrix_single_values()
38         return self.__single_values.min()/self.__single_values.max()
39     def calculate_Q_DCC(self):
40         """Calculate the quality metric Q_DCC
41
42         Q_DCC = the distance between the center of the vector P1P2 and
43         the center masse of the object
44
45         :returns: the quality metric Q_DCC
46         :rtype: float
47         """
48
49         center = (self.point_1 + self.point_2) / 2
50         return np.linalg.norm(center - self.center_masse)

```

5 Analyse des résultats

Dans cette section, on va valider la pertinence de l'algorithme développé en l'appuyant sur un fichier de données test.

5.1 Données d'entrée

Pour vérifier la pertinence des résultats obtenus, nous avons mise en oeuvre le programme sur des données de test.

Soit un objet de dimension $6 \times 8 \times 6$. Nous avons des préhensions possibles comme dans le fichier ci-dessous :

```

9
3 4 3
6 2 4.5 0 2 4.5
6 6 4.5 0 6 4.5
6 2 1.5 0 2 1.5
6 6 1.5 0 6 1.5
6 4 3 0 4 3
3 0 3 3 8 3
2 0 2 2 8 2
2 0 2 6 2 1.5
6 2 1.5 0 6 4.5

```

Figure 5.2 – Données d'entrée

En visualisant les préhensions dans l'outil geogebra, nous pouvons observer la Figure 5.3. Une préhension sont représentée par une ligne liée par les deux points de contact.

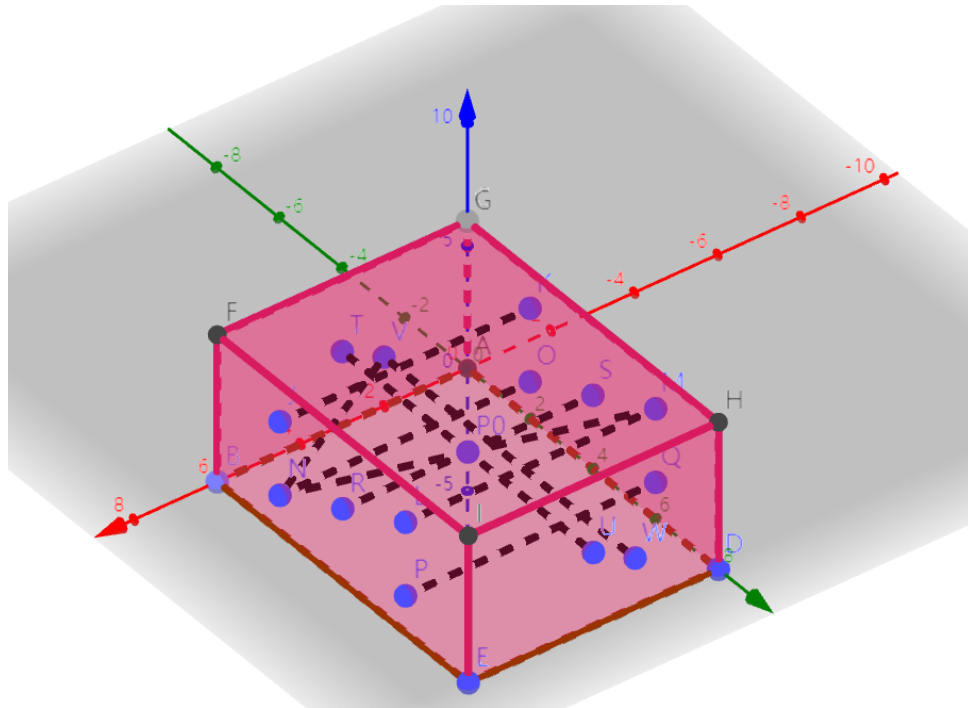


Figure 5.3 – Les préhensions

5.2 Récupérer les résultats

En saisissant le fichier de test dans le programme, nous pouvons éventuellement obtenir les résultats dans la Figure 5.4.

```

D:\prehesion\dist\main\main.exe
*****
Import the data file: D:\prehesion\Data\data.txt
File D:\prehesion\Data\data.txt imported.
*****
4 quality metrics of each grasp
[0.10688145 0.36      0.08764323 0.26478538]
[0.10688145 0.36      0.08764323 0.26478538]
[0.10688145 0.36      0.08764323 0.26478538]
[0.10688145 0.36      0.08764323 0.26478538]
[1.  0.36 1.  1.  ]
[1.  1.  0.70801003 1.  ]
[0.3364236 1.  0.21619052 0.5840998 ]
[0. 0. 0. 0. ]
[1.  0.93142857 0.72993675 1.  ]
*****
Score: 0.819 0.819 0.819 0.819 3.36 3.708 2.137 0.0 3.661
best grasp score and index (start by 0):
3.708 5
Press Enter to continue...

```

Figure 5.4 – Résultats

5.3 Analyse

Sur les résultats des calculs dans la Figure ci-dessus, nous observons que la première, deuxième, troisième et quatrième préhension ont le même score. Ceci est logique, car topologiquement ces quatre préhensions sont identiques.

Nous remarquons également que dans ces préhensions, la préhension où les points de contact sont sur deux faces adjacentes a un score plus faible que celles sur des faces opposées.

6

Bilan et conclusion

Ce projet se passe en 2 semestres, et le diagramme de Gantt initial et final sont dans l'annexes suivante. Pendant le semestre 9, je fait la partie de recherche, et la partie de développement pendant le semestre 10.

1 Bilan du semestre 9

Pendant le semestre 9, les tâches suivantes ont été réalisées :

- Analyser les besoins
- Recherche de solutions existantes
- Etudier le principe de la solution
- Valider la faisabilité de la solution
- Rédiger des spécifications

Les tâches à faire au semestre 10 sont les suivantes :

- Concevoir la modélisation de préhension
- Faire la modélisation logicielle telle que le diagramme de classe
- Programmer la modélisation
- Faire des tests et améliorer le programme
- rédiger le rapport et préparer la soutenance

2 Bilan du semestre 10

Toutes les taches sont listées dans les cartes de Trello comme Figure 6.1.

Pendant le semestre 10, les tâches suivantes ont été réalisées :

- Concevoir la modélisation de préhension
- Faire la modélisation logicielle telle que le diagramme de classe
- Programmer la modélisation
- Valider la faisabilité de la solution
- Faire des tests, améliorer le programme, rédiger la documentation
- rédiger le rapport et préparer la soutenance

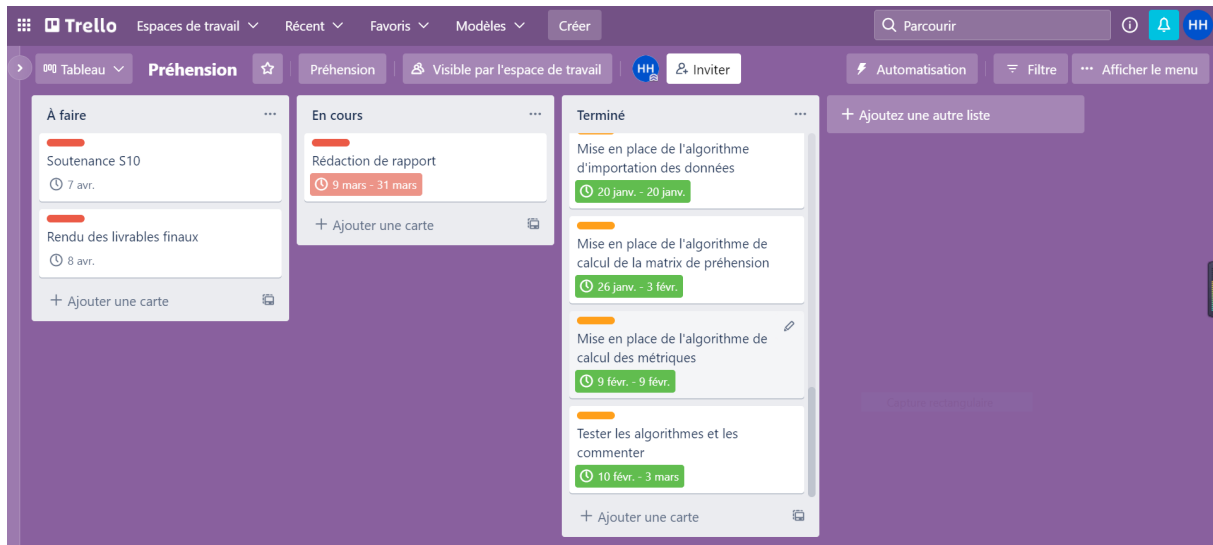


Figure 6.1 – Les tâches sur Trello

3 Bilan sur la qualité

La qualité des travaux est assurée dans ces aspects :

- La qualité du code et commentaires
- La qualité des instances de données
- La qualité d'analyse
- La qualité du rapport

Le projet a atteint la qualité de mise en œuvre requise, en tenant compte de la convention de nommage, de la convention de commentaire, des tests unitaires et du guide d'utilisation, qui sont tous importants pour la qualité du projet.

4 Bilan auto-critique

Pendant la conception et la réalisation, le PRD se passe généralement bien.

Au début du projet, J'ai rencontré beaucoup de problèmes, mais grâce aux aides de mes encadrants M. GAUCHER et M. MONMARCHE, j'ai pu analyser ses besoins et trouver une solution pour le sujet proposé. En raison de mes circonstances, je regrette de ne pas avoir rencontré régulièrement mes encadrants. J'ai eu mal à gérer la planification et la gestion du projet. Avec la rencontre avec Madame Marine HOGUET, j'ai appris d'utiliser les outils comme Git, Trello et Doodle. Et j'ai appris beaucoup pendant résoudre ces problèmes.

Comme je l'ai mentionné dans l'introduction, il y a également des améliorations qui peuvent être apportées à ce projet :

1. analyser l'image afin d'obtenir un ensemble de points de contact
2. manipuler le bras manipulateur

Pour l'instance, j'utilise le centre de masse pour représenter la position de l'objet. En réalité, il existe tellement de formes différentes d'objets qu'une telle solution est superficielle.

Finalement, je voudrais remercier tous les professeurs et intervenants que j'ai rencontrés pendant ce projet.

Annexes

A

Planification, gestion de projet

1 Evolution du projet

Le diagramme de Gantt initial pour la planification de ce projet :

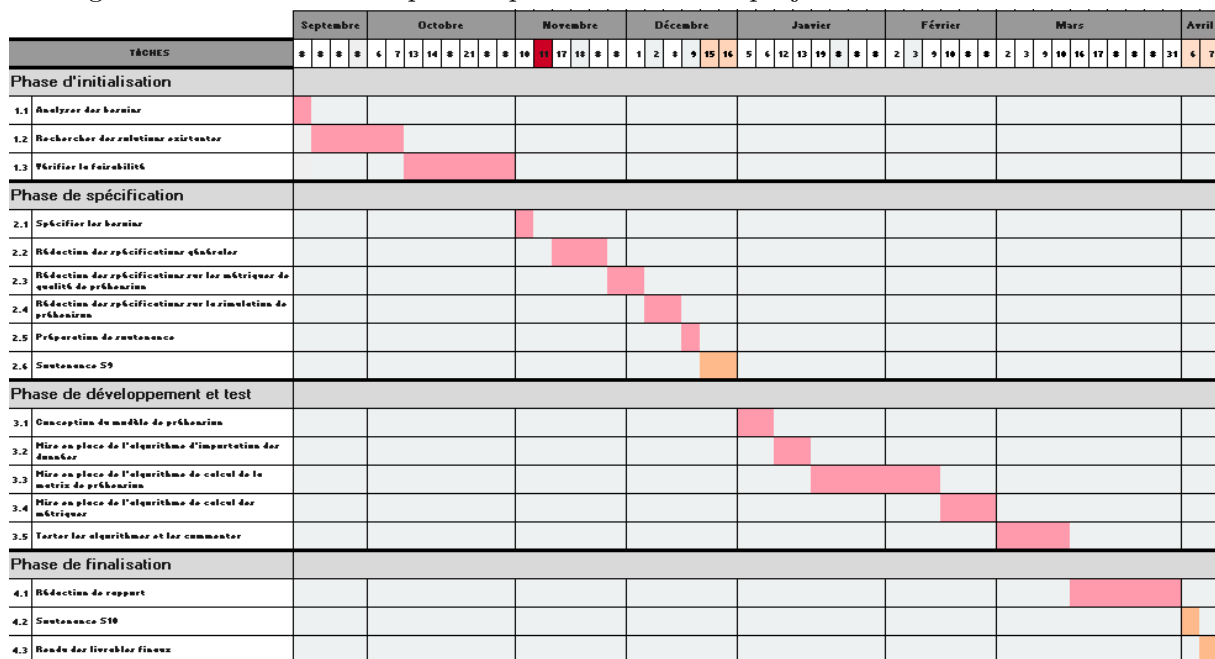


Figure A.1 – Le diagramme de Gantt initial

Le diagramme de Gantt final :

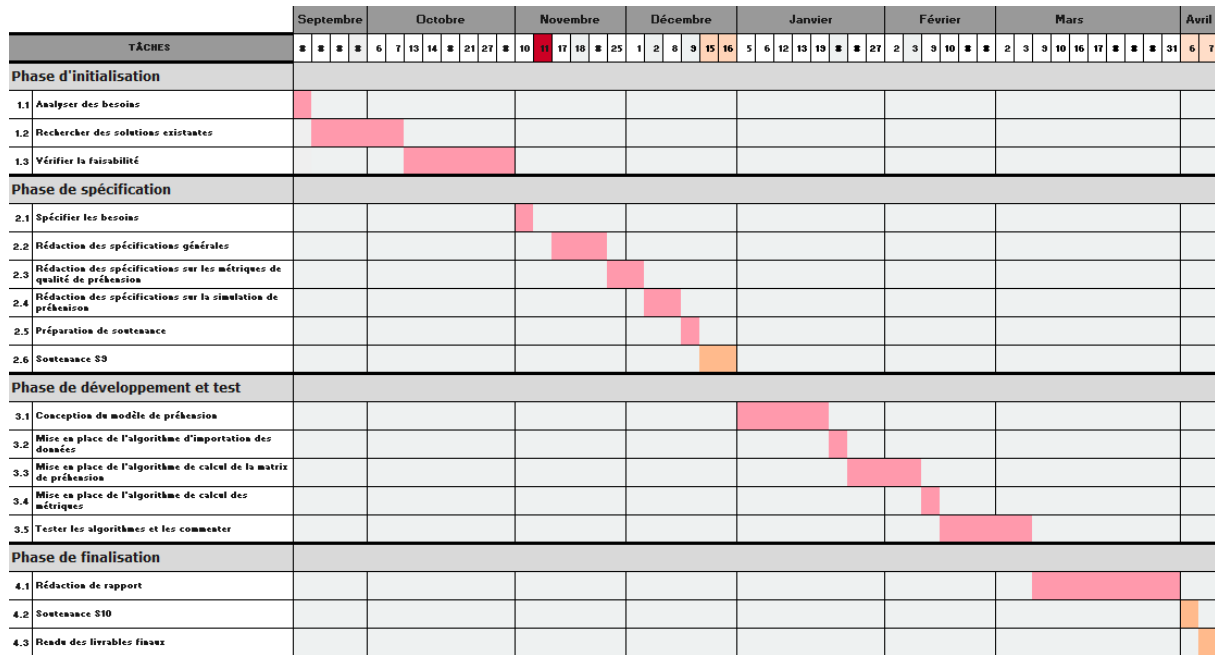


Figure A.2 – Le diagramme de Gantt final

2 Description des tâches

Tâche 1 : Analyser des besoins

- Date de début : 22/09/2021
- Date de fin : 22/09/2021
- Durée : 1 jour
- Description : rencontre avec les encadrants, analyser les besoins.

Tâche 2 : Rechercher des solutions existantes

- Date de début : 23/09/2021
- Date de fin : 07/10/2021
- Durée : 5 jours
- Description : lire des documents liés à ce sujet, petit synthèses conclurant les différentes méthodes existantes.

Tâche 3 : Vérifier la faisabilité

- Date de début : 13/10/2021
- Date de fin : 28/10/2021
- Durée : 6 jours
- Description : étudier la base de la robotique et vérifier les méthodes qui pourront être utilisées dans le projet.

Tâche 4 : Spécifier les besoins

- Date de début : 10/11/2021
- Date de fin : 10/11/2021
- Durée : 1 jour
- Description : spécifier les besoins et écrire le rapport de spécifications sur l'introduction du projet.

Tâche 5 : Rédaction des spécifications générales

- Date de début : 17/11/2021
- Date de fin : 24/11/2021
- Durée : 3 jours
- Description : spécifier les enjeux et le contexte et écrire le rapport de spécifications sur l'introduction du projet.

Tâche 6 : Rédaction des spécifications sur le modèle de préhension

- Date de début : 25/11/2021
- Date de fin : 01/12/2021
- Durée : 2 jours
- Description : spécifier le modèle de préhension et écrire le rapport de spécifications sur l'état de l'art.

Tâche 7 : Rédaction des spécifications sur les métriques d'évaluation de préhension

- Date de début : 02/12/2021
- Date de fin : 08/12/2021
- Durée : 2 jours
- Description : spécifier les métriques d'évaluation de préhension et écrire le rapport de spécifications sur l'état de l'art.

Tâche 8 : Préparation de soutenance

- Date de début : 09/12/2021
- Date de fin : 09/12/2021
- Durée : 1 jour
- Description : rédaction du rapport et préparer pour la soutenance.

Tâche 9 : Soutenance S9

- Date de début : 15/12/2021
- Date de fin : 16/12/2021
- Durée : 2 jours
- Description : présenter le projet devant le jury.

Tâche 10 : Conception du modèle de préhension

- Date de début : 05/01/2022
- Date de fin : 19/01/2022
- Durée : 5 jours
- Description : chercher la solution pour le calcul de la matrice de préhension

Tâche 11 : Mise en place de l'algorithme d'importation des données

- Date de début : 20/01/2022
- Date de fin : 20/01/2022
- Durée : 1 jour
- Description : concevoir une classe pour la lecture de données

Tâche 12 : Mise en place de l'algorithme de calcul de la matrice de préhension

- Date de début : 26/01/2022
- Date de fin : 03/02/2022
- Durée : 4 jours
- Description : modéliser la préhension et le calcul de la matrice de préhension

Tâche 13 : Mise en place de l'algorithme de calcul des métriques

- Date de début : 09/02/2022
- Date de fin : 09/02/2022
- Durée : 1 jour
- Description : implémenter des méthodes dans la classe Grasp

Tâche 14 : Tester les algorithmes et les commenter

- Date de début : 10/02/2022
- Date de fin : 03/03/2022
- Durée : 5 jours
- Description : mise en place des tests unitaires et la documentation

Tâche 15 : Rédaction de rapport

- Date de début : 09/03/2022
- Date de fin : 31/03/2022
- Durée : 8 jours
- Description : rédiger le rapport

Tâche 16 : Soutenance S10

- Date de début : 06/04/2022
- Date de fin : 06/04/2022
- Durée : 1 jour
- Description : présenter le projet devant le jury

Tâche 17 : Rendu des livrables finaux

- Date de début : 07/04/2022
- Date de fin : 07/04/2022
- Durée : 1 jour
- Description : rendre les livrables produits

B

Description des interfaces

1 Interfaces matérielles/logicielles

Pour utiliser le système d'évaluation de préhensions, l'utilisateur doit être en possession d'un ordinateur sur lequel le programme a été préalablement installée. Il doit également avoir une base de connaissance sur le format de fichier d'entrée. Le format doit être comme suit :

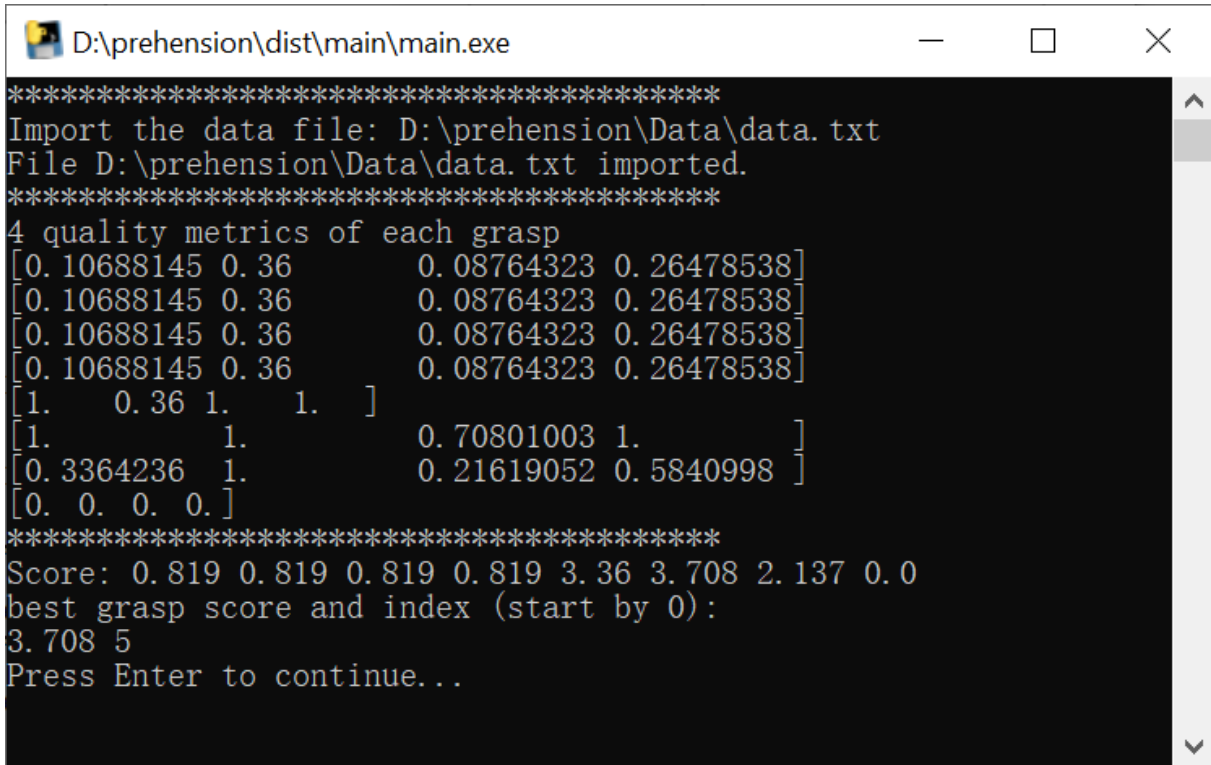
fichier prehension.txt :

```
8                - nombre de préhensions
1 1 1           - centre de masse de l'objet
9 6 5 6 5 9     - une préhension (point1 = (9, 6, 5), point2 = (6, 5, 9))
...
6 8 9 4 5 8
7 2 8 9 6 5
```

2 Interfaces homme/machine

Dans ce projet, l'interface homme/machine est tout simple comme un console sous Windows. Il ne nécessite pas d'une interface graphique.

L'utilisateur doit saisir l'emplacement du fichier contenant les préhensions à évaluer. Puis le système va afficher les 4 métriques calculés et ensuite afficher la meilleure préhension dans le fichier d'entrée.



```

D:\prehension\dist\main\main.exe
*****
Import the data file: D:\prehension\Data\data.txt
File D:\prehension\Data\data.txt imported.
*****
4 quality metrics of each grasp
[0.10688145 0.36      0.08764323 0.26478538]
[0.10688145 0.36      0.08764323 0.26478538]
[0.10688145 0.36      0.08764323 0.26478538]
[0.10688145 0.36      0.08764323 0.26478538]
[1.      0.36 1.      1.      ]
[1.      1.      0.70801003 1.      ]
[0.3364236 1.      0.21619052 0.5840998 ]
[0. 0. 0. 0.]
*****
Score: 0.819 0.819 0.819 0.819 3.36 3.708 2.137 0.0
best grasp score and index (start by 0):
3.708 5
Press Enter to continue...

```

Figure B.1 – Interface du logiciel



Cahier de Spécifications

1 spécifications Fonctionnelles

Fonctionnalités à développer

Les différentes fonctionnalités du système :

- Prendre en entrée les préhensions et le centre de masse de l'objet
- Modéliser la préhension sur chaque point de contact
- Calculer la matrice de préhension partiel
- Calculer la matrice de préhension complète
- Calculer les valeurs singulières de la matrice de préhension
- Calculer la métrique d'évaluation Q_{MVS}
- Calculer la métrique d'évaluation Q_{VEM}
- Calculer la métrique d'évaluation Q_{IIP}
- Calculer la métrique d'évaluation Q_{DCC}
- Trouver la meilleure préhension et afficher

1.1 Définition de la fonction 1 : lire le fichier de données

Description de la fonction 1 :

Default constructor of the class LoadFile

:param str file_path : the path of the input data file

:returns : a LoadFile Object that contains all the Grasp objects in the data file

:rtype : LoadFile

:raises OSError : Fail to open/read file

:raises ValueError : Fail to transform string to int/float

:raises InputFormatError : Wrong format of the file

Élément 1

Entrée : un fichier .txt

Sortie : un objet LoadFile

Préconditions : le fichier existe dans le chemin indiqué et il est bien formaté

Postconditions : le nombre d'ensembles de points de contacts = le nombre dans la première ligne du fichier

1.2 Définition de la fonction 2 : stocker une préhension

Description de la fonction 2 :

Default constructor of the class Grasp, convert the input list object to Numpy array

:param ndarray point_1 : contact point 1

:param ndarray point_2 : contact point 2

:param ndarray center_masse : center masse of the object to grasp

:returns : a Grasp Object

:rtype : Grasp

Élément 1

Entrée : deux points de contact et un centre de masse

Sortie : un objet Grasp

Préconditions : rien

Postconditions : rien

1.3 Définition de la fonction 3 : calculer le vecteur n des deux points de contact

Description de la fonction 3 :

Calculate the vector n by calculating the vector from point_1 to point_2 and from point2 to point1

:returns : the corresponding vector n of point_1 and point_2

:rtype : ndarray

Élément 1

Entrée : rien

Sortie : le vecteur n des deux points de contact de la préhension

Préconditions : rien

Postconditions : le sens des deux vecteurs est opposé

1.4 Définition de la fonction 4 : calculer les valeurs t et o à partir d'un point et le vecteur n

Description de la fonction 4 :

Élément 1

Entrée : un point de contact et son vecteur n

Sortie : le vecteur t et o correspondant

Préconditions : rien

Postconditions : les trois vecteurs sont orthogonaux et dans le bon sens

1.5 Définition de la fonction 5 :calculer la matrice de rotation**Description de la fonction 5 :**

Calculate the rotation matrix of the two contact points

First, normalize the vectors n , t , o . Then calculate the rotation matrix.

rotation matrix = $[n.T, t.T, o.T]$

:returns : 2 rotation matrix

:rtype : ndarray

Élément 1

Entrée : rien

Sortie : la matrice de rotation des deux points de contact

Préconditions : rien

Postconditions : chaque colonne de la matrice de rotation est un vecteur normalisé

1.6 Définition de la fonction 6 :calculer la matrice S **Description de la fonction 6 :**

Calculate the S Matrix of a contact point

$r = \text{point} - \text{center_masse}$

$S(r) =$

$\begin{bmatrix} 0 & -r_z & r_y \end{bmatrix}$

$\begin{bmatrix} r_z & 0 & -r_x \end{bmatrix}$

$\begin{bmatrix} -r_y & r_x & 0 \end{bmatrix}$

:param ndarray point : contact point

:returns : the S matrix of the contact point

:rtype : ndarray

Élément 1

Entrée : un point

Sortie : la matrice S du vecteur correspondant

Préconditions : rien

Postconditions : rien

1.7 Définition de la fonction 7 :calculer la matrice de préhension**Description de la fonction 7 :**

Calculate the Grasp Matrix

$G(\text{point}) =$

$\begin{bmatrix} R(\text{point}), 0 \end{bmatrix}$

$\begin{bmatrix} S(\text{point} - \text{cm}) * R(\text{point}), R(\text{point}) \end{bmatrix}$

$G = [G(\text{point1}), G(\text{point2})]$

:returns : the Grasp matrix of the current grasp

:rtype : ndarray

Élément 1

Entrée : rien
 Sortie : la matrice de préhension complète
 Préconditions : rien
 Postconditions : rien

1.8**Définition de la fonction 8 :calculer les valeurs singulières de la matrice de préhension****Description de la fonction 8 :**

Calculate the single values of the Grasp Matrix
 stock the results in the attribut self.__single_values
 :returns : the single values of the Grasp Matrix
 :rtype : ndarray

Élément 1

Entrée : rien
 Sortie : une liste des valeurs singulières
 Préconditions : rien
 Postconditions : rien

1.9**Définition de la fonction 9 :calculer Q_{MVS}** **Description de la fonction 9 :**

Calculate the quality metric Q_MVS
 Q_MVS = the minimum of the single values
 :returns : the quality metric Q_MVS
 :rtype : float

Élément 1

Entrée : rien
 Sortie : Q_{MVS}
 Préconditions : rien
 Postconditions : $Q_{MVS} = \min(s)$

1.10**Définition de la fonction 10 :calculer Q_{VEM}** **Description de la fonction 10 :**

Calculate the quality metric Q_VEM
 $Q_VEM = s1*s2*s3...*sn$
 :returns : the quality metric Q_VEM
 :rtype : float

Élément 1

Entrée : rien
 Sortie : Q_{VEM}
 Préconditions : rien
 Postconditions : $Q_{VEM} = s1*s2*s3...*sn$

1.11 Définition de la fonction 11 :calculer Q_{IIP} **Description de la fonction 11 :**

Calculate the quality metric Q_{IIP}

$Q_{IIP} = \min(s)/\max(s)$

:returns : the quality metric Q_{IIP}

:rtype : float

Élément 1

Entrée : rien

Sortie : Q_{IIP}

Préconditions : rien

Postconditions : $Q_{IIP} = \min(s)/\max(s)$

1.12 Définition de la fonction 12 :calculer Q_{DCC} **Description de la fonction 12 :**

Calculate the quality metric Q_{DCC}

Q_{DCC} = the distance between the center of the vector P1P2 and the center masse of the object

:returns : the quality metric Q_{DCC}

:rtype : float

Élément 1

Entrée : rien

Sortie : Q_{DCC}

Préconditions : rien

Postconditions : Q_{DCC} = the distance between the center of the vector P1P2 and the center masse of the object

2 Spécifications non fonctionnelles**2.1 Contraintes de développement et conception**

Pour développer ce projet, nous avons décidé de programmer en Python. Nous utiliserons un ordinateur dont le OS est Windows et un IDE (PyCharme) a été installé. De même, nous utiliserons GitLab afin de s'échanger et enregistrer les différents fichiers.

2.2 Contraintes de fonctionnement et d'exploitation**2.2.1 Performances**

Concernant les calculs matriciels, nous ferons en sorte que ce soit quasi vite. Cela n'étant (que) du code, il faudra faire en sorte de ne pas programmer de sorte à ce que le calcul de la matrice de préhension et ses valeurs singulières prenne trop de temps.

Nous pouvons partir du principe qu'au-delà d'environ 2 minutes pour charger les résultats d'un fichier de préhensions, le calcul sera abandonné et un message d'erreur s'affichera.

2.2.2 Capacités

L'évaluation fait une partie de la préhension d'objet. Nous souhaitons que le nombre de préhensions à évaluer ne soit pas beaucoup (moins de 100 idéalement). Pourtant, la capacité d'analyse dépend de la capacité de l'ordinateur utilisé.

2.2.3 Contrôlabilité

Pour la traçabilité du système, nous pouvons enregistrer le résultat d'un fichier d'entrée dans un autre fichier si l'utilisateur le souhaite.

2.2.4 Sécurité

Il n'y a pas de besoin sur la sécurité dans ce projet.

D

Guide d'installation des librairies

Dans ce projet, le calcul des métriques de préhension implique un certain nombre d'opérations matricielles. Dans l'environnement python, numpy fournit un certain nombre de fonctions qui facilitent les opérations matricielles. En tant que tel, numpy est l'une des principales librairies utilisées pendant la phase de développement de ce projet. Dans la phase de test, des tests unitaires ont été effectués en utilisant le module « unittest ». Pour mieux maintenir le projet, Sphinx a été utilisé pour générer une documentation intuitive. En résumé, les librairies suivantes ont été utilisées dans ce projet :

1. numpy : pour faire des opérations matricielles
 - Commande d'installation :
pip install numpy
2. Sphinx : pour générer une documentation formatée en HTML
 - Commande d'installation :
pip install -U sphinx
 - Commande d'utilisation :
sphinx-build source/ build/ (sous le dossier « Docs »)
3. HtmlTestRunner : pour visualiser le résultat des tests unitaires en HTML
 - Commande d'installation :
pip install html-testRunner

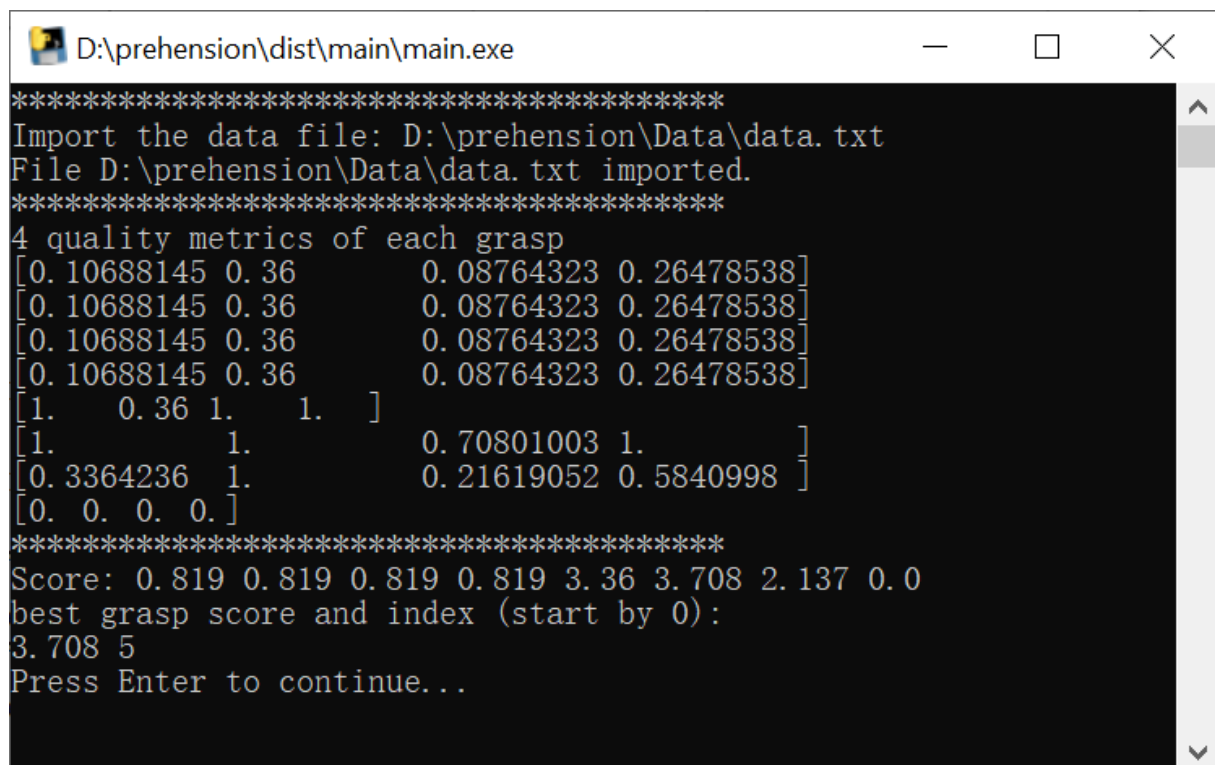
Pour s'assurer que le programme s'exécute correctement, il faut ajouter le chemin du projet dans la variable d'environnement « PYTHONPATH » pour le système Windows.

E

Guide d'utilisation

Pour faciliter l'utilisation de l'outil, une interface de console a été générée à l'aide de « pyinstaller ». Quand l'utilisateur utilise ce logiciel, il saisit l'emplacement du fichier d'entrée. Puis le logiciel s'assure que le fichier respecte absolument à son format standard. Le format standard du fichier d'entrée est décrit dans la rubrique « Interfaces matérielles/logicielles ».

Une fois le calcul terminé, le logiciel sort les valeurs des quatre métriques pour chaque préhension et renvoie un score relatif pour chaque préhension, indiquant à l'utilisateur celui qui a obtenu le meilleur score et son indice correspondant.



```
*****
Import the data file: D:\prehension\Data\data.txt
File D:\prehension\Data\data.txt imported.
*****
4 quality metrics of each grasp
[0.10688145 0.36      0.08764323 0.26478538]
[0.10688145 0.36      0.08764323 0.26478538]
[0.10688145 0.36      0.08764323 0.26478538]
[0.10688145 0.36      0.08764323 0.26478538]
[1.      0.36 1.      1.      ]
[1.      1.      0.70801003 1.      ]
[0.3364236 1.      0.21619052 0.5840998 ]
[0. 0. 0. 0.]
*****
Score: 0.819 0.819 0.819 0.819 3.36 3.708 2.137 0.0
best grasp score and index (start by 0):
3.708 5
Press Enter to continue...
```

Figure E.1 – Interface du logiciel

1 Diagramme de classe

Classe `Error`

Classe de base pour les exceptions dans le module

Classe `InputFormatError`

Exception à lever pour les erreurs dans le format d'entrée

Classe `LoadFile`

Classe pour lire les données dans une liste de type Numpy. Elle contient un attribut pour stocker des objets Grasp.

Classe `Grasp`

Classe pour calculer la matrice de préhension et les métriques de préhension. Un objet Grasp est défini par deux points et le centre de masse de l'objet.

2 Structure des fichiers utilisés

Dans ce projet, nous avons principalement 8 dossiers et un fichier.

- `build`, `dist` : les deux dossiers sont générés par `pyinstaller`, l'exécutable se trouve dans le dossier `dist`
- `Data` : ce dossier contient des fichiers de données d'entrée
- `Docs` : ce dossier contient la documentation générée par Sphinx
- `Error` : ce dossier contient la classe `Exception`
- `Grasp` : ce dossier contient la classe `Grasp`
- `LoadFile` : ce dossier contient la classe `LoadFile`
- `Test` : ce dossier contient les tests unitaires et le rapport de tests
- `main.py` : ce fichier contient l'interface console du programme

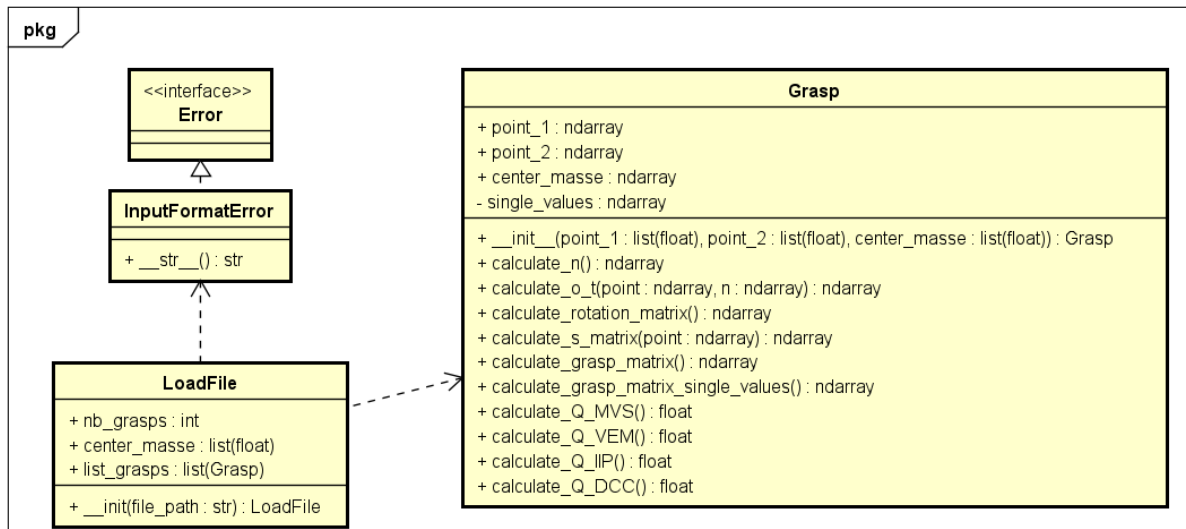


Figure F.1 – Diagramme de classe

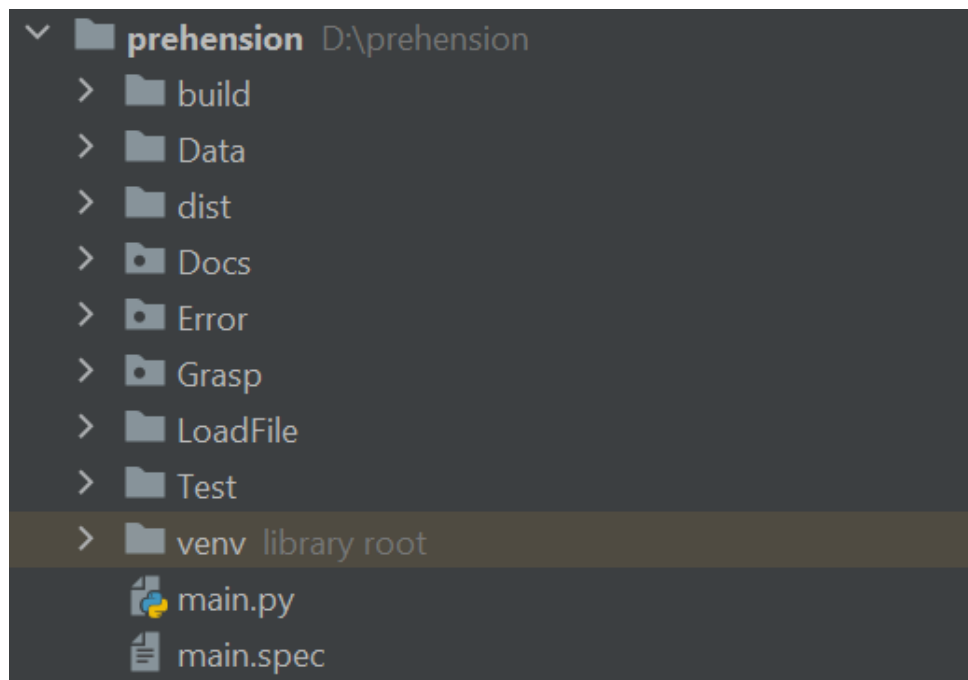


Figure F.2 – La structure du projet

1 Test avec unittest

Les tests réalisés par unittest sont comme Figure G.1. Pour faire les tests, on utilise un fichier de test suite.

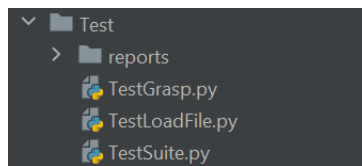


Figure G.1 – tests unitaires

2 Tests unitaires de la classe LoadFile

IDENTIFICATION OF COMPONENT
Créer correctement un objet LoadFile à partir d'un fichier d'entrée
DESCRIPTION OF THE TEST (granularity, scenario, values, actions)
Action : créer un objet LoadFile avec le fichier dans le dossier Doc/test.txt vérifier les attributs de l'objet créé vérifier le nombre de préhensions
EXPECTED RESULTS
True True True
OBTAINED RESULTS
True True True

3 Tests unitaires de la classe Grasp

Pour chaque fonction de la classe Grasp, nous avons implémenté des tests unitaires pour vérifier que le résultat du calcul est correct. Pour plus de détails, il suffit de voir le rapport de tests généré comme ci-dessous.

Unittest Results

Start Time: 2022-03-31 14:29:22

Duration: 27 ms

Summary: Total: 11, Pass: 11

TestLoadFile.TestLoadFile	Status
test_init	Pass
Total: 1, Pass: 1 -- Duration: 0 ms	
TestGrasp.TestGrasp	Status
test_calculate_Q_DCC	Pass
test_calculate_Q_IIP	Pass
test_calculate_Q_MVS	Pass
test_calculate_Q_VEM	Pass
test_calculate_grasp_matrix	Pass
test_calculate_n	Pass
test_calculate_o_t	Pass
test_calculate_rotation_matrix	Pass
test_calculate_s_matrix	Pass
test_init	Pass
Total: 10, Pass: 10 -- Duration: 27 ms	

Figure G.2 – *rapport de tests*

- [1] Beatriz LEÓN, Antonio MORALES et Joaquin SANCHO-BRU. *From Robot to Human Grasping Simulation*. T. 19. Cognitive Systems Monographs. Springer, 2014. ISBN : 978-3-319-01832-4. DOI : [10.1007/978-3-319-01833-1](https://doi.org/10.1007/978-3-319-01833-1). URL : <https://doi.org/10.1007/978-3-319-01833-1>.
- [2] Richard M. MURRAY, Zexiang LI et Shankar SASTRY. *A mathematical introduction to robotics manipulation*. CRC Press, 1994. ISBN : 978-0-8493-7981-9.
- [3] Domenico PRATTICHIZZO et Jeffrey C. TRINKLE. « Grasping ». In : *Springer Handbook of Robotics*. Sous la dir. de Bruno SICILIANO et Oussama KHATIB. Springer Handbooks. Springer, 2016, p. 955-988. DOI : [10.1007/978-3-319-32552-1_38](https://doi.org/10.1007/978-3-319-32552-1_38). URL : https://doi.org/10.1007/978-3-319-32552-1_38.
- [4] Máximo A. ROA et Raúl SUÁREZ. « Grasp quality measures : review and performance ». In : *Auton. Robots* 38.1 (2015), p. 65-88. DOI : [10.1007/s10514-014-9402-3](https://doi.org/10.1007/s10514-014-9402-3). URL : <https://doi.org/10.1007/s10514-014-9402-3>.
- [5] Carlos RUBERT, Beatriz LEÓN, Antonio MORALES et Joaquin SANCHO-BRU. « Characterisation of Grasp Quality Metrics ». In : *J. Intell. Robotic Syst.* 89.3-4 (2018), p. 319-342. DOI : [10.1007/s10846-017-0562-1](https://doi.org/10.1007/s10846-017-0562-1). URL : <https://doi.org/10.1007/s10846-017-0562-1>.

Apprentissage de la préhension d'objets à l'aide d'un bras manipulateur

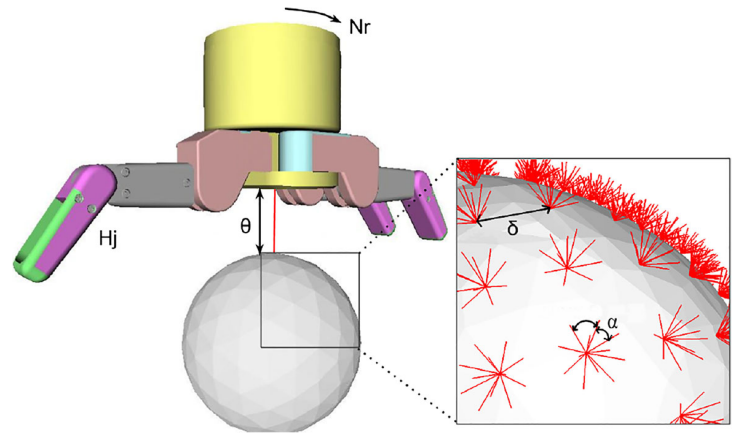
Hanlu HU

Encadrement : Pierre GAUCHER et Nicolas MONMARCHE

Objectifs

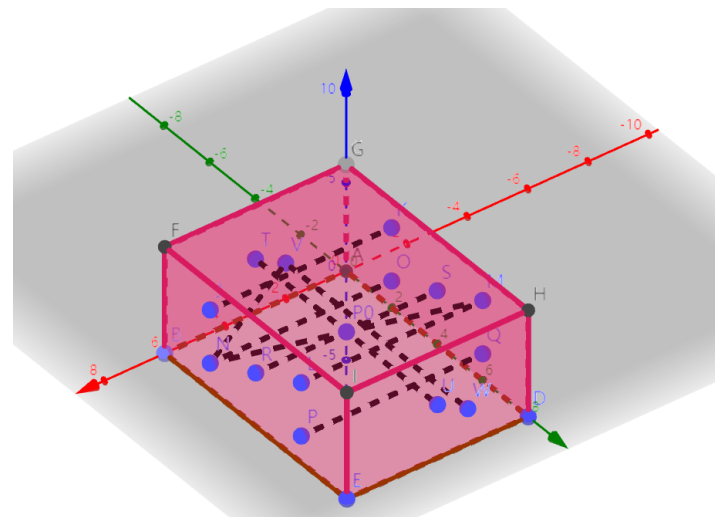
Répondre aux questions relatives à la sélection des préhensions

- Comment construire un modèle pour évaluer la qualité de préhensions
- Comment sélectionner des bonne métriques de qualité à partir du modèle
- Comment trouver la meilleure parmi un ensemble de préhensions



Mise en œuvre

1. Modéliser une préhension
2. Calculer la matrice de préhension
3. Comparer les préhensions d'entrée



Résultats attendus

L'utilisateur saisit un fichier de données et notre modèle permet de calculer et renvoyer le score de chaque préhension, ainsi la meilleure.



LABORATOIRE D'INFORMATIQUE FONDAMENTALE ET APPLIQUÉE DE TOURS

Apprentissage de la préhension d'objets à l'aide d'un bras manipulateur

Hanlu HU

Encadrement : Pierre GAUCHER et Nicolas MONMARCHÉ

Objectifs

Répondre aux questions relatives à la sélection des préhensions

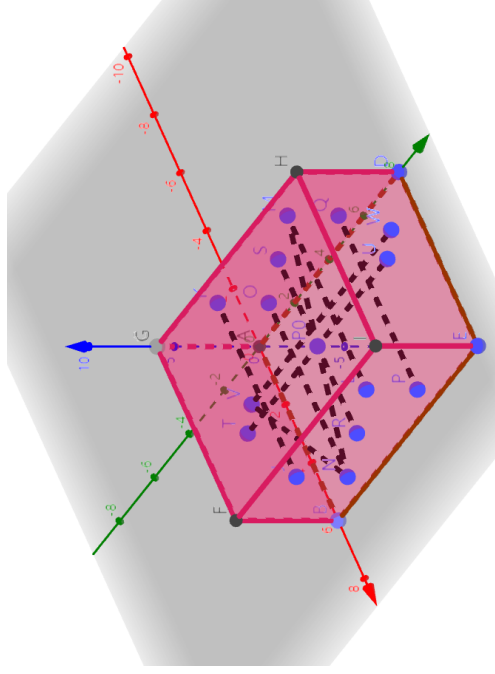
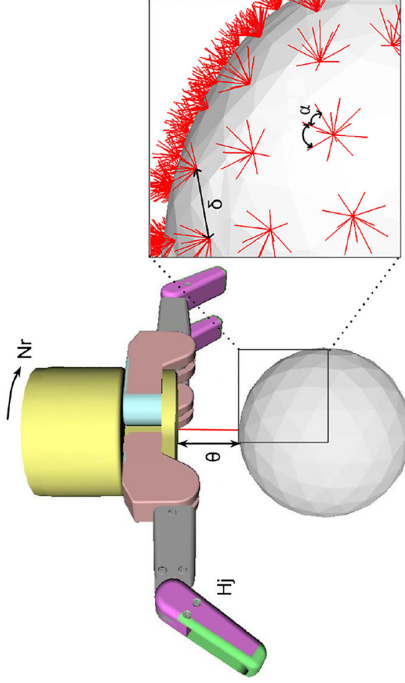
- Comment construire un modèle pour évaluer la qualité de préhensions
- Comment sélectionner des bonne méthodes de qualité à partir du modèle
- Comment trouver la meilleure parmi un ensemble de préhensions

Mise en œuvre

1. Modéliser une préhension
2. Calculer la matrice de préhension
3. Comparer les préhensions d'entrée

Résultats attendus

L'utilisateur saisit un fichier de données et notre modèle permet de calculer et renvoyer le score de chaque préhension, ainsi la meilleure.



LABORATOIRE D'INFORMATIQUE FONDAMENTALE ET APPLIQUÉE DE TOURS

Apprentissage de la préhension d'objets à l'aide d'un bras manipulateur

Résumé

Le sujet de Projet de Recherche et Développement est "Apprentissage de la préhension d'objets à l'aide d'un bras manipulateur". Ce projet est plutôt sur la partie modélisation et recherche. Il s'agit de résoudre un problème d'évaluation de préhensions. Sur la base modèle de préhension existant, nous calculons la matrice de préhension pour chaque point de contact, puis nous obtenons la matrice de préhension complète. En analysant les valeurs singulières de la matrice de préhension, nous pouvons éventuellement obtenir les métriques qui sont importants pour l'évaluation de la préhension et nous permettent de trouver la meilleure préhension.

Mots-clés

préhension, évaluation, matrice de préhension, bras manipulateur, robot, manipulator arm, robot

Abstract

The subject of the Research and Development Project is "Study on the grasps of objects using a manipulator arm". This project concentrates rather on the modeling and research part. It is about solving a problem of the evaluation of grasps. On the basis of existing grasp model, we calculate the grasp matrix for each contact point, then we obtain the complete grasp matrix. By analyzing the singular values of the grasp matrix, we can eventually obtain the metrics that are important for the grasp evaluation and allow us to find the best grasp.

Keywords

grasp, evaluation, grasp matrix

Tuteurs académiques

Pierre GAUCHER

Nicolas MONMARCHE

Étudiant

Hanlu HU (DI5)