

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene



Faculté de Génie Electrique

Département d'Automatique

Domaine Sciences et Technologie
Filière : Automatique

Mémoire de Master

Spécialité : Automatique et systèmes
Thème

Optimisation pour l'interaction homme robot industriel

Encadré par : Mr. GAHAM Mehdi

Présenté par : BENKHEDDA Abderrahim

TOUATI Mohamed Akram

Soutenu le : 26/06/2022, devant le jury composé par :

BENRABAH Mohamed

BENMRAR Tarik Zakaria

GAHAM Mehdi

Daoudi Abdelghani

Président

Examineur

Encadrant

Co-encadrant

Promotion : Juin 2022

ملخص

يشتمل المشروع على تطوير نموذج تحسين لنزاع التحكم للروبوت التعاوني من نوع كوكا، الذي أصبح يحل مكان الروبوتات الصناعية بسبب ميزاته المختلفة، بالإضافة إلى تأثيره على التقنيات الجديدة 1 في الصناعة 0.4. اخترنا تطوير تطبيق تحسين وقت الدورة و كذلك الطاقة التي تستهلكها الآلة من خلال استخدام نهج التحسين بواسطة خوارزمية جينية من الفئة التطورية يسمح تطوير هذا النموذج بحساب أفضل حل لتغطية مجموعة من النقاط بأقل وقت ممكن يستغرق الحد الأدنى من الطاقة، و هذا سيسمح للصناعيين بزيادة معدل الإنتاج مع تقليل السعر الساعي و الطاقوي.

Résumé

Notre projet consiste au développement d'un modèle d'optimisation pour le bras manipulateur du robot collaboratif « KUKA IIWA r800 ». Ce type de robot est entrain de remplacer les robots industriels pour ses différentes caractéristiques et son impact sur les nouvelles technologies de l'industrie 4.0. En utilisant une approche d'optimisation par un algorithme métaheuristique de classe évolutionnaire, nous avons opté pour le développement d'une application d'optimisation du temps de cycle et l'énergie consommée par un algorithme génétique. La programmation est centrée sur la recherche du meilleur chemin pour le robot, et déterminer deux vecteurs accélération et vitesse qui agiront sur la valeurs maximal de ces derniers. L'implémentation de ce modèle permet de calculer la meilleur solution pour parcourir un ensemble de points avec le moins de temps possible en consommant une énergie minimale, ce qui permettra aux industrielles d'augmenter le taux de production tout en diminuant les coûts temporelles et énergétique.

Abstract

Our project aims to develop an optimization model for a robotic arm of the collaborative robot « KUKA IIWA r800 ». This type of robot is replacing industrial robots for its advantageous characteristics over the latter as well as its impact on new technologies in Industry 4.0. Through the use of an optimization approach by an evolutionary class meta-heuristic algorithm. We opted for developing an application for optimizing the cycle time as well as the energy consumed by implementing a genetic algorithm. The optimization is based on the search for the best path for the robot, and the determination of an acceleration and velocity vectors that will act as factors on the maximum values of the latter. The implementation of this model allows the calculation of the best solution to cover a set of points with the least amount of time possible while consuming a minimum amount of time and energy, which will allow the industrialists to increase the production rate while decreasing their temporal and energy expenses

Dédicaces

*Je dédie ce travail à :
Mes parents, que dieu me les protègent,
mes frères et sœurs que dieu me les garde,
Et à tous ceux que j'aime.*

Benkhedda Abderrahim

*Je dédie ce travail à mes chers Parents,
à ma sœur et mon frère que dieu me les garde,
A tous mes amis et a tous qui m'ont soutenu.*

Touati Mohamed AKRAM

Remerciements

En premier lieu, on remercie Dieu, le tout puissant, pour nous avoir donné, la patience, la volonté et la force nécessaires pour achever ce travail.

*Nous tenons à exprimer toute notre Gratitude à **Mr GAHAM Mehdi** d'avoir accepté d'encadrer notre thèse. Merci pour vos conseils votre disponibilité et soutien dans les moments délicats.*

On remercie le président pour avoir accepté de juger ce travail, et les membres du jury qui ont pris de leurs temps pour lire, juger ce travail.

On tient à remercier nos famille que nous aimons énormément et qu'on remercie chaleureusement du fond du cœur, de nous avoir supporté.

Enfin, on tient à remercier tous ceux que on n'a pas cités mais que on n'a pas oubliés pour autant et qui de près comme de loin nous ont aidé, soutenu et encourager aux moments opportuns.

A tous, un grand MERCI!

Liste des notations et abréviations

CDTA :Centre de Développement des Technologies Avancé

DPR :devision productique robotique

Vr : virtual reality

Iot :internet of things

RA :Realite augmenté

3D :3 Dimensional

CPS : Cyber Physical system

AMR : Autonomous Mobile Robot

AGV : Automated guided vehicle

Mcd Model cinimatique direct

TSP :Traveling Selsman Problem

EC: Evolutionary Computation

CoEA : CoEvolutionary Algorithms

DE : Differential Evolution

EDA : Estimation of Distribution Algorithm

GA : Genetic Algorithms

PSO : Particle Swarm Optimization

ACO : Ant Colony Optimization

Sommaire

Introduction général.....	1
Chapitre I : Généralités et état de l'art	
I.1 Introduction	3
I.2 l'industrie 4.0.....	3
I.2.1 Définition.....	3
I.2.3 Historique.....	3
I.2.3.1 1-ère révolutions industrielle	3
I.2.3.2 2 -ème révolutions industrielle.....	4
I.2.3.3 3-ème révolutions industrielle	4
I.3 Les technologies de l'industrie 4.0	4
I.3.1 L'intelligence artificielle.....	4
I.3.2 Blockchain	5
I.3.3 réalité virtuelle et réalité augmentée	5
I.3.4 L'impression 3D.....	5
I.3.5 Internet of things (IOT).....	6
I.3.6 Le modèle numérique / jumeau numérique.....	6
I.3.7 Les systèmes cyber-physique	6
I.4 Les robots industriel.....	6
I.4.1 Définition	6
I.4.2 Type des robots industriels.....	7
I.4.2.1 Robots mobiles autonomes (AMR).....	7
I.4.2.2 Véhicules à guidage automatique (AGV).....	7
I.4.2.3 Robots articulés (bras robotique).....	7
I.4.2.4 Robot collaboratif (cobot)	8
I.4.2.5 Robot hybride.....	9
I.5 Avantages de l'industrie 4.0.....	9
I.5.1 Automatisation de la gestion logistique.....	9
I.5.2 Géolocalisation des pièces pour une traçabilité optimale	10
I.5.3 Renforcement du lien avec les clients	10
I.5.4 La maintenance prédictive.....	10
I.5.5 L'optimisation de la consommation de matières premières et d'énergie.....	10
I.6 État De l'Art :	11

I.6.1	Optimisation Robotique	12
I.6.2	Optimisation Des Performances Robotique.....	12
I.6.2.1	Optimisation de la Position du Point de Base et Trajectoire	12
I.6.2.2	Optimisation de La Consommation d'Énergie	14
I.6.3	Planification des Trajectoire.....	15
I.7	Plateforme CDTA	16
I.7.1.	Présentation CDTA	16
I.7.2	Divisons DPR:.....	16
I.8	Conclusion.....	18

Chapitre II : Optimisation des tâches robotique

II.1	Introduction	19
II.2	Approche Existante	19
II.2.1	Présentation	19
II.2.2	Problématique	19
II.2.3	Solution Proposé.....	19
II.2.4	Les sous tâches	20
II.2.5	Aspect Fonctionnel de Système d'Apprentissage/Commande.....	20
II.2.5.1	Application d'Apprentissage Et De Commande Robotique.....	20
II.2.5.2	Application de commande/d'exécution	21
II.2.6.	Problème Posé	21
II.3	Problématique	22
II.4	Algorithmes Métaheuristique.....	23
II.4.1	définition et concept des algorithmes méta-heuristique	23
II.4.2	Classification des algorithmes métaheuristique.....	25
II.4.2.1	Les métaheuristiques à solution unique.....	25
II.4.2.2	Les métaheuristiques à population de solutions	26
II.5	Algorithmes Génétique.....	28
II.5.1	Présentation l'algorithme génétique	28
II.5.1.1	La création de la population initiale	28
II.5.1.2	L'évaluation des individus	29
II.5.1.3	La production de nouveaux individus	29
II.5.1.4	L'insertion des nouveaux individus dans la population.....	31
II.5.1.5	Critères d'arrêt.....	32
II.6	approche utilise dans notre travail	32

II.6.1	La création de la population initiale	33
II.6.2	L'évaluation des individus	33
II.6.3	La création de nouveaux individus.....	33
II.6.3.1	La sélection	33
II.6.3.2	croisement.....	33
II.6.3.3	mutation	33
II.6.4	le critère d'arrêt.....	34
II.7	conclusion.....	34

Chapitre III : Réalisation et implémentation

III.1.	Introduction :.....	35
III.2.	Environnement de Programmation :.....	35
III.3.	Programmation Python :.....	35
III.3.1	Outils De Programmation :.....	35
III.3.1.1	Spyder:	35
III.3.1.2.	Connexion API :.....	36
III.3.1.3	Les Classes :.....	36
III.3.1.4	Libraires :	36
III.3.2.	Connexion API Python/ CoppeliaSim :.....	37
III.3.2.1	Création De La Connexion API :.....	38
III.3.2.2	Test De Fonctionnement :.....	39
III.3.3.	Programmation Algorithme Génétique :	39
III.3.3.1.	Début de Simulation :.....	40
III.3.3.2.	Initialisation du code GA :.....	41
III.3.3.3.	Génération de Population initial :.....	42
III.3.3.4.	Fitness :	43
III.3.3.5.	Sélection	45
III.3.3.6.	Permutation :.....	46
III.3.3.7	Mutation :	47
III.3.3.8	Prochaine Génération :.....	49
III.3.3.9.	Evolution De l'Algorithme Génétique :.....	49
III.4	CoppeliaSim	50
III.4.1	Caractéristiques principales de CoppeliaSim	51
III.4.2	simulation de notre travail avec CoppeliaSim.....	52
III.4.3	explication du code Lua	53

III.5 Conclusion.....	53
-----------------------	----

Chapitre IV : Résultats et interprétation

IV.1. Introduction.....	54
IV.2. Présentation et interprétations des résultats	54
IV.2.1. Validation du Modèle	54
IV.2.2 Résultats obtenue avec un vecteur points fixe.....	56
IV.2.2.1 Ensemble de 7 points.....	56
IV.2.2.2 Ensemble de 15 points.....	57
IV.2.2.3 Ensemble de 20 points.....	59
IV.2.3. Impact des coefficient de pondérations	60
IV.3. Conclusion	63
Conclusion générale.....	64

Liste des figures

Figure I. 1.	Schéma des révolution industrielles.....	4
Figure I. 2.	Robot mobile autonome (AMR)	7
Figure I. 3.	Robot articulé.....	8
Figure I. 4.	robot collaboratif.....	9
Figure I. 5.	Robot hybride.....	9
Figure I. 6.	Aperçu sur la localisation du point de base du robot	13
Figure I. 7.	Le schéma bloc de l'algorithme appliqué pour trouver la meilleure solution de la cinématique inverse.....	13
Figure I. 8.	Aperçu du système Développé.....	15
Figure I. 9.	Aperçu général sur la division DPR	17
Figure II. 1.	Schéma de dialogue pour l'apprentissage des points	21
Figure II. 2.	les 3 phases d'un algorithmes métaheuristique.....	25
Figure II. 3.	classes des algorithmes métaheuristique	27
Figure II. 4.	exemple de sélection.....	30
Figure II. 5.	exemple de croisement.....	31
Figure II. 6.	exemple de mutation.....	31
Figure II. 7.	schéma explicatif de l'algorithme génétique	32
Figure III. 1.	Importation des librairies de connexion API.....	38
Figure III. 2.	Script Connexion API	38
Figure III. 5.	Connexion API établie	Erreur ! Signet non défini.
Figure III. 6.	Script Final_KUKA	41
Figure III. 7.	Initialisation des vecteurs	42
Figure III. 8.	Création d'individu	43
Figure III. 9.	Génération d'une population de taille n.....	43
Figure III. 10.	Classe fitness	44
Figure III. 11.	Classement des Individus (identifiant d'individu,score fitness).....	45
Figure III. 12.	Création de groupe d'accouplement	45
Figure III. 13.	Code de Permutation	46
Figure III. 14.	permutation vecteur vitesse et accélération.....	47
Figure III. 15.	Mutation du vecteur points	48
Figure III. 16.	Mutation vecteur vitesse et accélération	48
Figure III. 17.	Mutation de toute la population.....	49
Figure III. 18.	Script de l'évolution.....	49
Figure III. 19.	Script de fonction d'exécution	50
Figure III. 20.	Vue du simulateur CoppeliaSim.....	52
Figure IV. 1.	Evolution de Fitness pour 7 points aléatoires	55
Figure IV. 2.	Evolution de Fitness pour 10 points aléatoires	55

Figure IV. 3.	Evolution de Fitness pour 15 points aléatoires	55
Figure IV. 4.	Evolution de Fitness pour 20 points aléatoires	55
Figure IV. 5.	Evolution de l'énergie avec vecteur point fixe pour 7 points.....	56
Figure IV. 6.	Evolution du temps avec vecteur point fixe pour 7 points.....	57
Figure IV. 7.	Evolution de l'énergie avec vecteur point fixe pour 15 points.....	58
Figure IV. 8.	Evolution du temps avec vecteur point fixe pour 15 points.....	58
Figure IV. 9.	Evolution de l'énergie avec vecteur point fixe pour 20 points.....	59
Figure IV. 10.	Evolution du temps avec vecteur point fixe pour 20 points.....	60

Liste des tableaux

Tableau IV. 1.	impacte des facteurs de pondération	61
-----------------------	---	----

Introduction générale

Introduction général

Dans la poursuite du développement technologique croissant et l'amélioration de production, le monde assiste à des avancées technologiques sans précédent dans l'industrie et la robotique. En effet, de nombreuses entreprises tentent d'améliorer le rendement quotidien en accélérant les processus de production dans le but de maximiser les profits. La robotique reste un des domaines les plus convoités qui attire la curiosité de nombreux professionnels et entreprises influents sur les secteurs de l'industrie, l'automobile, l'aérospatiale et même la médecine... Cette technologie révolutionnaire contribue à la réduction des risques, pertes, dangers et participe à l'amélioration la précision d'exécution des tâches.

Suite à la forte demande et consommation de l'énergie dans l'industrie 4.0, de nombreux pays industriels ont connu une augmentation des prix de l'électricité et des carburants au cours de la dernière décennie. Selon des statistiques récentes, l'industrie manufacturière est l'un des plus gros consommateurs d'énergie. La majorité de l'énergie est généralement consommée par les robots utilisés dans les différents secteurs industriels. En outre, au cours de la phase de production d'un produit, l'utilisation optimale de l'énergie et du temps de cycle par les robots joue un rôle important dans la réduction des émissions de CO₂.

Les robots industriels sont souvent perçus comme des machines non-durables, qui nécessitent un niveau élevé de consommation d'énergie. Par ailleurs, ces robots offrent une précision, une force et des capacités de détection qui peuvent générer des produits finis de haute qualité. Par conséquent, pour de nombreux organismes d'étude et producteurs de robots, la consommation énergétique est devenue un objectif important. La réduction et l'optimisation de la consommation de l'énergie et du temps d'exécution des tâches des robots sont donc devenues un objectif très important pour de nombreux organismes d'étude et producteurs de robots.

Le robot manipulateur collaboratif est très souvent utilisé dans les applications de l'assemblage, soudage et peinture. Pour que le bras manipulateur puisse effectuer des tâches de manière autonome, les ingénieurs et les techniciens ont pensé à équiper le bras manipulateur d'un programme qui nécessite une certaine expérience pour permettre l'apprentissage des points et le contrôle du robot. Pour cela, il faut un langage d'échange simple avec l'utilisateur.

Un système d'apprentissage/commande du robot KUKA a été conçu par Dine et Mechouar et encadré par M. Gaham, ce travail sert à faire une commande simple et conviviale, sans arrêt du robot ni utilisation de son interface habituelle de programmation, même les utilisateurs disposant de faibles connaissances techniques peuvent exécuter des tâches à l'aide

de ce système. Mais l'application de ce système ne couvre pas le volet optimisation des performances et effectue des tâches qui parfois consomment beaucoup d'énergie. Ce problème peut se régler en introduisant un modèle d'optimisation qui complète ce travail [1].

Le travail a été développé dans la division robotique productique au centre de développement des technologies avancées CDTA, son objectif consiste à créer un système d'optimisation des performances du robot « KUKA LBR IIWA ». Ce modèle va essayer de déterminer l'ordonnancement optimal des points avec les vecteurs vitesse et accélération adéquates.

Le but est de minimiser le temps d'exécution et l'énergie dissipée, pour un ensemble de points prédéfini.

Ce travail sera subdivisé en quatre parties :

- Le premier chapitre aborde des généralités sur l'industrie 4.0, ainsi que quelques notions sur l'optimisation des robots manipulateurs.
- Le deuxième chapitre est consacré à la description de l'approche existante du problème abordé et la problématique du travail. Un aperçu sur les algorithmes d'optimisation métaheuristique avec une présentation détaillée sur les différents aspects d'optimisation en présentant l'algorithme retenu dans cette étude.
- Le troisième chapitre commence par une brève description sur les outils de programmation utilisés pour le développement d'un modèle d'optimisation robotique. Suivi par une présentation des conceptions et de la réalisation du modèle mentionné ainsi sa programmation sur les deux logiciels utilisés.
- Le quatrième chapitre sera consacré aux tests effectués par notre modèle, protocole d'expérimentation et interprétation des résultats.

Pour conclure, le travail se termine par une conclusion générale qui mettra l'accent sur les points les plus importants du travail.

CHAPITRE I

Généralités ET

État DE L'ART

I.1 Introduction

Depuis le début du 21^{ème} siècle, le monde grandit de plus en plus ce qui implique l'apparition de nouvelles contraintes, cela a poussé les industries à produire en masse dans un temps optimale tout en optimisant le coût, l'énergie sans négliger la qualité des produits. Tout cela a donné naissance à une 4^{ème} révolution industrielle, qui est capable de subvenir au besoin des industries grâce aux différentes technologies utilisé dans l'industrie 4.0, tel que les robots collaboratifs qui interagissent et coexistent avec l'homme.

Dans ce chapitre, nous allons parler des différentes révolutions industrielles avant l'apparition de l'industrie 4.0, ensuite nous allons aborder quelques technologies utilisées dans cette industrie et parler de ces avantages. Après cela, nous allons présenter l'optimisation en robotique et présenter la plateforme CDTA en général et la division DPR en particulier.

I.2 l'industrie 4.0

I.2.1 Définition

Le terme industrie 4.0 est apparu la première fois en 2011 au Forum mondial de l'Industrie de Hanovre en Allemagne, ce terme est utilisé pour décrire la quatrième révolution industrielle qui se déroule dans l'industrie manufacturière et dans les tendances en matière d'automatisation de la fabrication et d'échange de données.

L'industrie 4.0 désigne la nouvelle génération d'usines connectées, robotisées et intelligentes, elle repose sur quatre piliers fondamentaux : les systèmes cyber-physiques, l'Internet des objets, le Big Data et le Cloud Computing. Ensemble, ces technologies créent une usine connectée où les machines peuvent communiquer entre elles et prendre des décisions de manière autonome.[2]

I.2.3 Historique

Avant l'industrie 4.0 l'être humain est passé par une période de pré-industrie, apparue au moyen-âge. Durant cette période, il n'y avait pas d'usine, la production était artisanale où ils utilisaient la force des animaux et des humains. Après la période de pré-industrie, l'homme a connu 3 majeures révolutions industrielles que nous allons citer ci-dessous :

I.2.3.1 1^{ère} révolutions industrielle

Elle remonte à l'exploitation du charbon et la mise au point de la machine à vapeur par James Watt en 1769, l'artisanat est remplacé par la production mécanique, et dans les usines, la révolution correspond à l'utilisation de la machine à vapeur comme moteur pour actionner les machines ce qui entraîne une fabrication plus importante.[2]

I.2.3.2 2 -ème révolutions industrielle

La seconde repose sur l'utilisation du pétrole et de l'électricité à la fin du 19-ème siècle. Cela va permettre de moderniser les moyens de production. Les machines à vapeur sont remplacées par des machines électriques ce qui permet la production en masse de produits identiques.[2]

I.2.3.3 3-ème révolutions industrielle

Elle a eu lieu au milieu du 20-ème siècle après l'apparition de l'électronique, la télécommunication et l'informatique. Ces différentes disciplines permettront la mise en œuvre d'importantes automatisations qui déchargeront les travailleurs des tâches les plus difficiles. C'est ainsi que commence la robotique, la flexibilité des outils de production et la production à grande échelle. [2]

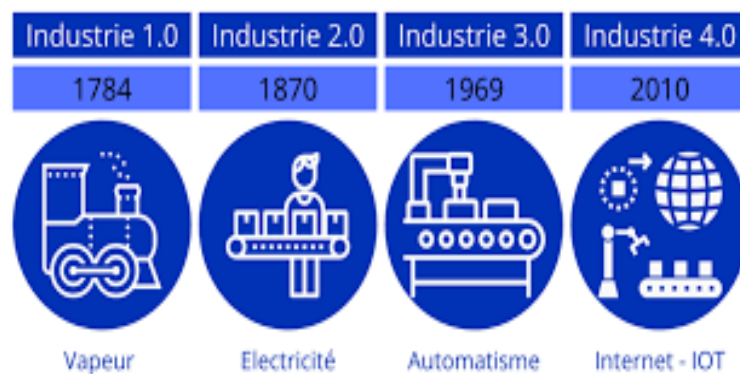


Figure I. 1 schéma des révolution industrielles.[2]

La figure I.1 nous montre les 4 révolutions industrielles qu' a connues l'homme, elle mentionne aussi l'espace temporel et les caractéristiques de chaque révolution.

I.3 Les technologies de l'industrie 4.0

L'industrie 4.0 se base sur plusieurs technologies nous allons citer les plus communes.

I.3.1 L'intelligence artificielle

L'intelligence artificielle (IA) consiste à simuler l'intelligence humaine dans des machines programmées pour penser comme les humains et imiter leurs actions. Le terme peut aussi être appliqué à n'importe quelle machine qui a des traits associés à un esprit humain comme l'apprentissage et la résolution de problèmes.[3]

Cette technologie est essentielle à l'exploitation des quantités de données issues des objets connectés. Il n'y a aucune raison de récupérer et de stocker des données si elles ne sont pas

traitées. L'intelligence artificielle est le moyen d'exploiter, en temps réel, la masse d'informations collectées, de les trier, de les analyser et de les transmettre à un opérateur ces informations aideront les techniciens à prévoir les modifications apportées aux composants de l'équipement, on parle de maintenance liée aux cellules de production.[2]

I.3.2 Blockchain

C'est une technologie de stockage et de transmission de l'information sous la forme d'une base de données qui a la particularité d'être partagé simultanément avec tous ses usagers et qui ne dépend pas d'un organe central, elle a pour avantage d'être rapide et sécurisée.[4]

Dans l'industrie 4.0, son utilisation va garantir le traçage de toute chaîne de production ou de distribution et d'autres transaction.

I.3.3 réalité virtuelle et réalité augmentée

La réalité virtuelle offre des expériences numériques immersives (à l'aide d'un casque VR) qui simulent le monde réel[5], tandis que la réalité augmentée (RA) fusionne les mondes numérique et physique et elle permet l'ajout d'informations à notre champ visuel superposant le monde réel, les images juxtaposent avec l'environnement visible.

Dans l'industrie, les employés utilisent des lunettes intelligentes ou des appareils mobiles pour visualiser en temps réel des données IoT, des pièces numérisées, des instructions de réparation ou d'assemblage, du contenu de formation, etc. lorsqu'ils regardent une chose physique, comme un équipement ou un produit.

La RA est encore en émergence, mais elle a des répercussions importantes sur l'entretien, le service et l'assurance de la qualité ainsi que sur la formation et la sécurité des techniciens[7]. Prenons l'exemple des opérations de maintenance : en pointant un équipement à l'aide d'un outil de réalité augmentée, les procédures appropriées apparaîtront dans le champ de vision de l'opérateur. Ainsi, il pourra intervenir rapidement en suivant les informations affichées.[2]

I.3.4 L'impression 3D

Elle permet aux entreprises manufacturières d'imprimer leurs propres pièces, avec moins d'outillage, à moindre coût et plus rapidement que par le biais de processus traditionnels. De plus, les conceptions peuvent être personnalisées pour assurer un ajustement parfait. Des pièces de formes complexes peuvent être réalisées à l'aide de cette technologie.[5]

Avec l'impression 3D, les pièces et les produits peuvent être stockés sous forme de fichiers de conception dans des inventaires virtuels et imprimés à la demande au moment opportun, ce qui réduit à la fois les distances et les coûts de transport.

I.3.5 Internet of things (IOT)

C'est le terme utilisé pour parler de la connexion des dispositifs à Internet, ils ont la possibilité de communiquer entre eux grâce à des capteurs dans les machines ou bien par l'usage de robots et engins autonomes, un flux d'information constant est créé, ce qui aide à ajuster la production, prévenir de possibles défauts ou détecter les stades où la production peut s'améliorer.[6]

I.3.6 Le modèle numérique / jumeau numérique

Le modèle numérique permet d'anticiper les besoins de diverses conceptions et facilite la mise en service d'un processus de production en s'appuyons sur un jumeau numérique du produit.

Le jumeau numérique est la reproduction virtuelle des pièces et machines de façon qu'on simule son fonctionnement ce qui permet d'effectuer des tests avant même la réalisation du premier prototype physique, ou de la première mise en production d'une ligne industrielle. Les différentes itérations sur le modèle numérique économisent un temps précieux en passant aux tests physiques.[2-6]

I.3.7 Les systèmes cyber-physique

un système cyber-physique (CPS) est une machine contrôlée par des calculs et des algorithmes. Ces dispositifs sont surveillés par l'ordinateur qui contrôle le processus physique. Cependant, ce qui est important, c'est la rétro alimentation, cela permet au système d'interpréter les actions, de suivre les résultats et d'apprendre automatiquement les améliorations de performances. Par exemple, les voitures autonomes, les systèmes domotiques, les infrastructures et la surveillance des routes sont des exemples de ce type de système.

En plus de ces technologies, l'industrie 4.0 utilise les cobots que nous allons parler d'eux dans les prochains titres.

Les technologies utilisées dans l'Industrie 4.0 permettent de transformer la vision d'une entreprise et de la rendre plus numérique et virtuelle. L'industrie devient intelligente et connectée par rapport au numérique. Ces technologies ne visent pas seulement à améliorer l'automatisation ,mais aussi à rendre les machines plus intelligentes en permettant à tous les systèmes de communiquer en temps réel.

I.4 Les robots industriel**I.4.1 Définition**

Selon la norme ISO 8373:2012 relative aux robots et composants robotiques, un robot industriel est défini comme un système commandé automatiquement, programmé sur trois axes

ou plus, qui peut être fixe ou mobile, destiné à être utilisé dans des applications d'automatisation industrielle.[8]

Les robots industriels sont conçus pour automatiser certaines tâches sur une chaîne de production et donc de remplacer l'être humain en apportant flexibilité et gain de productivité.

I.4.2 Type des robots industriels

Dans l'industrie, on retrouve plusieurs types de robots qui sont utilisés selon la tâche, les objectifs et l'environnement de travail.

Voici les types des robots les plus utilisés en industrie :

I.4.2.1 Robots mobiles autonomes (AMR)

Ce sont des robots qui peuvent comprendre et se déplacer dans leur environnement sans surveillance directe de l'opérateur ou être contraint à un chemin fixe prédéterminé et prennent des décisions en temps réel au fur et à mesure de leurs déplacements. Ils naviguent à l'aide des caméras, capteurs, intelligence artificielle et vision par ordinateur.[9]



Figure I. 2. Robot mobile autonome (AMR)

La figure I.2 nous montre un exemple de robot mobile autonome (AMR).

I.4.2.2 Véhicules à guidage automatique (AGV)

Contrairement au AMR qui sont libres de traverser l'environnement, les AGV reposent sur des trajectoires ou des chemins prédéfinis et nécessitent souvent la supervision d'un opérateur. Ils sont souvent utilisés pour transporter des matériaux et déplacer des articles dans des environnements contrôlés tels que des entrepôts et des usines.[9]

I.4.2.3 Robots articulés (bras robotique)

Ces robots sont censés reproduire les fonctions d'un bras humain. Ils peuvent comporter de deux à dix articulations rotatives. Chaque articulation ou axe supplémentaire permet un plus grand

degré de mouvement, ce qui en fait un outil idéal pour la soudure à l'arc, la manutention, l'entretien des machines et l'emballage.[9]

La figure I.3 nous montre un robot articulé de la marque ABB



Figure I. 3 Robot articulé

I.4.2.4 Robot collaboratif (cobot)

Les robots collaboratifs sont des robots conçus pour fonctionner aux cotes des humains ou directement avec eux et contrairement aux autres types de robots qui effectuent leurs tâches de manière indépendante ou dans des zones de travail strictement isolées, les cobots peuvent partager des espaces avec les travailleurs.[9]

Les cobots ont pour rôle d'assistant et participent à des tâches complexes et délicates qui ne peuvent pas être automatisées. Par exemple, les robots 6 axes sont maintenant capables de ramasser un objet et de le donner à un humain. Les cobots peuvent rapidement apprendre des tâches par le biais d'une démonstration et d'un apprentissage par renforcement et d'effectuer diverses tâches en toute sécurité à proximité d'un humain, grâce au progrès de l'informatique cognitive, de la technologie tactile, de la technologie mobile, de la vision artificielle.[10]

Un robot collaboratif n'est pas destiné à prendre la place d'un humain, ils prennent souvent la forme d'un bras robotisé, offrant à l'opérateur une paire de mains supplémentaire. Ils permettent ainsi de rendre les tâches moins pénibles. Les hommes peuvent ainsi s'acquitter de leur tâche avec encore plus d'efficacité, avec plus de concentration et, surtout, avec plus d'ergonomie. [10]

L'intégration des robots collaboratifs dans l'industrie a plusieurs avantages, ils permettent des gains de productivité le quotidien des opérateurs sur postes équipés amélioré, un gain en sécurité des postes de travail, un gain en confort de travail immédiat dès l'installation du cobot

une mise en œuvre rapide grâce à la facilité de programmation et les interfaces de travail simplifiées .[11]

La figure I.4 nous montre un robot collaboratif de la marque Kuka modèle lbr iwa 7



Figure I. 4. robot collaboratif

I.4.2.5 Robot hybride

Les robots hybrides sont des combinaisons des différents types de robots capables de réaliser des tâches plus complexes. Par exemple, un AMR peut être combiné avec un bras robotique pour créer un robot destiné à la manipulation des colis dans un entrepôt.

La figure I.5 nous montre un robot hybride qui est une combinaison entre un robot AMV et un bras robotique



Figure I. 5 Robot hybride

I.5 Avantages de l'industrie 4.0

La 4^{ème} révolution industrielle a plusieurs avantages, et cela est dû aux nombreuses nouvelles technologies utilisées. Parmi ces avantages , on peut citer :

I.5.1 Automatisation de la gestion logistique

Une entreprise 4.0 est une entreprise intelligente, essentiellement en raison de l'interconnectivité entre tous les maillons de la chaîne de production établie par Internet. C'est ce qu'on appelle la technologie IoT. Cette technologie permet une production plus rapide, plus efficace, moins chère et plus sûre, ainsi que la possibilité de modifier de manière autonome les méthodes de production. Ces machines soulageront l'opérateur en effectuant les tâches les plus ingrates par exemple, ce qui lui permettra de se concentrer sur des tâches à plus forte valeur ajoutée.[12]

I.5.2 Géolocalisation des pièces pour une traçabilité optimale

Grâce à l'industrie 4.0, les entreprises pourront suivre toutes les pièces dont elles ont besoin ainsi, un opérateur ou un chef d'entreprise aura la possibilité d'identifier sur son smartphone où sa tablette les trajets trop longs et trop chers, les objets perdus, limitant ainsi les pertes monétaires et les erreurs d'inventaire qui peuvent s'avérer extrêmement préjudiciables. Cela permet aux entreprises de gagner du temps, d'éviter un gaspillage d'argent et de prendre les meilleures décisions en matière d'inventaire.[12]

I.5.3 Renforcement du lien avec les clients

Grâce aux outils informatiques et des réseaux de communication modernes, les entreprises peuvent adopter une approche plus directe de la gestion des applications avec leurs clients. Par exemple, le live commerce (commerce en temps réel) est un modèle dans lequel certaines entreprises utilisent les données d'utilisation des produits pour prévoir les besoins des clients, répondre immédiatement à leurs besoins avec des produits appropriés et les partager avec des partenaires, dans le but d'offrir aux clients de nouveaux horizons et plus de valeur.[12]

I.5.4 La maintenance prédictive

Le principe de la maintenance prédictive s'agit d'analyser les performances et de détecter en avance les baisses de rentabilité et/ou le dysfonctionnement des machines et appareils. Concrètement, un logiciel étudie le cycle des machines et évalue leur espérance de vie selon les cadences imposées et s'il y a un souci, un message d'alerte est envoyé au personnel en charge de la maintenance afin qu'il fasse le nécessaire. Avoir un coup d'avance dans le but de s'éviter une perte de productivité, de temps et d'argent.[12]

I.5.5 L'optimisation de la consommation de matières premières et d'énergie

Produire plus intelligemment est l'un des enjeux majeurs de l'industrie 4.0 et c'est pourquoi l'utilisation intelligente des ressources nécessaires à l'entreprise jouera un rôle primordial.

Certaines applications permettront de disposer d'une vision globale de l'usine et de ses équipements afin d'optimiser les consommations. C'est le cas de la technologie de l'Internet des Objets.[12]

Une étude menée par Senai (société brésilienne) à l'aide des données fournies par un sondage McKinsey démontre les impacts positifs et significatifs des entreprises intelligentes :

- Réduction des coûts de la maintenance entre 10% et 40%.
- Augmentation de la productivité entre 10% et 25%.
- Précisions améliorées pour la prévision des demandes autour de 80%
- Diminution de la consommation énergétique entre 10% et 20%.

En plus de l'efficacité énergétique, élément essentiel pour les industries mondiales qui consomment 54% de l'énergie produite dans le monde, l'industrie 4.0 permet le développement de services et de nouveaux business model tout en ayant une vision des produits depuis la production jusqu'à l'utilisation finale.

I.6 État De l'Art :

Avec la quatrième révolution industrielle, l'automatisation de l'industrie ainsi que la majorité des processus industriels sans oublier l'introduction de la robotique a permis d'améliorer les performances des processus industriels en matière de rendement, durée de cycle, minimisation du coût de production tout en gardant le même but depuis la naissance de l'industrie : gagner le maximum d'argent avec le moins de coût possible.

Le déploiement des robots dans les applications industrielles a permis d'optimiser les processus industriels. Ceci a permis d'optimiser les processus industriels dans plusieurs aspects.

Les principaux avantages de la robotique pour optimiser les processus industriels sont :

1. **Précision et rapidité** : particularité d'exécuter des tâches de manière précise, tout en étant rapides.
2. **Reproductibilité des tâches** : c'est-à-dire la capacité à exécuter des tâches précises de manière répétable
3. **Réduction des délais** : les robots permettent de produire à un rythme plus élevé en réduisant les rebuts.
4. **Diminution des coûts** : un investissement plus rentable qu'un humain sur le long terme avec une précision meilleure qu'un humain.

Le déploiement des robots dans les différentes applications industrielles a donné naissance à des nouvelles notions telles que : calcul de trajectoire des robots, optimisation de

trajectoire du robot, l'optimisation robotique, optimisation des performances des machines robotisées.

I.6.1 Optimisation Robotique

L'optimisation dans la robotique traite plusieurs volets, et a permis aux spécialistes d'arriver à un très haut niveau de maîtrise des robots. Cette très haute maîtrise a permis d'avoir des résultats très avancés et personnalisés sur la manipulation et l'utilisation de ces robots.

Parmi les applications de l'optimisation en robotique, on note les suivantes : trouver la meilleure solution pour augmenter la précision en espace 3D, réduire les vibrations, choisir le point de base appliqué optimal des robots pour minimiser le temps d'application, trouver des paramètres fonctionnels ou constructifs qui garantissent une consommation d'énergie réduite, réduire les vibrations et augmenter la stabilité du mouvement, choisir la variation optimale des moments dans toutes les articulations. Tout ce qui a été mentionné a un seul but : « **optimiser le comportement dynamique du robot** ».

I.6.2 Optimisation Des Performances Robotique

L'optimisation des performances robotiques a toujours été le but ultime depuis l'intégration de la robotique dans l'industrie. Plusieurs travaux ont été faits sur cette thématique, ainsi que pleins d'algorithmes ont été inventé où créer pour résoudre ce dernier.

On note quelques algorithmes d'optimisation :

I.6.2.1 Optimisation de la Position du Point de Base et Trajectoire

Dans ce genre d'application, il faudra trouver la meilleure position de la base du robot pour une application dans l'espace 3D afin d'obtenir un temps minimum de travail de ce dernier. L'algorithme contiendra la résolution cinématique inverse du robot utilisé. Le MCD modèle cinématique directe et le réseau de neurones pour obtenir les coordonnées internes qui assurent une précision supérieure à 0,001 mm entre la cible et l'outil de travail. L'application les points se trouvent dans un cube qui suit les contraintes les conditions. L'algorithme proposé est « **PIJMM- SBHTNN- TDRL** » (Pseudo Inverse Jacobian Matrix Method with Sigmoid Bipolar Hyperbolic Tangent Neural Network with Time Delay and Recurrent- Links). [13]

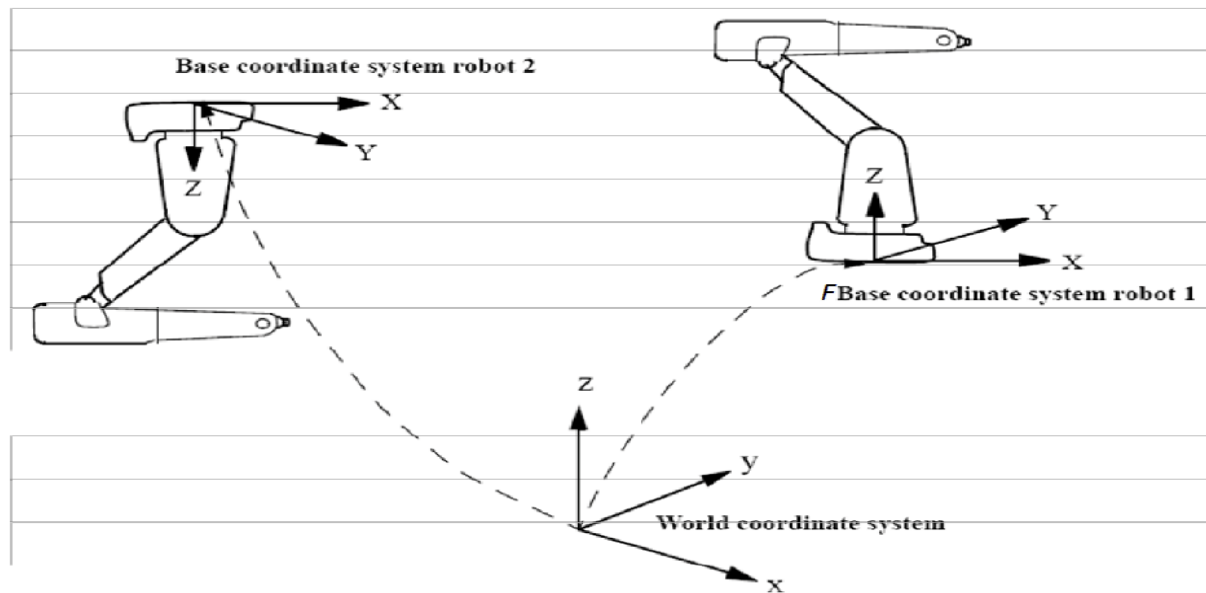


Figure I. 6 Aperçu sur la localisation du point de base du robot [13]

Voici le schéma exemplaire de l'algorithme utilisé :

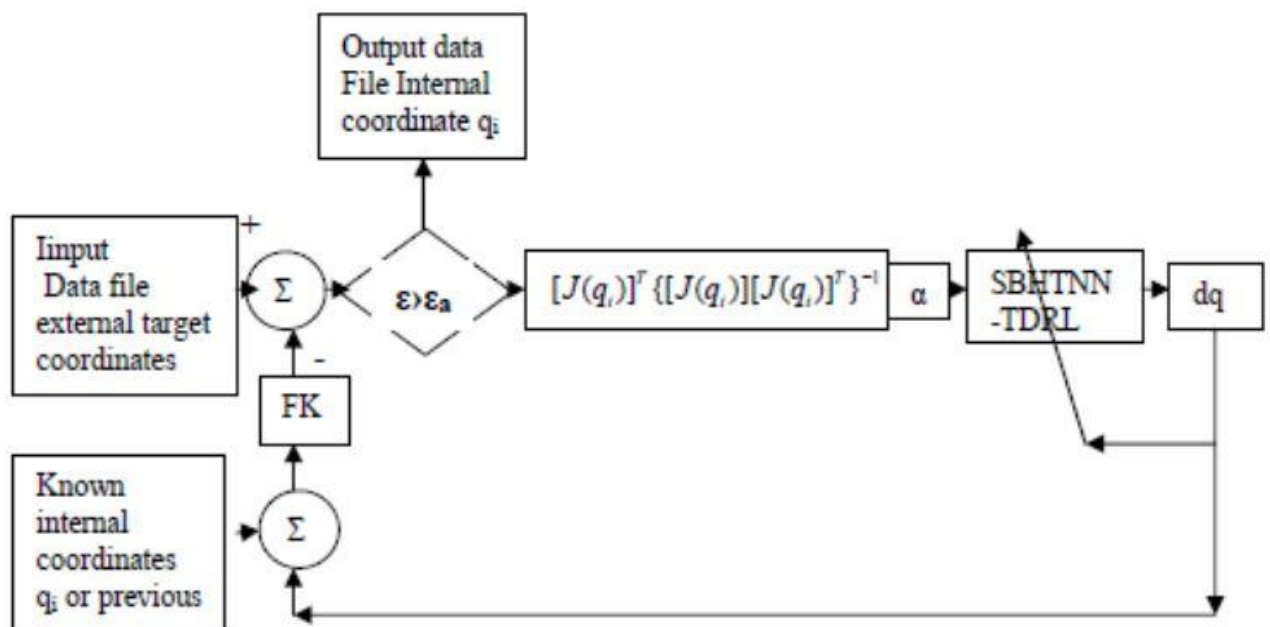


figure I. 7. Le schéma bloc de l'algorithme appliqué pour trouver la meilleure solution de la cinématique inverse [14]

I.6.2.2 Optimisation de La Consommation d'Énergie

1) Dans de nombreux cas, il est nécessaire de diminuer la consommation d'énergie, davantage dans l'application robotique industrielle. On introduit une méthode d'optimisation avec et sans contrainte.

L'algorithme dans ce cas est sous la forme complexe, il doit minimiser la consommation d'énergie en minimisant la trajectoire et les forces actives (couple) pour les mouvements successifs. [13]

$$E(q_i) = \min \iiint F_x F_y F_z d(x(q_i)) d(y(q_i)) d(z(q_i)) \quad (1)$$

On peut rajouter les contraintes suivantes :

$$F_{x\min} \leq F_x \leq F_{x\max}$$

$$F_{y\min} \leq F_y \leq F_{y\max}$$

$$F_{z\min} \leq F_z \leq F_{z\max}$$

$$X_{\min} \leq X \leq X_{\max}$$

$$Y_{\min} \leq Y \leq Y_{\max}$$

$$Z_{\min} \leq Z \leq Z_{\max}$$

$$Q_{i\min} \leq Q_i \leq Q_{i\max}$$

2) Cette approche d'optimisation d'énergie se base sur beaucoup de calculs mathématiques. A l'aide de la notation de Denavit-Hartenberg, l'approche commence d'abord par résolution du MCI du robot afin de trouver un ensemble de configurations d'articulations réalisables nécessaires à l'exécution de la tâche. La résolution du modèle cinématique inverse est généralement une étape difficile qui nécessite des analyses approfondies du robot. Par la suite, on résout le modèle dynamique inverse du robot pour analyser les forces et les couples appliqués sur chaque articulation et chaque lien du robot.

En outre, un calcul de la consommation d'énergie est effectué pour chaque configuration. L'étape finale du processus représente l'optimisation des configurations calculées en choisissant celle qui présente la consommation d'énergie la plus faible. [14]

La figure suivante montre l'approche expliquée :

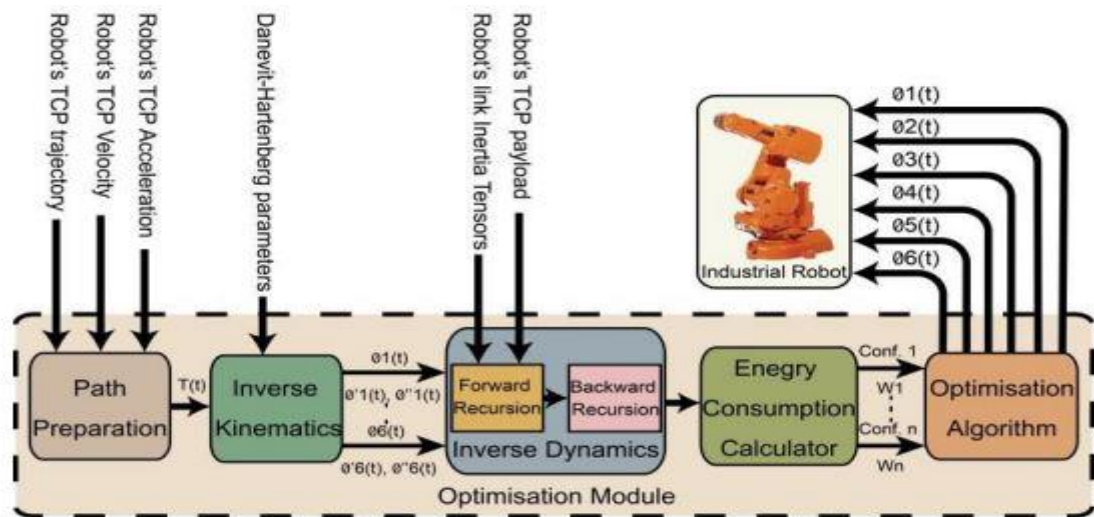


figure I. 8. Aperçu du système Développé. [14]

I.6.3 Planification des Trajectoire

Il existe aussi beaucoup d'algorithmes de planification de trajectoire des robots dont on note quelques-uns :

1. **Méthode par décomposition de l'environnement en cellule** : Dans ces cas, l'environnement discrétisé est représenté dans un graphe et trouver une trajectoire revient à chercher un chemin dans un graphe, problème qui peut être facilement traité informatiquement. [15]
2. **Planification par logique floue** : Cette méthode de planification se base sur la logique floue. Elle convertit chaque grandeur physique (Température, humidité, vitesse par exemple) en variables linguistiques (haut, bas par exemple). Cette approche permet de ne pas raisonner au niveau de la grandeur physique, mais plus comme un humain le ferait. [15]
3. **méthode de la bande élastique** : Une méthode qui tend une "bande élastique" entre le robot et l'objectif. Cette bande étant capable de se déformer en présence d'un obstacle, celle-ci génère dont une trajectoire envisageable par le robot. [15]
4. **méthode de la fenêtre dynamique** : Cette méthode un peu plus abstraite permet de prendre en compte les contraintes cinématiques du robot. Par contre, elle manque en flexibilité, ce qui rend difficile son implémentation dans un cadre multi-robot.
5. **planification par juxtaposition de splines polynomiales** : L'algorithme de 'Michael Defoort' génère des bouts de splines polynomiales qui respectent les contraintes

cinématiques du robot. Cette méthode permet donc de respecter les contraintes cinématiques du robot.

6. **DKP : (Deterministic Kinodynamic Planning)** : Cette méthode permet de respecter toutes les contraintes du robot et à d'éviter les problèmes de minimums locaux.

I.7 Plateforme CDTA

I.7.1. Présentation CDTA

Le Centre de Développement des Technologies Avancées « CDTA » est un établissement public à caractère scientifique et technologique « EPST », sous la tutelle du Ministère de l'Enseignement Supérieur de la Recherche Scientifique « MESRS ». [16]

Le CDTA contribue aux développements scientifiques et technologiques du pays par la formation des jeunes étudiants et doctorants sur les technologies les plus avancées, disponible sur le marché. Ce centre contribue au développement d'un large éventail de technologies telles que: l'architecture des systèmes et multimédia, la microélectronique et les nanotechnologies, la robotique et la production, la technologie laser et ses applications.

Le CDTA contient plusieurs division technologiques qui contient de nombreuse équipes. Chaque équipe se concentre sur une spécialité précise. Dans notre cas on s'intéresse à la division « **Division Productique et Robotique** ».

I.7.2 Divisons DPR:

La Division Productique et Robotique œuvre dans plusieurs domaines relevant en général de la productique et de la robotique. Ainsi, plusieurs axes sont abordés par les équipes de recherche structurées chacune autour d'une thématique de recherche particulière. Favorisant une approche R&D expérimentale, ces équipes utilisent pour leurs travaux un ensemble de plateformes d'expérimentation et de validation robotiques et une plateforme d'assemblage flexible et robotisée d'expérimentation et de démonstration des technologies Industrie 4.0. [16]

Les équipes de recherches sont les suivantes :

1. **Controle.**
2. **IRVA**(Itération réalité virtuelle augmentée)
3. **SRP** (Système robotisé de production)
4. **NCRM** (Navigation et control des robot mobiles)
5. **CSE** (Conception des système embarqués)
6. **CFAO** (Conception et fabrication assisté par ordinateur)

Les travaux de recherche portent particulièrement sur :

- Les investigations théoriques et expérimentales sur des approches et méthodes technologique des systèmes manufacturiers et robotiques cryophysiques type Industrie 4.0.
- L'étude et la mise en œuvre des modèles logicielles pour l'assistance à l'interaction multimodale de réalité virtuelle et augmentée.
- Le contrôle des systèmes dynamiques complexes tels que les systèmes non-linéaires, sous-actionnés, multi-variables, fortement couplés et même les engins volants (quad-rotor).
- La mise en place d'une plateforme robotique ayant la fonction de robot guide avec toute la logique de raisonnement du système pour répondre au mieux aux modalités d'interaction envisagées pour ce robot.
- Pilotage/Ordonnancement/Conduite des systèmes manufacturiers flexibles et complexes : développement d'approches exactes, heuristiques, par optimisation-simulation et méta-heuristiques pour la résolution centralisée et distribuée de différents modèles d'ordonnancement industriel.
- Interactions et simulations avancées pour les systèmes manufacturiers et robotiques cryophysiques.

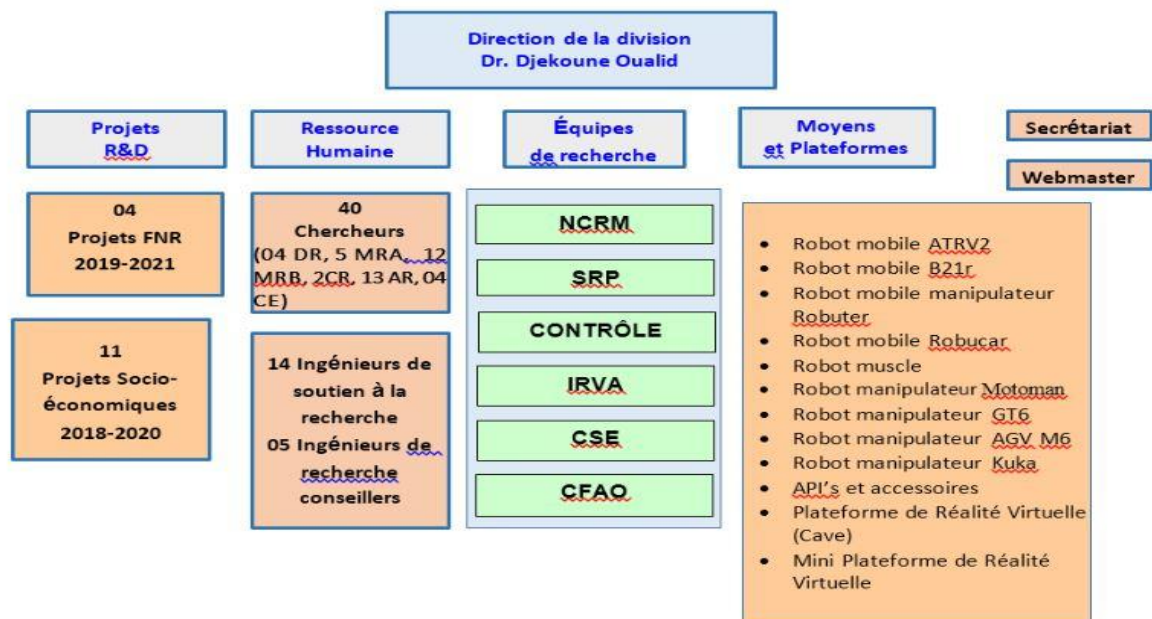


Figure I. 9. Aperçu général sur la division DPR [16]

I.8 Conclusion

Au cours du premier chapitre, on a parlé sur des généralités, ainsi qu'une présentation de l'industrie en général et de révolution industrielle en particulier jusqu'à arriver à la quatrième révolution industrielle, plus connu sous le nom « **industrie 4.0** » avec les nouvelles technologies introduites. On a aussi parlé de la robotique en général et la robotique industrielle en particulier. Un bref état de l'art est introduit sur l'optimisation robotique et le calcul de trajectoire. Enfin, une présentation de la plateforme CDTA avec une description sur la division productique robotique DPR, ainsi que les différentes équipes et projets développés au sein de cette division.

Le prochain chapitre contiendra la présentation de l'approche existante et une description détaillée sur la problématique traité dans ce projet. Il y aura aussi une présentation des différents algorithmes d'optimisation méta-heuristiques.

Chapitre II :

Optimisation des

tâches robotique

II.1 Introduction

La robotique est devenue indispensable dans divers secteurs industriels en raison de ses avantages et de sa contribution à l'industrie. Cependant, cette technologie a des limites de coût et de temps d'exécution. Dans ce chapitre, nous avons essayé de proposer une solution pour résoudre ces problèmes.

Un système d'apprentissage a été développé pour le robot « **KUKA LBR IIWA** » dans le but de minimiser l'effort et de terminer la tâche le plus rapidement possible. Ce chapitre détaille l'approche existante du problème abordé, la problématique du travail ainsi que l'apport de ce dernier. Un aperçu sur les algorithmes d'optimisation métaheuristique avec une présentation détaillée sur les différents aspects d'optimisation en présentant l'algorithme retenu dans cette étude.

II.2 Approche Existante

II.2.1 Présentation

Le sujet de travail [15] a été proposé par Mr M.GAHAM maître de recherche au CDTA et a été développé au sein de la division DPR du CDTA. L'objectif du travail développé consiste à proposer un développement d'un système de reconfiguration des tâches commandé par le robot KUKA LBR IIWA, dans le but d'une commande simple et conviviale, sans arrêt du robot ni utilisation de son interface habituelle de programmation, même les utilisateurs disposant de faibles connaissances techniques peuvent exécuter des tâches à l'aide de ce système. [1]

II.2.2 Problématique

La programmation est l'outil principal de commande des robots collaboratifs industriels. Cet outil est toujours en développement et cela permet d'agrandir le champ d'utilisation de ces robots. Malheureusement cette dernière est toujours limitée aux experts c'est à dire une personne qui ne maîtrise pas la programmation textuelle ne pourra pas aller loin dans l'utilisation des cobots. [1]

II.2.3 Solution Proposé

Afin de régler le problème mentionné dans le titre précédent. Durant ce travail, les réalisateurs du projet ont envisagé le développement d'un système d'apprentissage qui permet à n'importe quel opérateur de coopérer à la commande des robots collaboratifs. L'opérateur aura la possibilité d'éditer et de reconfigurer des tâches bien définies par l'expert à l'avance, sans devoir

arrêter le système (comme dans la plupart des cas), ou chercher l'aide d'un expert, ce qui facilitera l'utilisation des cobots.[1]

Ce système est réalisé sur la base de deux applications java, à savoir :

- Une application d'apprentissage et de configuration extérieure.
- Une application de commande.

II.2.4 Les sous tâches

Une sous-tâche est une tâche qui s'inscrit dans une tâche plus complexe.[1]

Une sous-tâche représente un ensemble de tâches de base qui sont essentielles pour récupérer le contenu d'une tâche. Exemple : Déplacement linéaire d'un point défini à une vitesse définie.

II.2.5 Aspect Fonctionnel de Système d'Apprentissage/Commande

Le fonctionnement du système réalisé est basé sur le principe de l'apprentissage en boucle. La première étape consiste à apprendre les coordonnées des points, puis à effectuer la tâche avec la configuration appropriée pour effectuer la bonne exécution de la tâche, cela se fait en deux étapes.

II.2.5.1 Application d'Apprentissage Et De Commande Robotique

Afin de réaliser un système d'apprentissage et de commande, il va falloir créer deux programmes connectés qu'on va appeler (client et serveur). Cette étape permettra d'établir une communication entre les deux applications citées(commande, apprentissage).

En première étape l'application d'apprentissage et de configuration consiste à faire un apprentissage des points, ainsi que l'édition, la configuration des sous-tâches puis l'envoi de ces dernières à l'application de commande après avoir été configuré selon les facteurs et les besoins du projet.

Afin de contrôler le robot, l'apprentissage de points se fait en utilisant le mode commande par impédance (le hand-guiding en anglais). Ces points représentent la position du point centrale de l'outil de travail du robot au moment d'apprentissage.

La configuration des tâches se fait par l'opérateur lui-même, en choisissant une suite de points (chemin) et le type de mouvement dans le but de satisfaire son cahier des charges.

Afin d'accélérer l'exécution des tâches, les développeurs ont opté de créer deux programmes serveur/client. Le premier est responsable sur l'apprentissage des points et l'autre pour l'exécution des tâches.

La procédure générale représentée sur la **figure II.1**, de dialogue entre le serveur de l'application d'apprentissage et le contrôleur du robot (le client) est la suivante :

1. Le serveur initialisé se place en écoute (attente de la requête envoyée par le contrôleur).

2. Le contrôleur émet une requête à destination du serveur.
3. Le serveur renvoie une réponse au client. Il rend le service,
4. Le serveur se replace en écoute (attente des points envoyés par le client).

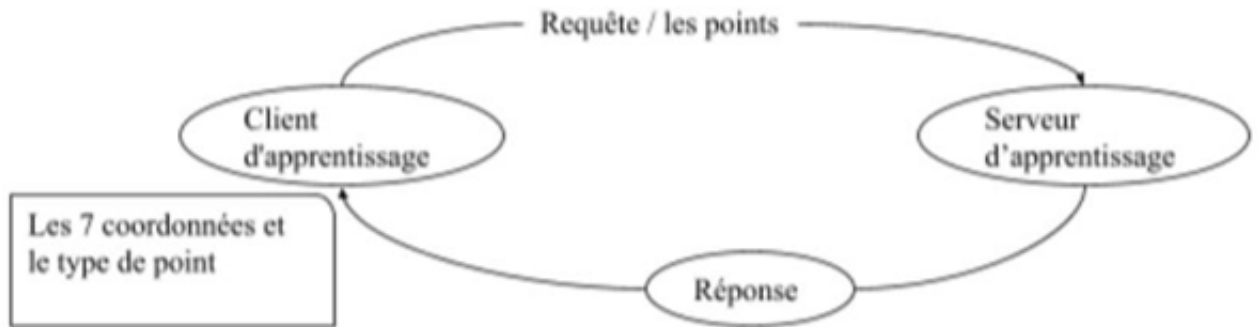


Figure II. 1. Schéma de dialogue pour l'apprentissage des points [1]

II.2.5.2 Application de commande/d'exécution

La commande est représentée par des tâches prédéfinies par l'expert, configurées et éditées par l'opérateur, les instructions sont reçues, traitées et exécutées par l'application de commande qui est basée sur le contrôleur.

Le serveur d'exécution est connecté aux sous-tâches sélectionnées sur l'interface graphique.

Le contrôleur reçoit l'ensemble des sous-tâches et s'occupe de les exécuter sous forme d'une tâche complète qui représente la commande du robot, voir.[1]

La procédure générale de l'application de commande se résume en quatre étapes :

1. La réception des sous-tâches venant la première application.
2. L'exécution de la commande sur le robot KUKA.
3. L'envoi d'un accusé de réception.
4. Attente d'une nouvelle configuration de sous-tâche communiquée par la première application.

II.2.6. Problème Posé

Le problème qui se pose dans cette application d'apprentissage/commande est le suivant :

Les tâches et les chemins des différentes commandes ont été choisi pour remplir un cahier des charges sans optimiser les performances du robot.

Dans le domaine industriel, le dernier mot revient aux chiffres, et avec une haute consommation d'énergie et un lent temps d'exécution, l'introduction d'un modèle d'optimisation est impérative pour diminuer les coûts énergétiques et temporels.

Cela indique qu'on peut optimiser la consommation d'énergie et le temps de cycle de chaque application effectué avec ce système d'apprentissage/commande en introduisant un modèle d'optimisation des performances.

Dans le prochain titre, nous parlerons de la problématique d'optimisation traitée dans ce projet de fin d'études et l'approche choisie pour essayer de résoudre ce dernier.

II.3 Problématique

Selon des statistiques récentes, l'industrie manufacturière est l'un des plus gros consommateurs d'énergie. Les robots industriels sont souvent considérés comme des équipements non-durables qui exigent un niveau élevé de consommation d'énergie. D'un autre côté, ces robots offrent une précision, une force et des capacités qui permettent de fabriquer des produits finis de haute qualité.[17]

En conséquence, la minimisation de la consommation d'énergie des robots ainsi que leurs temps d'exécution sont devenues un objectif majeur pour de nombreux groupes de recherche et fabricants de robots. Plusieurs chercheurs se sont attachés à définir des outils pour mesurer et analyser la consommation d'énergie des robots afin d'optimiser cette dernière.

Par exemple, le travail rapporté par [18] contribue à identifier les stratégies d'efficacité énergétique dans les applications robotiques. D'autres, comme [19], ont résumés différentes méthodes d'utilisation efficace de l'énergie pour les robots industriels courants.

L'énergie totale consommée par le robot est généralement affectée par le couple requis sur chaque articulation. D'autres chercheurs se sont concentrés sur l'optimisation d'un système de fabrication robotisée dans son ensemble [20-21]. Malgré les efforts susmentionnés, la minimisation de la consommation d'énergie des robots ainsi que le temps de cycle reste un défi et nécessite des efforts supplémentaires.

Dans ce projet, nous présentons une approche visant à minimiser le temps d'exécution et la consommation d'énergie des mouvements d'un robot « **KUKA LBR IIWA** » avec un algorithme évolutionnaire. On y parvient en développant un modèle qui optimise le parcours d'un ensemble de points dans l'espace de travail, pour suivre une certaine trajectoire définie par un opérateur, tout en agissant sur les valeurs maximales des vitesses et accélérations. On abordera une approche qui se rassemble à un **problème du voyageur de commerce TSP (Traveling salesman problem)**.

Cette méthode consiste à prendre une suite de points à parcourir dans l'espace de travail, puis essayer de trouver l'ordre/ordonnancement optimal des points afin que le robot puisse parcourir

ces coordonnées avec des coûts énergétiques et temporels optimaux. L'optimisation se fera en variant l'ordre des points ainsi que la vitesse et l'accélération maximale de chaque articulation pour chaque mouvement entre deux points.

Nous évaluons les performances du système en utilisant un logiciel de simulation robotique open source appelé «**CoppeliaSim** ». Plusieurs scénarios ont été utilisés pour étudier les résultats du modèle développé; Plusieurs expériences ont été réalisées pour différents ensembles de positions initiales du robot afin d'identifier les différents comportements que le modèle développé peut avoir sur l'optimisation des performances du robot étudié.

II.4 Algorithmes Métaheuristique

Au cours des dernières décennies, de nombreux algorithmes d'optimisation, y compris des algorithmes approximatifs ont été proposés pour résoudre les problèmes d'optimisation. Dans la classe des algorithmes d'optimisation exacts, la conception et la mise en œuvre des algorithmes sont généralement basés sur des méthodes telles que la programmation dynamique, la méthode de backtracking et la méthode d'évaluation et de séparation (branch and bound) .[22] Cependant, ces algorithmes ont une bonne performance dans de nombreux problèmes, mais ils ne sont pas efficaces dans la résolution de problèmes d'optimisation combinatoires et non linéaires, en raison du fait que l'espace de recherche augmente de façon exponentielle avec la taille du problème et une recherche exhaustive est impraticable dans ces problèmes.[22]

En outre, les méthodes approximatives comme les algorithmes glouton nécessitent généralement de faire plusieurs hypothèses qui pourraient ne pas être faciles à valider dans de nombreuses situations. [22]

Par conséquent, un ensemble d'algorithmes plus adaptables et flexibles est nécessaire pour surmonter ces limitations.[22]

Sur la base de cette motivation, plusieurs algorithmes généralement inspirés Par des phénomènes naturels ont été proposés dans la littérature. Parmi eux, certains algorithmes de recherche méta-heuristique avec cadre basé sur la population des capacités satisfaisantes pour gérer les problèmes d'optimisation de haute dimension.[22]

II.4.1 définition et concept des algorithmes méta-heuristique

Une métaheuristique est un algorithme d'optimisation visant à résoudre des problèmes d'optimisation difficiles (souvent issus des domaines de la recherche opérationnelle, de l'ingénierie ou de l'intelligence artificielle).[23]

Les métaheuristicues sont généralement des algorithmes stochastiques itératifs, qui progressent,

vers un optimum global, c'est-à-dire l'extremum global d'une fonction par échantillonnage d'une fonction objectif. Elles se comportent comme des algorithmes de recherche, tentant d'apprendre les caractéristiques d'un problème afin d'en trouver une approximation de la meilleure solution (d'une manière proche des algorithmes d'approximation). [23]

Il existe un grand nombre de métaheuristiques différentes, allant de la simple recherche locale à des algorithmes complexes de recherche globale. Ces méthodes utilisent cependant un haut niveau d'abstraction, leur permettant d'être adaptées à une large gamme de problèmes différents.[23]

Le but d'une métaheuristique est de résoudre un problème d'optimisation donné : elle cherche un objet mathématique (une permutation, un vecteur, etc.) minimisant (ou maximisant) une fonction objective, qui décrit la qualité d'une solution au problème.[23]

L'ensemble des solutions possibles forme l'espace de recherche. L'espace de recherche est au minimum borné, mais peut être également limité par un ensemble de contraintes.

Les métaheuristiques manipulent une ou plusieurs solutions, à la recherche de l'optimum, la meilleure solution au problème. Les itérations successives doivent permettre de passer d'une solution de mauvaise qualité à la solution optimale. L'algorithme s'arrête après avoir atteint un critère d'arrêt, consistant généralement en l'atteinte du temps d'exécution imparti ou en une précision demandée. Les métaheuristiques ne nécessitent pas de connaissances particulières sur le problème optimisé pour fonctionner, le fait de pouvoir associer une (ou plusieurs) valeur à une solution est la seule information nécessaire. [23]

Les métaheuristique sont souvent employées en optimisation combinatoire, mais on en rencontre également pour des problèmes continus ou mixtes (problèmes à variables discrètes et continues).[23]

Les métaheuristiques s'appuie autour de plusieurs notions :

- **Voisinage**

Le voisinage d'une solution est le sous-ensemble de solutions qui peuvent être obtenues par un ensemble donné de transformations. Par exemple, dans le problème TSP, un voisinage est un ensemble de solutions qui peuvent être construites en permutant deux villes dans une solution donnée.[23]

- **Intensification**

Le renforcement (ou l'exploitation) vise à utiliser les informations collectées pour définir et parcourir les régions intéressantes de l'espace de recherche.[23]

- **Diversification**

La diversification (ou exploration) fait référence à un processus visant à recueillir des informations sur un problème d'optimisation .[23]

- **Apprentissage**

La mémoire est le support de l'apprentissage, ce qui permet à l'algorithme de ne considérer que les régions où un optimum global est susceptible d'être trouvé, évitant ainsi les optima locaux.

La figure II.2 nous montre les 3 phases d'un algorithmes métaheuristique ou Les points rouges représentent l'échantillonnage de la fonction objectif

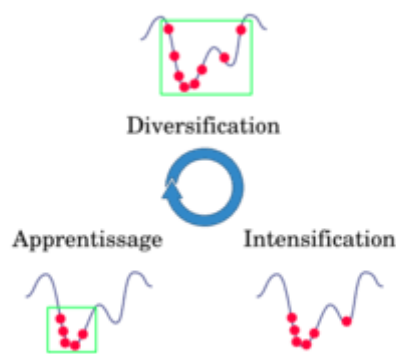


Figure II. 2. les 3 phases d'un algorithmes métaheuristique [23]

Les méta-heuristiques évoluent de manière itérative, alternant des phases de renforcement, de diversification et d'apprentissage, ou intègrent plus étroitement ces concepts. L'état de départ est généralement choisi au hasard, puis l'algorithme s'exécute jusqu'à ce que le critère d'arrêt soit atteint.[23]

II.4.2 Classification des algorithmes métaheuristique

Les métaheuristiques sont généralement classées dans 2 catégories. On peut distinguer les méta-heuristiques qui font évoluer une seule solution sur l'espace de recherche à chaque itération, et les méta-heuristiques qui se basent sur un grand nombre de solutions. D'une manière générale, les méta-heuristiques basées sur une solution unique se concentrent davantage sur l'exploitation de l'espace de recherche, nous ne pouvons donc jamais être sûrs d'obtenir la valeur optimale. Les métaheuristiques à base de population sont assez exploratoires et peuvent mieux diversifier l'espace de recherche.[24]

II.4.2.1 Les métaheuristiques à solution unique

Dans cette section, nous introduisons les métaheuristiques basées sur une solution unique, également appelées méthodes de trajectoire, elles construisent des trajectoires dans l'espace de

recherche, commençant par une seule solution initiale et en s'en éloignant progressivement.[24]

Les méthodes de trajectoire comprennent principalement :

- la méthode de descente
- la méthode du recuit simulé
- la recherche taboue
- la méthode GRASP
- la recherche à voisinage variable
- la recherche locale itérée.

II.4.2.2 Les métaheuristiques à population de solutions

Contrairement aux algorithmes qui partent d'une seule solution, les métaheuristiques à population de solutions améliorent, au fur et à mesure des itérations, une population de solutions. On distingue dans cette catégorie, les algorithmes évolutionnaires, qui sont une famille d'algorithmes issus de la théorie de l'évolution par la sélection naturelle, énoncée par Charles Darwin et les algorithmes d'intelligence en essaim qui proviennent eux aussi d'analogies avec des phénomènes biologiques naturels.[24]

II.4.2.2.1 les Algorithmes évolutionnaires

Les algorithmes évolutionnistes ou algorithmes évolutionnaires(EC: Evolutionary Computation), sont une famille d'algorithmes s'inspirant de la théorie de l'évolution darwinienne pour résoudre des problèmes divers. [24]

Selon la théorie de l'évolution de Darwin ,l'évolution des espèces est la conséquence de la conjonction de deux phénomènes : d'une part la sélection naturelle qui favorise les individus les plus adaptés à leur milieu à survivre et à se reproduire, laissant une descendance qui transmettra leurs gènes et d'autre part, la présence de variations non dirigées parmi les traits génétiques des espèces (mutations).[24]

Dans la famille des algorithmes évolutionnaires, on retrouve plusieurs types :

- Algorithme génétique (GA : Genetic Algorithms)
- Algorithme à estimation de distribution (EDA : Estimation of Distribution Algorithm)
- Algorithme à évolution différentielle (DE : Differential Evolution)
- Algorithme Co évolutionnaire (CoEA : CoEvolutionary Algorithms)
- Algorithme culturel

II.4.2.2.2 L'intelligence en essaim (SI : Swarm Intelligence)

Cet algorithme est né de la modélisation mathématique et informatique des phénomènes biologiques rencontrés en éthologie. Elle recouvre un ensemble d'algorithmes, à base de population d'agents simples (entités capables d'exécuter certaines opérations), qui interagissent localement les uns avec les autres et avec leur environnement. On retrouve 2 majeurs algorithmes :[24]

- L'optimisation par les colonies de fourmis (ACO : Ant Colony Optimization)
- L'optimisation par Essaim particulaire (PSO : Particle Swarm Optimization)

D'autres algorithmes d'optimisation qui proviennent d'analogies avec des phénomènes biologiques naturels ont été proposés. Parmi eux :

- l'algorithme Bacterial foraging optimization
- l'optimisation par colonie d'abeilles (Bee Colony Optimization)
- les systèmes immunitaires artificiels
- l'algorithme d'optimisation basée sur la biogéographie insulaire (Biogeography-Based Optimization).

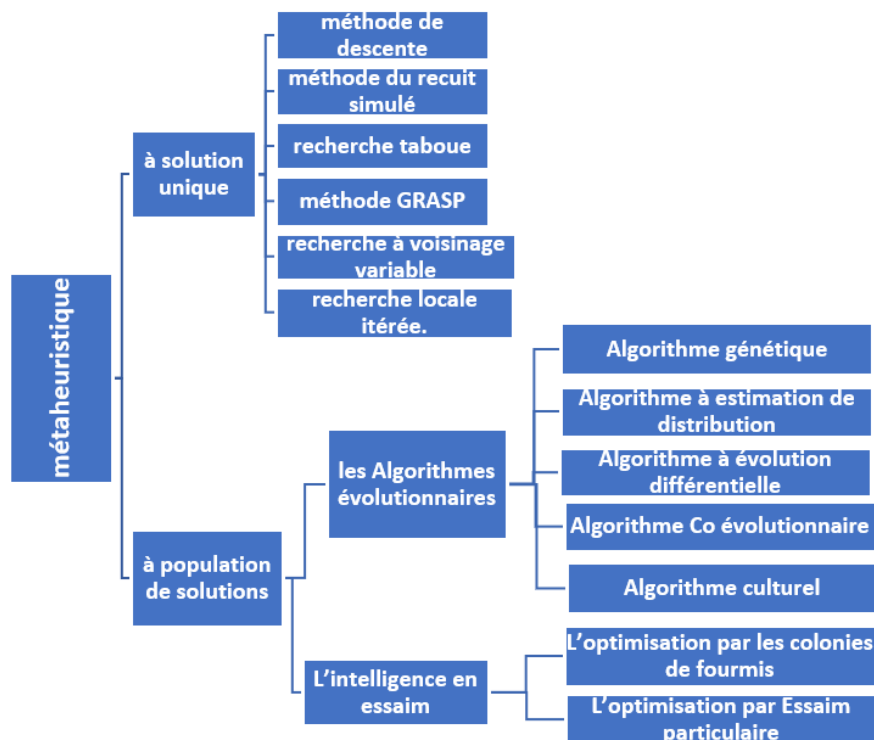


Figure II. 3. classes des algorithmes métaheuristique

La figure II.3 représente les différentes classes des algorithmes métaheuristique

II.5 Algorithmes Génétique

Les algorithmes génétiques (GA: Genetic Algorithms) sont sans aucun doute la technique la plus populaire et la plus largement utilisée des algorithmes évolutionnaires. Les origines de ces algorithmes remontent au début des années 1970, avec les travaux de John Holland et ses élèves à l'Université du Michigan sur les systèmes adaptatifs. Le travail de référence de David E. Goldberg a grandement contribué à leur développement.

L'algorithme génétique va reproduire ce modèle d'évolution dans le but de trouver des solutions pour un problème donné. Souvent, on utilise des termes biologiques et génétiques afin de mieux représenter chacun des concepts de cet algorithme.

- une population sera un ensemble d'individus.
- un individu sera une solution à un problème donné.
- un gène sera une partie d'une solution, donc d'un individu.
- une génération est une itération de notre algorithme.

Les algorithmes génétiques vont faire évoluer la population pour améliorer ces individus. Par conséquent, à chaque génération, un groupe d'individus sera mis en avant plutôt qu'un individu spécifique. Donc, au lieu d'une solution unique, nous obtiendrons un ensemble de solutions au problème. Les solutions trouvées sont souvent différentes mais de même qualité. Nous reviendrons sur le concept de qualité de la solution dans les prochaines parties. [25]

Le processus d'un algorithme génétique peut être divisé en cinq parties :

1. La création de la population initiale.
2. L'évaluation des individus.
3. La création de nouveaux individus.
4. L'insertion des nouveaux individus dans la population.
5. Réitération du processus.

En premier lieu nous allons expliquer l'algorithme génétique d'une manière générale ensuite nous allons parler sur l'approche utilisée dans notre travail.

II.5.1 Présentation l'algorithme génétique

II.5.1.1 La création de la population initiale

Pour démarrer un algorithme génétique, il faut lui fournir une population à faire évoluer. La manière dans laquelle le programmeur créera chaque individu de cette population est complètement libre. Il suffit que tous les individus créés prennent la forme d'une solution

possible, et il n'est pas nécessaire d'envisager de créer de bons individus. Il suffit juste de tenir compte du problème. [25]

Il est tout à fait possible de créer les individus de manière aléatoire. Cette approche apporte un concept très utile aux algorithmes génétiques : la diversité. Plus les individus de la population initiale seront différents les uns des autres, plus nous aurons de chances d'y trouver, non pas la solution parfaite, mais suffisamment pour en faire les meilleures solutions possibles. [25]

La taille de la population initiale est également laissée au choix du programmeur. Il n'est généralement pas nécessaire d'utiliser une populations excessive. [25]

II.5.1.2 L'évaluation des individus

Une fois la population initiale créée, nous sélectionnerons les individus les plus prometteurs, ceux qui participeront à l'amélioration de notre population. Par conséquent, nous attribuerons à chaque individu une "note" ou un indice de qualité. La méthode d'évaluation d'un individu est laissée au programmeur, en fonction du problème qu'il doit optimiser ou résoudre. Cette étape intermédiaire d'évaluation peut même être une étape importante dans le processus d'amélioration de la population. En effet, des individus différents ne sont pas toujours comparables, et il n'est pas toujours possible de dire que l'un est meilleur ou pire qu'un autre. On est toujours en mesure de dire si un nombre est supérieur ou inférieur à l'autre, mais on ne peut pas dire si un vecteur est supérieur ou inférieur à l'autre. [25]

II.5.1.3 La production de nouveaux individus

II.5.1.3.1 La sélection

Cette étape consiste à choisir les combinaisons de la population qui seront ensuite croisées et mutées. Nous allons essayer de prendre des morceaux de solution de certains individus et d'autres morceaux d'autres individus pour créer des nouveaux individus Il s'agit là de favoriser la sélection des meilleures combinaisons, tout en laissant une petite chance aux moins bonnes combinaisons. Les méthodes de sélection permettent de déterminer quels individus nous allons croiser. Il existe de nombreuses façons de procéder à cette étape de sélection. [25]

- **La sélection par tournoi**

Elle consiste à choisir aléatoirement deux combinaisons et à sélectionner la meilleure des deux (ou bien à sélectionner une des deux selon une probabilité dépendant de la fonction objectif). [25]

- **La roulette**

Cette méthode s'inspire des roues de loterie. À chacun des individus de la population est associé un secteur d'une roue. L'angle du secteur étant proportionnel à la qualité de l'individu qu'il représente. Les tirages des individus sont pondérés par leur qualité. les meilleurs individus ont plus de chances d'être croisés et de participer à l'amélioration de la population. [25]

- **La sélection par rang**

cette technique consiste à trier la population en fonction de la qualité des individus puis leur attribuer à chacun un rang. Les individus de moins bonne qualité obtiennent un rang faible Et ainsi en itérant sur chaque individu on finit par attribuer le rang N au meilleur individu (où N est la taille de la population). La suite de la méthode consiste uniquement en l'implémentation d'une roulette basée sur les rangs des individus. [25]

- **L'élitisme**

Cette méthode permet de mettre en avant les meilleurs individus de la population. les individus les plus prometteurs qui vont participer à l'amélioration de la population. [25]

la figure II.4 nous montre un exemple avec des individus en représentation binaire une fois la sélection effectuée :

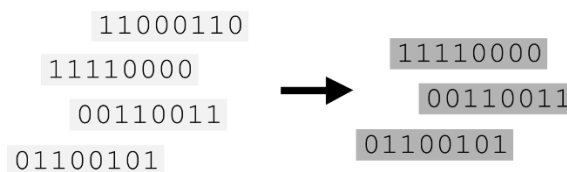


Figure II. 4. exemple de sélection [27]

II.5.1.3.2 Le croisement

Le croisement, consiste à générer de nouvelles combinaisons, à partir des combinaisons sélectionnées. il existe de nombreux opérateurs de croisement. Une méthode simple consiste à choisir aléatoirement un point de croisement, à couper chaque combinaison parente en ce point, puis à reformer deux enfants en échangeant les parties composant les parents de part et d'autre du point de croisement. [26]

la figure II.5 nous montre un exemple avec des individus en représentation binaire une fois le croisement effectuée :

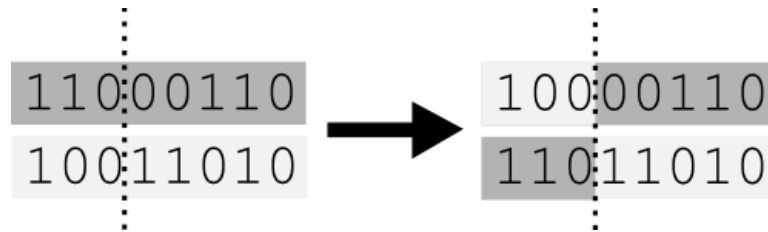


Figure II. 5. exemple de croisement [27]

II.5.1.3.3 Les mutations

La mutation consiste à altérer un gène dans un chromosome selon un facteur de mutation. Ce facteur est la probabilité qu'une mutation soit effectuée sur un individu. Cet opérateur est l'application du principe de variation de la théorie de Darwin et permet, par la même occasion, d'éviter une convergence prématurée de l'algorithme vers un extremum local. [27]

la figure II.6 nous montre un exemple avec des individus en représentation binaire une fois la mutation effectuée :

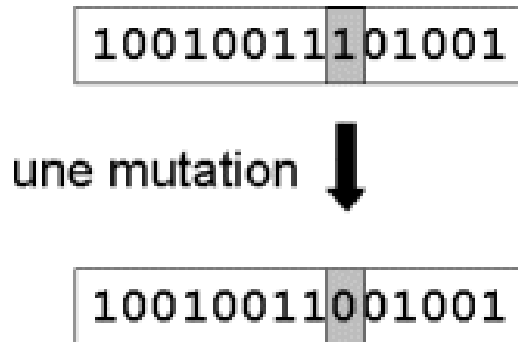


Figure II. 6. exemple de mutation

II.5.1.4 L'insertion des nouveaux individus dans la population

Une fois de nouveaux individus sont créés, il faut sélectionner ceux qui vont continuer à participer à l'amélioration de la population. Une fois encore, libre au programmeur de choisir ceux qu'il souhaite conserver.

Cette étape consiste à remplacer certaines combinaisons de la génération précédente par certaines combinaisons issues des opérations de croisement et de mutation, formant de la sorte une nouvelle génération. il existe différentes stratégies de remplacement. On peut par exemple

choisir de ne garder que les meilleurs individus, qu'ils soient issus de la nouvelle génération ou de l'ancienne, ou bien ne garder que les individus de la nouvelle génération, indépendamment de leur qualité.

Cependant, Le nombre d'individus N à conserver est à choisir avec soin. En prenant un N trop faible, la prochaine itération de l'algorithme se fera avec une population plus petite et elle deviendra de plus en plus petite au fil des générations, elle pourrait même disparaître. En prenant un N de plus en plus grand, nous prenons le risque d'augmenter le temps de traitement puisque la population de chaque génération sera plus grande.

Une fois la nouvelle population obtenue, on recommence le processus d'amélioration des individus pour obtenir une nouvelle population et ainsi de suite. [25]

II.5.1.5 Critères d'arrêt

le processus d'évolution est itéré, de génération en génération, jusqu'à ce qu'une combinaison de qualité suffisante soit générée, ou bien jusqu'à ce qu'une limite de temps soit atteinte ou bien jusque à l'arrivée d'un nombre de génération spécifique introduit par le programmeur.[25]

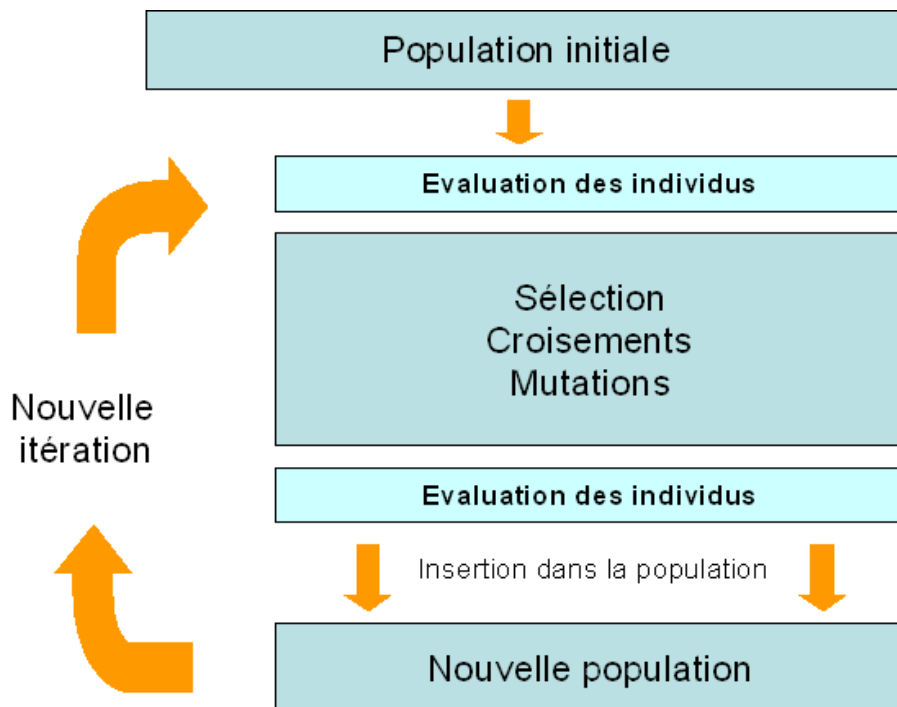


Figure II. 7 schéma explicatif de l'algorithme génétique

la figure II.7 représente le schéma du déroulement d'un algorithme génétique

II.6 approche utilise dans notre travail

II.6.1 La création de la population initiale

Nous initialisant une population d'individus au hasard par exemple $[p1, p3, p4, p2]$, c.à.d, le robot va suivre cette route qui commence par la position $p1$ puis $p3$ ensuite $p4$ pour finir en $p2$.

Cependant, nous avons créé 2 autres vecteurs, une pour la vitesse qui contient des coefficients compris entre 0.1 et 1 et l'autre pour l'accélération qui contient des coefficients compris entre 0.5 et 2.

II.6.2 L'évaluation des individus

Dans notre cas, le critère d'évaluation sera basé sur le temps de parcours des points par le robot et l'énergie dépensée. Nous avons choisi en premier lieu de faire une somme pondérée en multipliant l'énergie par 0.5 et le temps par 0.5, puis nous allons faire l'inverse de cette somme qui sera notre critère d'évaluation qu'on nommera fitness, pour rendre notre problème un problème de maximisation. Plus l'énergie et le temps sont bas plus le résultat sera meilleur.

II.6.3 La création de nouveaux individus

II.6.3.1 La sélection

Pour la sélection des individus nous avons choisi d'utiliser la méthode de la roulette et l'élitisme.

On a commencé en premier lieu d'appliquer la méthode de roulette, après cela, nous utiliserons la méthode de l'élitisme où les individus les plus performants de la population vont automatiquement se transmettre à la génération suivante, en s'assurant que les individus les plus performants persistent.

II.6.3.2 croisement

Vu que notre problème est un problème semblable au problème du TSP, nous devons inclure tous les points exactement une fois. Par exemple, on ne peut pas avoir $[p1, p2, p2, p4]$.

Donc nous sélectionnons aléatoirement un sous-ensemble du premier parent, puis nous remplissons le reste avec les gènes du second parent sans duplication des gènes sélectionnés du premier parent.

II.6.3.3 mutation

La méthode de mutation que nous avons appliquée consiste en une permutation de deux points. Nous sommes certains que les individus mutés auront toujours la forme d'une solution potentielle, car nous ne changeons que l'ordre des points.

Par exemple, $[p1, p2, p3, p4]$ pourra être muté en $[p1, p2, p4, p3]$

Pour la vitesse et l'accélération, nous allons augmenter ou diminuer les éléments du vecteur qui contient les coefficients de vitesse et accélération par 0.1.

II.6.4 le critère d'arrêt

On fixe un nombre de générations et quand l'algorithme arrive à cette génération, il sera arrêté.

II.7 conclusion

Au cours de ce chapitre, nous avons présenté notre problématique et l'approche existante dans un travail qui a été fait, que nous utilisons pour collecter des angles à différents points de chaque articulation du robot, puis nous avons étudié l'algorithme métaheuristique avec ces différentes classes. Nous avons constaté que ces algorithmes sont les mieux adaptés à notre problème, plus précisément l'algorithme génétique que nous avons expliqué en premier lieu de manière générale, ensuite, nous avons expliqué l'approche utilisée dans cet algorithme pour résoudre notre problème.

Dans le chapitre suivant nous allons parler sur l'environnement de programmation et du logiciel de simulation « CoppeliaSim »

Chapitre III :

Réalisation et

implémentation

III.1. Introduction :

Dans ce chapitre nous présentons l'aspect pratique ainsi que les différentes étapes du code réalisé dans notre projet qui consiste à créer un modèle d'optimisation de l'énergie et du temps à l'aide d'un algorithme génétique qui nous donnera le meilleur chemin à parcourir avec les vitesses et accélération maximales appropriée.

La programmation se divise en deux étapes :

1. Développement d'un modèle d'optimisation sur le langage de programmation « Python ».
2. Simulation des résultats sur le logiciel open-source « CoppeliaSim ».

En montrant des tests et des implémentations à chaque étape du développement, nous prouvons voir et vérifier le bon fonctionnement du modèle.

III.2. Environnement de Programmation :

La programmation de ce modèle d'optimisation se divise en deux parties.

La première partie représente le codage sur Python, cette partie se constitue de 3 codes pythons qui se complètent entre eux :

1. Le premier code représente l'algorithme d'optimisation qui est responsable sur le travail d'optimisation de notre modèle.
2. Le deuxième code contient le transfert des données entre Python et le simulateur « CoppeliaSim ».
3. Le troisième code est responsable sur la connexion Python/CoppeliaSim, ainsi que le commencement et l'arrêt de la simulation.

La deuxième partie du modèle est la simulation des résultats sur « CoppeliaSim ».

« CoppeliaSim » contient un seul code qui s'écrit sous le langage de programmation « LUA ».

III.3. Programmation Python :

Avant de présenter le code réalisé, il est nécessaire présenter quelques définitions techniques sur les outils et les bibliothèques utilisées dans la création du modèle d'optimisation.

III.3.1 Outils De Programmation :

III.3.1.1 Spyder:

Initialement créé et développé par « Pierre Raybaut » en 2009, Spyder est un environnement de développement intégré (IDE) multiplateforme ouvert pour la programmation scientifique en langage Python. **Spyder** s'intègre à un certain nombre de paquets/librairies importants de la pile

Python scientifique, notamment NumPy, SciPy, Matplotlib, pandas, IPython, SymPy et Cython, ainsi qu'à d'autres logiciels libres. [28]

Nous avons choisi de travailler avec cet IDE car il contient la plupart des librairies nécessaires pour notre projet.

III.3.1.2. Connexion API :

API est l'acronyme d'Application Programming Interface (interface de programmation d'application), une solution qui permet à deux applications de communiquer des données entre elles.[29]

Dans notre cas, nous enverrons des données du code Python au simulateur afin de tester la solution proposée par le GA, puis renvoyer les résultats obtenus sur le simulateur à Python et procéder à la prochaine étape.

III.3.1.3 Les Classes :

Les classes sont un moyen de réunir des données et des fonctionnalités. Créer une nouvelle classe crée un nouveau « *type* » d'objet et ainsi de nouvelles « *instances* » de ce type peuvent être construites. Chaque instance peut avoir ses propres attributs, ce qui définit son état. Une instance peut aussi avoir des méthodes (définies par la classe de l'instance) pour modifier son état. [30]

Dans notre cas, chaque classe contiendra une solution proposée appelée vecteur chemin. Cette solution aura un vecteur point et un vecteur vitesse et un vecteur accélération, ainsi qu'elle aura une valeur fitness. La valeur fitness permettra de déterminer la validité de la solution proposée.

III.3.1.4 Libraires :

En langage Python, une librairie est un ensemble de fonctions, de classes d'objets et de constantes qui permettent de travailler sur un thème particulier. Il existe de très nombreuses bibliothèques Python, et c'est pour cela que c'est le langage de programmation le devenu très populaire parmi les programmeurs.[31]

Voici les plus importantes libraires utilisées dans notre travail :

- **NumPy** :Le terme NumPy est en fait l'abréviation de « Numerical Python ». Il s'agit d'une bibliothèque opensource en langage Python. On utilise cet outil pour la programmation scientifique en Python, et notamment pour la programmation en Data Science, pour l'ingénierie, les mathématiques ou la science.[32]

On utilisera cette librairie dans la manipulation des vecteurs solution.

- **Time** : La bibliothèque "time" fournit de nombreuses façons de représenter le temps dans le code, comme les objets, les nombres et les chaînes de caractères. Il fournit également des fonctionnalités autres que la représentation du temps, comme l'attente pendant l'exécution du code et la mesure de l'efficacité de votre code.[33]
Dans notre cas, on utilisera cette librairie pour déterminer le temps total d'apprentissage du modèle.
- **Random** : Le module « Random » est un module intégré à Python qui sert à générer des nombres aléatoires pour vérifier les différentes conditions dans le code.[34]
- **Vrep** : Ce module « Vrep » permet d'envoyer et de recevoir des commandes de/à Vrep (CoppeliaSim). Il permet aussi de commander des scènes sur le simulateur avec le langage Python, C++ ou bien JAVA au lieu du langage « Lua » qui est un langage très peu utilisé.[35]
Dans notre cas, nous utiliserons cette librairie pour commander le commencement et fin de simulation, ainsi pour la transmission des données entre les deux programmes Python et CoppeliaSim.
- **Matplotlib.pyplot** : « matplotlib.pyplot » est une collection de fonctions qui font fonctionner « matplotlib » comme MATLAB. Chaque fonction « pyplot » apporte des modifications à une figure. Pour notre travail, elle va servir à créer une représentation graphique du développement des résultats obtenus durant l'apprentissage du GA.[36]
- **Importation des fichiers** : L'utilisation des fonctions et variables d'un script Python dans un autre code est possible avec l'importation des fichiers Python sous la syntaxe montrée dans la **figure III.1**.

III.3.2. Connexion API Python/ CoppeliaSim :

Le développement d'un modèle d'optimisation robotique nécessite un échange de données entre le simulateur dynamique « CoppeliaSim » et le modèle d'optimisation codé en Python. Pour faire cela, on a opté à établir une connexion entre les deux logiciels pour ensuite adapter ces deux programmes à notre travail.

Avant de commencer la programmation qui relie l'algorithme d'optimisation et le simulateur, on a établi une connexion du type « Interface De Programmation d'Application » avec le module API Python. On a opté vers cette méthode de connexion car le logiciel de simulation offre cette option avec la disponibilité de l'API Python intégré en ce dernier.

III.3.2.1 Création De La Connexion API :

La création et la mise en œuvre de cette connexion nécessitent quelques bibliothèques et libraires installées sur Python. Ces libraires vont permettre de lancer et recevoir des commandes.

```
6 import sim
7 import vrep
```

Figure III. 1. Importation des librairies de connexion API

L'établissement d'une connexion API entre les deux logiciels se fait en deux étape :

1. Importation des librairies
2. Établir une connexion avec la fonction API_cnx

```
class Kuka(object):
    def init(self):
        self.clientID= None

    def API_cnx(self):
        sim.simxFinish(-1)

        self.clientID=sim.simxStart('127.0.0.1',19997,True,True,5000,5)

        if self.clientID== -1:
            print('Could not connect')
        else:
            print("API Connexion Established ")
        return
```

Test de fonctionnement

Figure III. 2. Script Connexion API

La ligne de commande self.clientID contient plusieurs paramètres comme le montre la **Figure III.3** suivante :

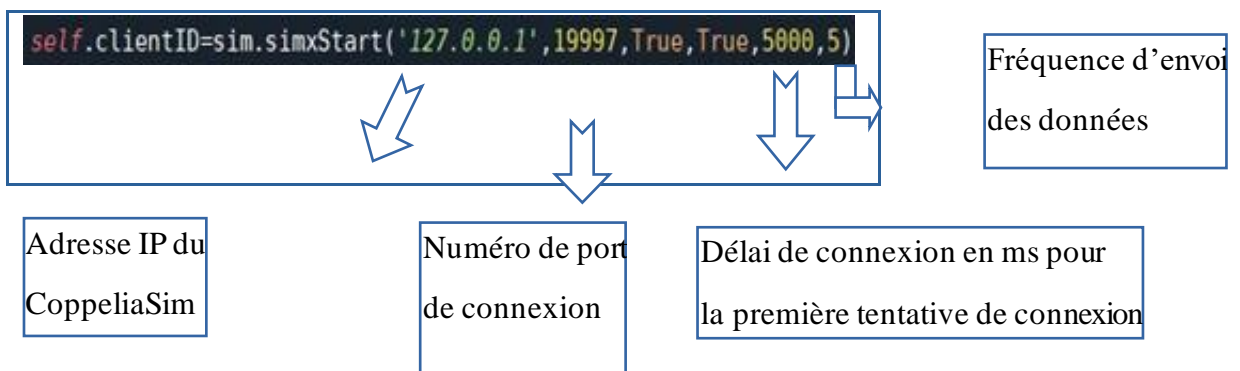


Figure III. 3. ligne de commande self.clientID

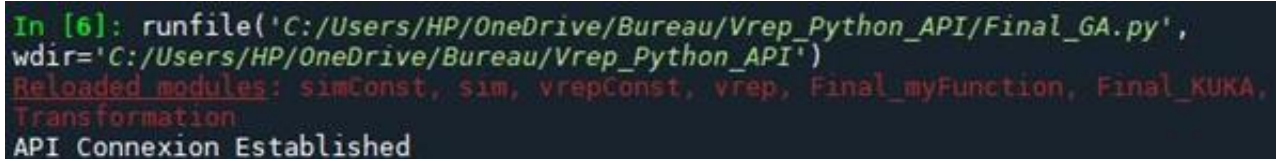
L'avantage de travailler avec le logiciel CoppeliaSim est que ce dernier contient un API Python intégré. Donc la connexion se fait par l'ouverture d'un port de connexion dans sur un seul logiciel est non pas les deux (Python et CoppeliaSim). Dans notre cas, on a choisi de le faire sur Python.

A noter que dans le clientID = 1 dans le cas de bon établissement de connexion.

III.3.2.2 Test De Fonctionnement :

Afin de déterminer si la connexion API est établie ou bien échouée, nous avons créé un test de fonctionnement qui consiste à retourner un message après la tentative de connexion. Ce message pourrait être une de validation « API Connexion Established » ou bien d'échec « Could not Connect ».

Voici le résultat obtenu après la tentative de connexion :



```
In [6]: runfile('C:/Users/HP/OneDrive/Bureau/Vrep_Python_API/Final_GA.py',  
wdir='C:/Users/HP/OneDrive/Bureau/Vrep_Python_API')  
Reloaded modules: simConst, sim, vrepConst, vrep, Final_myFunction, Final_KUKA,  
Transformation  
API Connexion Established
```

Figure III. 5. Connexion API établie

Ce message montre que la connexion entre les deux logiciels est établie.

III.3.3. Programmation Algorithme Génétique :

L'algorithme génétique passe par plusieurs étapes avant de donner une solution finale. La programmation de l'algorithme génétique passe par plusieurs étapes car chaque étape nécessite une programmation particulière ainsi que son propre codage.

Remarque : notre code est inspiré du travail développé par « Eric STOLTZ » [37], dans le but de résoudre un problème TSP (problème du voyageur de commerce). Le code a été adapté puis modifié pour notre travail.

Voici le pseudo-code de notre algorithme génétique :

Début

Génération de la population initial

Déterminer la valeur fitness de chaque individu

Répéter

 Sélection

 Permutation

 Mutation

 Déterminer la valeur fitness de chaque
individu

Jusqu'à atteindre le nombre de génération introduit

Fin

III.3.3.1. Début de Simulation :

On lance la simulation à partir du troisième script intitulé de « **Final_KUKA** ».

Ce script est responsable sur :

- Lancement et arrêt de simulation sur CoppeliaSim.
- Exécution du code d'apprentissage GA.

Voici le script :

```

class Kuka(object):
    def init(self):
        self.clientID= None

    def API_cnx(self):
        sim.simxFinish(-1)
        self.clientID=sim.simxStart('127.0.0.1',19997,True,True,5000,5)

        if self.clientID== -1:
            print('Could not connect')

        else:
            print("API Connexion Established ")
            return

    def Start_Simulation(self):
        vrep.simxStopSimulation(self.clientID, vrep.simx_opmode_oneshot)
        vrep.simxStartSimulation(self.clientID, vrep.simx_opmode_oneshot)

        return

    def Stop_Simulation(self):

        time.sleep(0.1)
        vrep.simxStopSimulation(self.clientID, vrep.simx_opmode_oneshot)
        time.sleep(0.1)
        vrep.simxStartSimulation(self.clientID, vrep.simx_opmode_oneshot)

def main(v):
    robot = Kuka()
    robot.API_cnx()
    fit = Final_myFunction.myFunction(v)
    robot.Stop_Simulation()

    return fit

```

Figure III. 3 . Script Final_KUKA

Commentaire :

La classe « KUKA » contient les fonctions utilisées pour commander le robot sur le simulateur à partir du script Python.

Les fonctions « Start_Simulation » et « Stop_Simulation » servent à démarrer et arrêter la simulation respectivement.

La fonction « main » rassemble et gère l'appel des différentes fonctions dans l'ordre d'exécution. Cette fonction retourne la variable « fit » d'un individu de la population.

III.3.3.2. Initialisation du code GA :

Le modèle d'optimisation développé va utiliser les coordonnées des points déjà pris sur le robot réel en utilisant le travail [38]. Son but est de déterminer un ordonnancement entre les points, ainsi que deux autres vecteurs appelé vecteur vitesse et vecteur accélération vont être créé en prenant des échantillons des deux vecteurs d'initialisation. « CityList2 » et « cityList3 »

La liste des coordonnées des points est appelée « CityList ».

La liste des coefficients du vecteur vitesse est appelée « cityList2 ».

La liste des coefficients du vecteur accélération est appelée « cityList3 ».

Voici le code d'initialisation :

```

" Initialisation Liste des vecteurs"

"Test pour 7 Points aléatoire"
p1=[-103.88 ,50 ,1.26 , -92.32 , -1.45 ,37.67 ,4.4 ]
p2=[-7.27 ,36.23 ,1.26 , -55.72 , -0.66 ,88.05 , -4.55 ]
p3=[116.22 ,34.79 ,1.26 , -87.50 , -0.78 ,57.66 ,50.49 ]
p4=[-40.10 ,53.34 ,1.26 , -66.46 , -1.09 ,60.27 ,37.13 ]
p5=[-56.97 ,28.88 ,1.26 , -40.06 , -0.6 ,111.0 ,50.24 ]
p6=[-144.61 ,42.07 ,1.26 , -48.58 , -0.78 ,89.42 , -37.34 ]
p7=[-21.31 ,34.79 ,1.26 , -87.51 , -0.79 ,57.66 ,50.49 ]

cityList = [p1,p2,p3,p4,p5,p6,p7]

# Velocity vecotr
cityList2 = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]

# Acceleration vector
cityList3= [0.5,0.7,0.9,1,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2]

```

1

2

3

Figure III. 4. Initialisation des vecteurs

1 : Coordonnées des points initiales

2 : liste des coefficients du vecteur vitesse

3 : liste des coefficients du vecteur accélération

On note qu'au moment de l'exécution du modèle d'optimisation, on introduit le nombre d'individus d'une génération ainsi que le nombre de générations et la probabilité de mutation.

III.3.3.3. Génération de Population initial :

La création de la population initiale passe par trois étapes :

1. Création d'un vecteur position, en permutant les positions des points présents sur le vecteur « cityList ».
2. La création d'un vecteur vitesse qui a la même taille que le vecteur position et qui doit ne contenir que les valeurs présentes sur le vecteur « cityList2 ».
3. La création d'un vecteur accélération qui a la même taille que le vecteur position et qui doit ne contenir que les valeurs présentes sur le vecteur « cityList3 ».

Voici le code utilisé pour cette tâche :

```
""" Création de population """
def createRoute(cityList):
    route = random.sample(cityList, len(cityList)) # Vecteur des points
    Route = random.choices(cityList2, k = len(route)) # Vecteur vitesse
    RRoute = random.choices(cityList3, k = len(route)) # Vecteur acceleration
    route.append(Route)
    route.append(RRoute)
    global lengthRoute
    lengthRoute = len(Route)

    # Envoyer le chemin à vrep pour le simuler
    return route
```

Figure III. 5. Création d'individu

Le vecteur route(chemin) contient les trois vecteurs : point, vitesse et accélération.

Mais la fonction « CreateRoute » ne produit qu'un seul individu. Nous avons donc créé la fonction « InitialPopulation » pour avoir autant d'individus que nous voulons pour notre population.

```
def initialPopulation(popSize, cityList):
    population = []

    for i in range(0, popSize):
        population.append(createRoute(cityList))
    return population
```

Figure III. 6. Génération d'une population de taille n

Le nombre d'individus est appelé « popSize ».

III.3.3.4. Fitness :

Chaque fois qu'un individu est créé, on doit lui attribuer une valeur fitness. Cette valeur sera plus tard calculée par le simulateur « CoppeliaSim ».

Cette procédure passe par 3 étapes :

1. Envoie du vecteur chemin (vecteur route) au simulateur.
2. Simulation et récupération des paramètres T (temps d'exécution de tâche) et E (énergie consommée).

3. Transfert des paramètres T et E à l'algorithme génétique.
4. Calcul de la valeur fitness en utilisant la somme pondérée de T et E.

Voici le code utilisé :

Pour l'envoi du vecteur chemin, on utilise la commande « `simxCallScriptFunction` ». Cette commande permet d'envoyer et de recevoir des données entre le simulateur « Coppeliasim » et Python.

On utilise la même commande pour recevoir les paramètres T et E du simulateur.

Pour le code de transfert de données.



```

class Fitness1:
    def __init__(self, route):
        self.route = route
        self.fit = 0.0
        self.fitness = 0.0

    def CalculFit(self):
        Fit = 0.0
        P = self.route
        v = P
        print("L'algorithme a donné le résultat suivant: ", v )

        Fit = Final_KUKA.main(v)
        print ("Fitness = ",Fit)

        Fitt = 0.5 * Transformation.TimeTransfo(Fit[0]) + 0.5 * Transformation.EnergyTransfo(Fit[1])

        self.fit = Fitt
        return self.fit

    def routeFitness(self):
        if self.fitness == 0:
            self.fitness = 1 / float(self.CalculFit())
        print('self.fit = ',self.fitness)
        return self.fitness

```

The image shows a code editor with Python code for a `Fitness1` class. Two blue arrows point from the code to numbered boxes on the right. Arrow 1 points to the line `Fit = Final_KUKA.main(v)`. Arrow 2 points to the line `Fitt = 0.5 * Transformation.TimeTransfo(Fit[0]) + 0.5 * Transformation.EnergyTransfo(Fit[1])`.

Figure III. 7. Classe fitness

- 1 : Récupération des paramètres E et T sur le code GA
- 2 : Somme pondéré.

Nous avons créé une classe « `Fitness1` » pour calculer la valeur fitness de chaque individu créé. Nous traiterons le fitness comme l'inverse de la somme pondérée « `Fitt` ». Nous voulons minimiser cette somme `Fitt`, donc un score de fitness plus élevé est préférable.

Une fois le calcul de fitness d'une génération est terminé, l'évolution commence.

Pour commencer l'évolution, nous pouvons utiliser la valeur Fitt pour faire un classement des individus de la population. Notre sortie sera une liste dans l'ordre décroissant, contenant les ID des vecteurs individus avec le score de fitness associé comme montré dans la **Figure III.10**.

On utilise la fonction « rankRoutes » pour le travail de classement.

Voici le résultat obtenu :

```
Classement des indiviuds = [(1, 2.2657440193501452), (2, 2.154477661845596), (3, 1.8765820240256774), (4, 1.7663524348612556), (0, 1.599683512615051)]
```

Figure III. 8. Classement des Individus (identifiant d'individu, score fitness)

III.3.3.5. Sélection

Dans cette partie, nous allons créer le groupe d'accouplement en deux étapes. Tout d'abord, nous utilisons la sortie de la fonction « rankRoutes » pour déterminer les individus à sélectionner dans notre fonction de sélection. Ensuite, nous mettons en place une roulette en calculant un poids de fitness relatif pour chaque individu. Enfin, nous comparons un nombre tiré au hasard à ces poids pour sélectionner notre groupe d'accouplement.

Nous voudrions également conserver nos meilleurs individus, c'est pourquoi nous introduisons l'élitisme.

À la fin, la fonction de sélection renvoie une liste d'identifiant d'individus, que nous pouvons utiliser pour créer le groupe d'accouplement dans la fonction « matingPool ». Pour le code de la sélection. (Voir annexe 2)

Maintenant que nous avons les identifiants des individus qui constitueront notre groupe d'accouplement à partir de la fonction de sélection, on crée le groupe d'accouplement.

Nous allons simplement extraire les individus sélectionnés de notre population.

Voir la **figure III.11**.

```
def matingPool(population, selectionResults):  
    matingpool = []  
    for i in range(0, len(selectionResults)):  
        index = selectionResults[i]  
        matingpool.append(population[index])  
    return matingpool
```

Figure III. 9. Création de groupe d'accouplement

III.3.3.6. Permutation :

Dans cette partie, nous allons utiliser deux méthodes différentes pour la permutation de vecteur point et vecteur accélération/vitesse.

Pour le vecteur points, comme notre problème est unique dans le sens où nous devons impérativement inclure tous les points exactement une fois, afin de respecter cette règle, nous avons créé une fonction de reproduction « **breed** ». Dans la fonction « **breed** », nous sélectionnons aléatoirement un sous-ensemble du premier vecteur parental, puis nous remplissons le reste de l'individu avec les points du second parent dans l'ordre dans lequel ils apparaissent, sans dupliquer aucun gène dans le sous-ensemble sélectionné du premier parent. Voir la **figure III.12.** :

```
def breed(parent1, parent2):
    child = []
    childP1 = []
    childP2 = []
    P1 = parent1
    P2 = parent2

    parent1 = parent1[:len(parent1)-2]
    parent2 = parent2[:len(parent2)-2]

    geneA = int(random.random() * len(parent1))
    geneB = int(random.random() * len(parent1))

    startGene = min(geneA, geneB)
    endGene = max(geneA, geneB)

    for i in range(startGene, endGene):
        childP1.append(parent1[i])

    childP2 = [item for item in parent2 if item not in childP1]
    child = childP1 + childP2          # Permutation des points

    x = PREBREED_Vitesse(P1,P2)       # Permutation Vitesse
    y = PREBREED_Accel(P1,P2)         # Permutation Accelération
    child.append(x)
    child.append(y)
    return child
```

Figure III. 10. Code de Permutation

Pour les vecteurs accélération et vitesse, nous utilisons une autre méthode de permutation qui consiste à diviser les deux vecteurs parents. Nous sélectionnons un sous-ensemble de la première chaîne parentale, puis nous remplissons le reste de l'individu avec les gènes du second parent dans l'ordre dans lequel ils apparaissent. Voir la figure suivante :


```
def PREBREED_Accel(parent1,parent2):
    BB = parent1[len(parent1)-1 :len(parent1) ]
    BB = list(flatten(BB))

    AA = parent2[len(parent2)-1 : len(parent2)]
    AA = list(flatten(AA))

    kid = []

    index = int(len(BB)/2)

    for i in range(index):
        kid.append(BB[i])

    for i in range (index,len(AA)):
        kid.append(AA[i])
        if len(kid) > len(AA):
            break

    return kid
```

```
def PREBREED_Vitesse(parent1,parent2):
    B = parent1[len(parent1)-2 :len(parent1)-1 ]
    B = list(flatten(B))

    A = parent2[len(parent2)-2 : len(parent2)-1]
    A = list(flatten(A))

    KID = []

    index = int(len(parent1)/2)

    for i in range(index):
        KID.append(B[i])

    for i in range (index,len(A)):
        KID.append(A[i])
        if len(KID) > len(A):
            break

    return KID
```

Figure III. 11. permutation vecteur vitesse et accélération

Ensuite, nous allons généraliser cela pour créer notre population de descendants. Nous utilisons l'élitisme pour retenir les meilleurs individus de la population actuelle. Après, nous utilisons les fonctions mentionnées dans ce titre de permutation pour remplir le reste de la prochaine génération.

III.3.3.7 Mutation :

Comme l'étape précédente, on utilise deux méthodes de mutation pour chaque individu.

Pour le vecteur points, comme nous devons respecter la règle qui cite que nous ne pouvons pas supprimer des individus. À la place, nous allons utiliser la mutation par échange où on introduit une faible probabilité, deux individus vont échanger leur place dans notre vecteur.

Le code est présenté dans la **Figure III.15** :

```
def mutate(individual, mutationRate):

    Avector = individual[len(individual)-1:len(individual)] # Acceleration
    VVector = individual[len(individual)-2:len(individual)-1] # Velocity

    individual = individual[:len(individual)-2]

    for swapped in range(len(individual)):
        if(random.random() < mutationRate):
            swapWith = int(random.random() * len(individual))

            city1 = individual[swapped]
            city2 = individual[swapWith]

            individual[swapped] = city2
            individual[swapWith] = city1

    x = list(flatten(Vmutate(VVector,mutationRate))) # Vecteur vitesse après mutation
    y = list(flatten(Amutate(Avector,mutationRate))) # Vecteur accélération après mutation
    individual.append(x)
    individual.append(y)

    return individual # Vecteur chemin après mutation
```

Figure III. 12. Mutation du vecteur points

Pour les vecteurs vitesse/accélération, la mutation consisterait simplement à attribuer une faible probabilité à chaque élément du vecteur et d'augmenter ou diminuer sa valeur par 0.1.

Le code est présenté dans la figure suivante :

```
def Amutate(vector,mutationRate):
    vector = list(flatten(vector))
    for i in range (lengthRoute):

        if random.random() < mutationRate:
            if random.random() < 0.5:
                vector[i] += 0.1
            else:
                vector[i] -= 0.1
            while True:
                if vector[i] > 2:
                    vector[i] = 2
                    break
                elif vector[i] < 0.5:
                    vector[i] = 0.5
                    break
            else:
                break
```

```
def Vmutate(V,mutationRate):
    V = list(flatten(V))
    for i in range (lengthRoute):

        if random.random() < mutationRate:
            if random.random() < 0.5:
                V[i] += 0.1
            else:
                V[i] -= 0.1
            while True:
                if V[i] > 1:
                    V[i] = 1
                    break
                elif V[i] <= 0:
                    V[i] = 0.1
                    break
            else:
                break

    return V
```

Figure III. 13. Mutation vecteur vitesse et accélération

Le test de mutation est le même pour tous les vecteurs, il consiste à générer une valeur aléatoire entre 0 et 1. Si cette valeur vérifie la condition de mutation, la mutation se produit.

Ensuite, nous devons étendre la fonction de mutation à toute la population en introduisant la fonction « **mutatePopulation** » comme le montre **Figure III.16**.

```
def mutatePopulation(population, mutationRate):  
    mutatedPop = []  
  
    for ind in range(0, len(population)):  
        mutatedInd = mutate(population[ind], mutationRate)  
        mutatedPop.append(mutatedInd)  
    return mutatedPop
```

Figure III. 14. Mutation de toute la population

III.3.3.8 Prochaine Génération :

En rassemblant toutes ces fonctions précédentes, on crée une fonction qui produit une nouvelle génération.

Tout d'abord, nous classons les individus dans la génération actuelle en utilisant la fonction « **rankRoutes** ». Nous déterminons ensuite les parents potentiels en exécutant la fonction de sélection, ce qui nous permet de créer le groupe d'accouplement à l'aide de la fonction « **matingPool** ». Enfin, nous créons notre nouvelle génération à l'aide de la fonction « **breedPopulation** », puis nous appliquons la mutation à l'aide de la fonction « **mutatePopulation** ». Le code est présenté dans la **Figure III.17** :

```
def nextGeneration(currentGen, eliteSize, mutationRate):  
    popRanked = rankRoutes(currentGen)  
    selectionResults = selection(popRanked, eliteSize)  
    matingpool = matingPool(currentGen, selectionResults)  
    children = breedPopulation(matingpool, eliteSize)  
    nextGeneration = mutatePopulation(children, mutationRate)  
    return nextGeneration
```

Figure III. 15. Script de l'évolution

III.3.3.9. Evolution De l'Algorithme Génétique :

Nous avons enfin toutes les pièces en place pour lancer l'évolution; Avec la population initiale déjà initialisée, nous pouvons passer en boucle le nombre de générations désiré. Pour voir combien nous nous sommes améliorés, nous enregistrons le meilleur chemin ainsi que les

meilleures valeurs de fitness de chaque génération dans le vecteur « **progress** ». Le code est montré sur la **Figure III.18**.

Nous voudrions aussi visualiser l'amélioration de la valeur fitness, la meilleure façon de le faire est de tracer un graphe contenant les meilleures valeurs de chaque génération.

```
def geneticAlgorithmPlot(population, popSize, eliteSize, mutationRate, generations):
    pop = initialPopulation(popSize, population)
    print("Initial distance: " + str(1 / rankRoutes(pop)[0][1]))
    progress = []
    progress.append(1 / rankRoutes(pop)[0][1])

    for i in range(0, generations):
        print("progress [" + str(i) + "] = ", progress)
        pop = nextGeneration(pop, eliteSize, mutationRate)
        progress.append(1 / rankRoutes(pop)[0][1])

    print("progress final =", progress)
    plt.plot(progress)
    plt.ylabel('Fitness')
    plt.xlabel('Generation')
    plt.show()

    print("Final distance: " + str(1 / rankRoutes(pop)[0][1]))
    bestRouteIndex = rankRoutes(pop)[0][0]
    bestRoute = pop[bestRouteIndex]  # best route to take
    print("best résultat est ", bestRoute)
    print("simulation time = ", time.time() - start_time)
```

Figure III. 16. Script de fonction d'exécution

Afin de connaître le temps de simulation, on calcule la différence entre le temps de simulation actuelle moins le temps initial pris au début de simulation.

L'évolution de l'algorithme génétique se termine une fois le nombre de générations désiré est atteint.

III.4 CoppeliaSim

Le simulateur robotique CoppeliaSim, avec environnement de développement intégré et moteur dynamique, est basé sur une architecture de contrôle distribuée : chaque objet/modèle peut être contrôlé individuellement via un script intégré, des nœuds ROS/ROS2, des clients API distants ou une solution personnalisée.

Cela rend CoppeliaSim très polyvalent et idéal pour les applications multi-robots. Les contrôleurs peuvent être écrits en C/C++, Python, Java, Lua, Matlab ou Octave. [38]

Voici quelques-unes des applications de CoppeliaSim :

- Simulation de systèmes d'automatisation industrielle
- Surveillance à distance
- Contrôle matériel
- Prototypage et vérification rapides
- Développement rapide d'algorithmes
- Formation liée à la robotique
- Présentation du produit

CoppeliaSim peut être utilisé comme une application autonome ou peut facilement être intégré dans une application cliente principale : son faible encombrement et son API élaborée font de CoppeliaSim un candidat idéal à intégrer dans des applications de haut niveau. L'interpréteur de script Lua ou Python fait de CoppeliaSim une application extrêmement polyvalente, laissant à l'utilisateur la liberté de combiner les fonctionnalités de bas/haut niveau pour obtenir de nouvelles fonctionnalités de haut niveau.[38]

III.4.1 Caractéristiques principales de CoppeliaSim

CoppeliaSim possède plusieurs Caractéristiques nous allons citer quelques-unes :

- **5 Approches de programmation**

Simulateur et simulations sont entièrement personnalisables, avec 5 approches de programmation qui sont mutuellement compatibles et qui peuvent même travailler main dans la main. 6 langages de programmation supportés.[39]

- **API puissantes, 7 langues**

API régulière : Python, Lua et C/C++

API distante : (C/C++, Python, Java, JavaScript, Matlab & Octave. Cela permet de Contrôler une simulation ou le simulateur lui-même à distance (par exemple à partir d'un vrai robot ou d'un autre PC)

Interfaces ROS : éditeurs, abonnés et appels de service. Prise en charge de tous les messages standard.[39]

- **Dynamique/Physique**

4 moteurs physique (Bullet Physics, ODE, Vortex Studio et Newton Dynamics) pour des calculs dynamiques rapides et personnalisables, pour simuler la physique du monde réel et les interactions d'objets (réponse de collision, saisie, etc.).[39]

- **Direct et inverse cinématique**

- **Particules Dynamiques**

CoppeliaSim prend en charge des particules personnalisables qui peuvent être utilisées pour simuler des jets d'air ou d'eau, des moteurs à réaction, des hélices, etc.[39]

- **Détection De Collision, Calcul De La Distance Minimale, Simulation de capteur de proximité et de vision**
- **Planification de trajectoire/mouvement**

La figure III.21 représente une vue d'ensemble du simulateur CoppeliaSim.

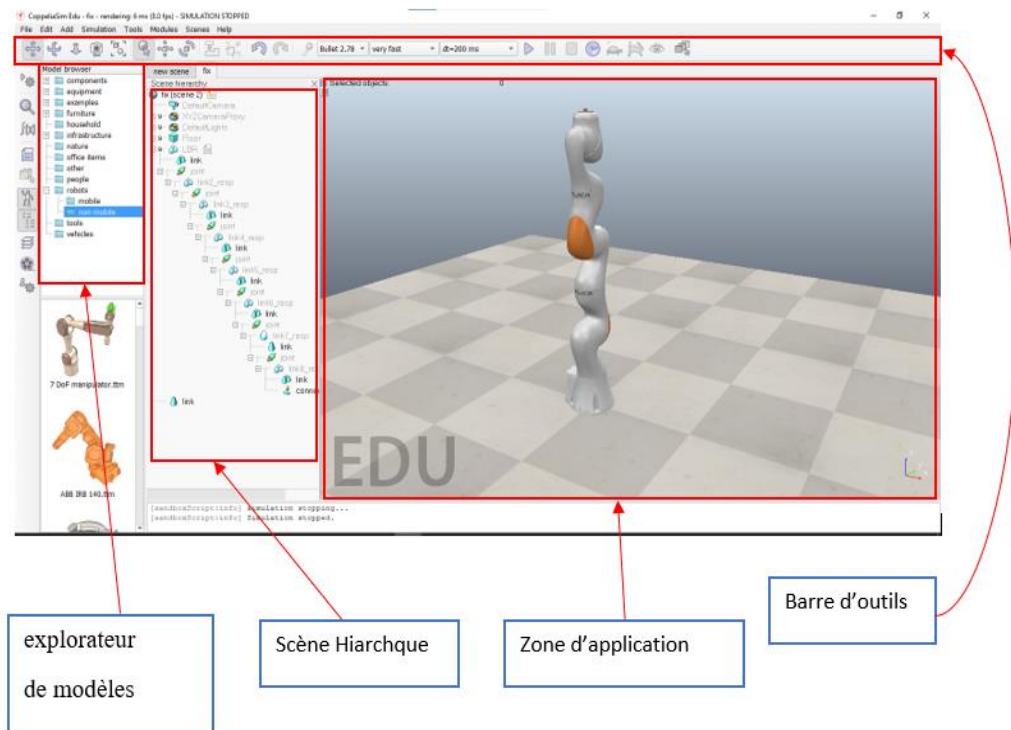


Figure III. 17. Vue du simulateur CoppeliaSim

III.4.2 simulation de notre travail avec CoppeliaSim

Dans notre travail nous avons utilisé le langage python pour compiler l'algorithme génétique et envoyer à chaque fois les nouvelles routes pour les simuler et récolter le temps et l'énergie déposée par chaque axe durant tout le parcours.

Cependant, nous avons décidé d'utiliser le langage qu'utilise CoppeliaSim qui est Lua, cela pour pouvoir calculer l'énergie et le temps de chaque parcours.

Pour calculer le temps de parcours de chaque route nous avons utilisé la fonction « `sim.getSimulationTime()` » avec l'unité seconde.

Pour calculer l'énergie dépensée par chaque axe tout au long du parcours de chaque route nous avons utilisé la fonction « `sim.getJointForce` ». cette fonction Récupère la force ou le couple appliqué à une articulation le long/autour de son axe actif avec l'unité n.m(Newton. Mètre).

III.4.3 explication du code Lua

- **La fonction `my_Function`**

Cette fonction sert à recevoir le vecteur chemin. Ce vecteur est composé des trois vecteurs points, vitesse et accélération.

- **La fonction `movecallback`**

Cette fonction sert à calculer la force applique sur chaque articulation tout au long du parcours en utilisant la commande « `sim.getJointForce` ». Le calcul de l'énergie se fait instant par instant.

- **La fonction `courtinemain`**

Cette fonction est responsable sur simulation de l'individu créé par le modèle d'optimisation ainsi le calcul du temps d'exécution.

- **La fonction `vrep_Python`**

Cette fonction a le rôle de retourner à l'algorithme génétique le temps de cycle et l'énergie consommé par le robot durant cette tâche.

III.5 Conclusion

Dans ce chapitre nous avons vu un aperçu sur les outils utilisés dans l'implantation de notre modèle d'optimisation qui est basé sur l'algorithme génétique.

La combinaison entre le langage de programmation python et le logicielle de simulation « Coppeliasim » pour implémenter le modèle d'optimisation, nous offre un grand avantage, grâce aux nombreux caractéristiques que possède le logiciel « Coppeliasim » et la fluidité du langage python.

Dans le prochaine chapitre, nous allons exposer les résultats obtenu après la simulation en suivant différents protocoles expérimentale.

Chapitre IV

Présentation et interprétation des résultats

IV.1. Introduction

Dans ce chapitre, nous allons présenter les résultats obtenus en utilisant l'application développée au sein de ce projet.

La présentation des résultats suivra un protocole d'expérimentation qui a été fait pour bien montrer les résultats finaux de ce travail.

Le protocole d'expérimentation suivi se divise en trois parties :

1. La première partie est consacrée à la validation du modèle conçu, et voir si le modèle fonctionne comme il était prévu ou pas. On essaiera de trouver un vecteur chemin optimal.
2. Dans la deuxième partie nous fixerons le vecteur point pour voir l'impact des vecteurs vitesse et accélération sur le temps et l'énergie consommés par le robot.
3. Dans la troisième partie du protocole, nous allons varier les instances de pondérations pour voir si on peut favoriser l'optimisation de l'énergie ou du temps.

Pour les tests effectués, nous avons utilisé quatre différents ensembles de points. Ces ensembles contiennent des configurations de la position de l'outil de travail du robot. Les points ont été pris d'une façon aléatoire dans l'espace du travail du robot.

Pour les paramètres de l'algorithme génétique on a fixé le nombre de générations à 10, la taille de population à 15, la taille de l'élitisme à 2 et la probabilité de mutation à 0,02.

Les coefficients de pondération sont fixes à 0.5 .

IV.2. Présentation et interprétations des résultats

Dans ce titre nous allons présenter les résultats obtenus puis les interpréter en suivant l'ordre du protocole d'expérimentation :

IV.2.1. Validation du Modèle

Notre problème d'optimisation est un problème de minimisation ; Ça veut dire que notre but est la diminution de la valeur de la fonction coût qui est dans notre cas la variable « **Fitness** ». Ceci va permettre de valider le bon fonctionnement de notre modèle d'optimisation.

Dans cette partie nous avons calculé le meilleur vecteur chemin à prendre pour 4 ensembles de points différents. Voici les résultats obtenus :

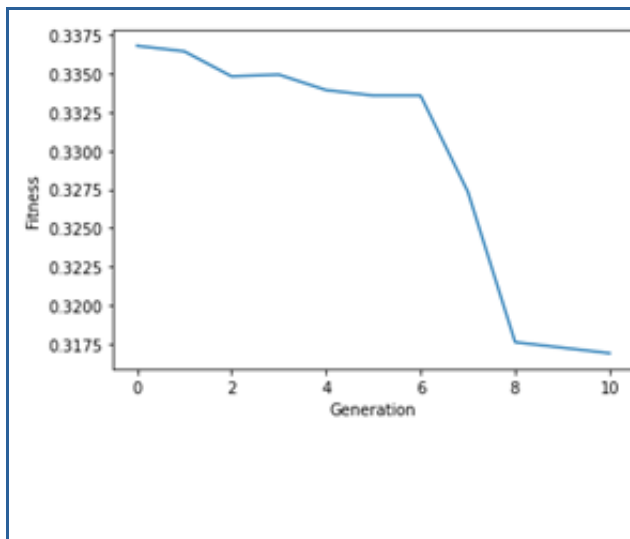


Figure IV. 1. Evolution de Fitness pour l'ensemble de 7 points aléatoires

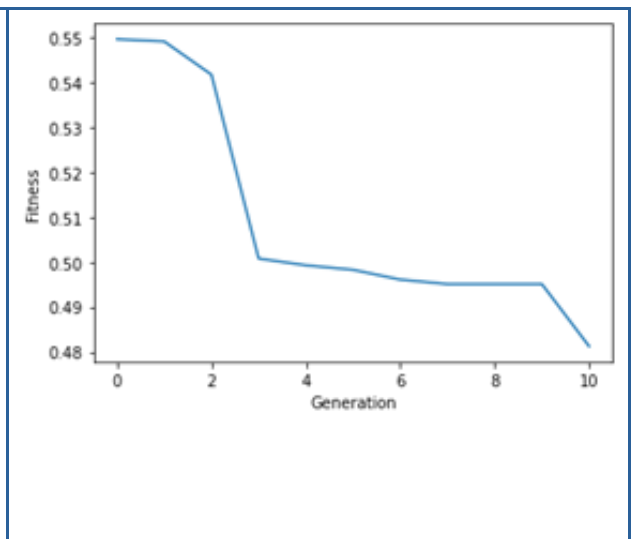


Figure IV. 2. Evolution de Fitness pour l'ensemble de 10 points aléatoires

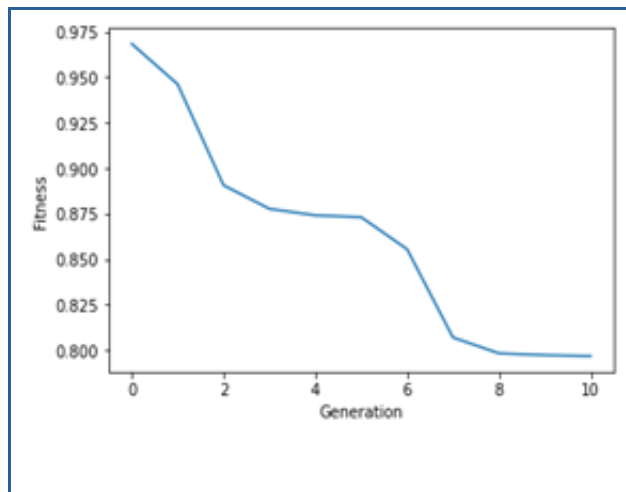


Figure IV. 3. Evolution de Fitness pour l'ensemble de 15 points aléatoires

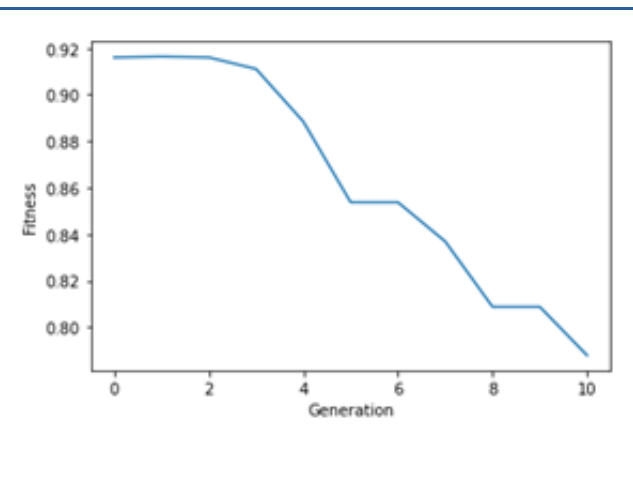


Figure IV. 4. Evolution de Fitness pour l'ensemble de 20 points aléatoires

Interprétation :

Les **figures IV.1, 2, 3 et 4** représentent l'évolution des meilleures valeurs fitness par generation. On remarque dans les quatre figures précédentes que les courbes sont décroissantes. Ça veut dire que la fonction coût (Fitness) a diminué, ce qui indique la diminution de l'énergie et du temps consommé par cycle entre la première et la dernière génération, car la fonction coût est la somme pondérée du temps et de l'énergie consommée.

Le modèle d'optimisation a donné une meilleure solution que celle trouvée initialement pour les quatre ensembles utilisés (génération 0).

Conclusion

Le modèle d'optimisation a donné de bons résultats pour les ensembles de points aléatoirement prédéfinis. On peut donc conclure la validité de ce modèle d'optimisation.

IV.2.2 Résultats obtenue avec un vecteur points fixe

Dans cette partie, nous allons fixer le vecteur point, et varier les deux vecteurs vitesses et accélération pour voir leur impact sur l'évolution des coûts énergétiques et temporels.

Nous allons effectuer l'expérience sur 3 ensembles de points.

On note aussi que les vecteurs points choisis sont tous des vecteurs optimaux prédéterminés par le modèle d'optimisation. Donc nous allons essayer d'optimiser un vecteur déjà optimal et voir si on peut aller plus loin.

IV.2.2.1 Ensemble de 7 points

Voici les résultats obtenus :

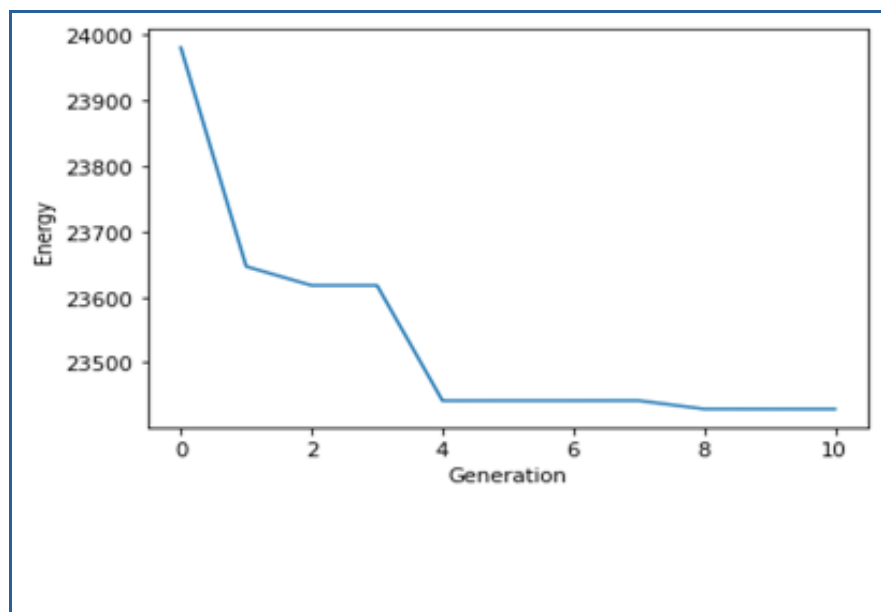


Figure IV. 5. Evolution de l'énergie avec vecteur point fixe pour un ensemble de 7 points

Interprétation

La **figure IV.5** représente l'évolution de l'énergie consommée par le robot pour un vecteur point constant.

Initialement, l'énergie consommée pour ce travail était **23980.5117 N.m**. Après l'optimisation l'énergie dissipée est devenue **23428.9394 N.m**.

On remarque une diminution dans l'énergie consommée avec une différence de **551.5723 N.m**

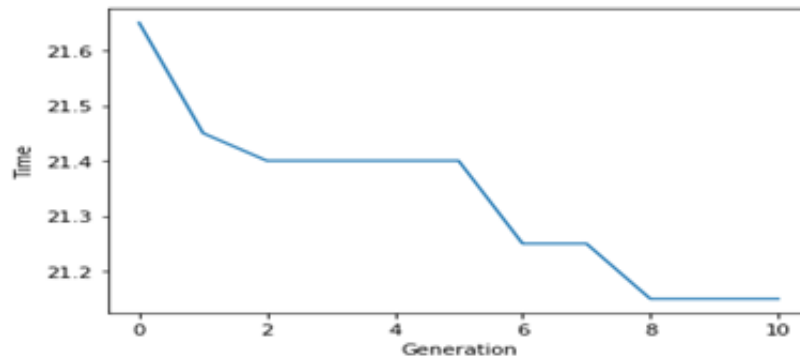


Figure IV. 6. Evolution du temps avec vecteur point fixe pour un ensemble de 7 points

Interprétation

La **figure IV.6** représente l'évolution du temps de cycle de la tâche pour un vecteur point constant.

Initialement, le temps d'exécution pour cette solution était **24.65 sec**. Après l'optimisation le temps de cycle est devenu **21.15 sec**.

On remarque que le temps d'exécution est diminué par 3.5sec ce qui est significatif dans le domaine industriel.

Conclusion

Pour cet ensemble de points, on conclut que l'application d'optimisation a donné de bons résultats pour une optimisation avec un vecteur de points fixe. On dira aussi que la variation de l'accélération et la vitesse maximale impacte le temps de cycle et l'énergie consommée.

IV.2.2.2 Ensemble de 15 points

Voici les résultats obtenus :

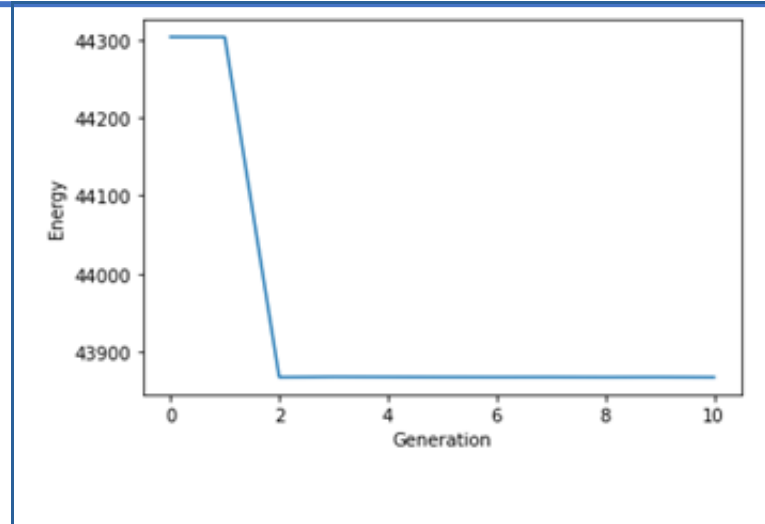


Figure IV. 7. Evolution de l'énergie avec vecteur point fixe pour un ensemble de 15 points

Interprétation

La **figure IV.7** représente l'évolution de l'énergie dissipée dans un cycle pour un vecteur point fixe.

Initialement, l'énergie consommée était **44300 N.m** . Après l'optimisation, elle est devenue **43866N.m**.

On remarque une amélioration de **434 N.m**.

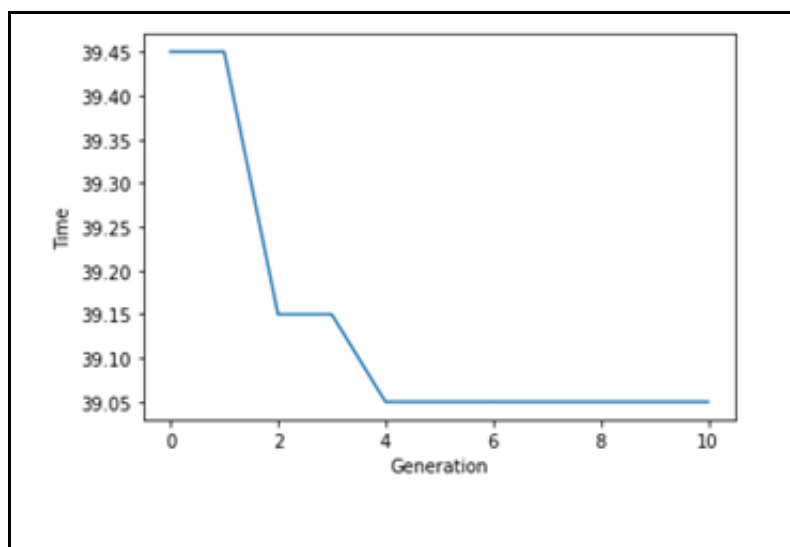


Figure IV. 8. . Evolution du temps avec vecteur point fixe pour un ensemble de 15 points

Interprétation

La **figure IV.8** représente l'évolution du temps d'exécution d'une tâche pour un vecteur point fixe. On remarque une diminution dans le temps de cycle entre la valeur initiale et la valeur finale avec une différence de **0.4sec**

Conclusion

On conclut que la variation de l'accélération et la vitesse maximale impacte le temps de cycle et l'énergie consommée pour cet ensemble de points.

IV.2.2.3 Ensemble de 20 points

Voici les résultats obtenues :

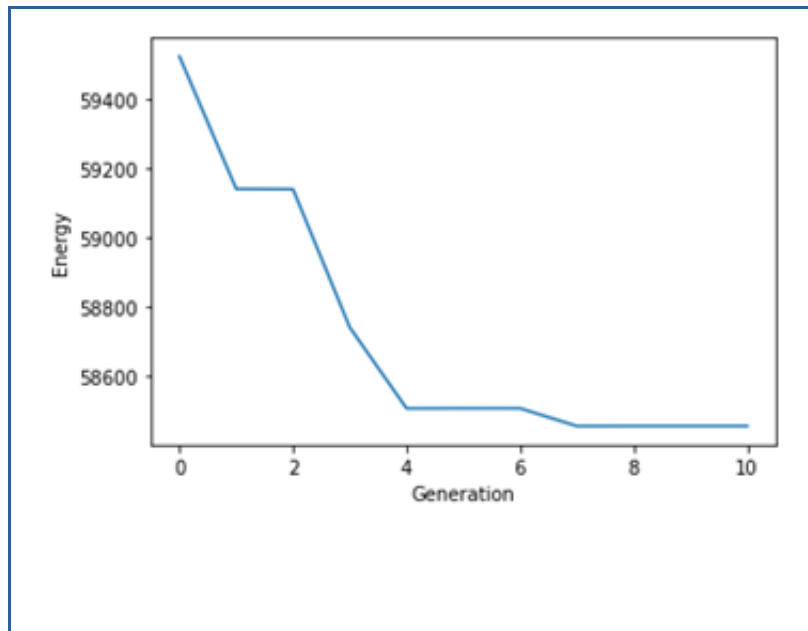


Figure IV. 9. Evolution de l'énergie avec vecteur point fixe pour un ensemble de 20 points

Interprétation

La **figure IV.9.** représente l'évolution de l'énergie dissipée par le robot durant un cycle pour une permutation fixe.

On remarque que la courbe est décroissante, ce qui implique la diminution de l'énergie à travers les générations. On remarque aussi une évolution de **1069.204 N.m** dans l'énergie consommée entre la première et la dernière valeur.

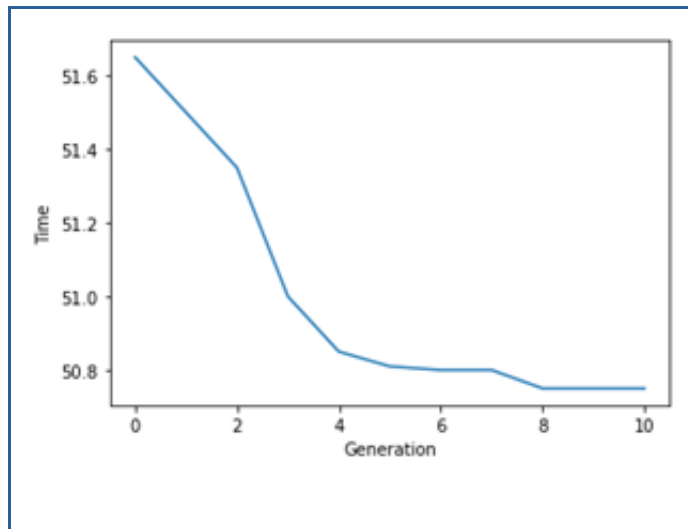


Figure IV. 10. Evolution du temps avec vecteur point fixe pour un ensemble de 20 points

Interprétation

La **figure IV.10** représente l'évolution du temps de cycle par génération.

On distingue sur cette figure une amélioration dans le temps d'exécution de la tâche, avec une diminution de **0.9 sec** dans la durée de cycle.

Conclusion

On conclut qu'en variant la vitesse et l'accélération, le temps de cycle ainsi que l'énergie dissipée ont diminué.

Conclusion général

D'après les trois expérimentations précédentes, et après les interprétations et nos conclusions tirées de chaque expérimentation, on peut avec certitude conclure que la variation de la vitesse et l'accélération impactent directement le bilan énergétique du robot ainsi que le temps de cycle d'une tâche ; et qu'on peut optimiser la fonction coût en variant ces deux paramètres.

IV.2.3. Impact des coefficient de pondérations

Dans ce titre, nous allons varier les coefficients de pondération de la fonction coût (Fitness), afin de :

- Déterminer si on peut favoriser l'optimisation de l'énergie ou du temps selon le choix.
- Voir la réaction du système d'optimisation de ces changements.

Nous allons effectuer nos expériences sur 4 ensembles de points différents (7, 10, 15 et 20 points).

Pour cette partie, nous allons présenter les résultats obtenus sur un tableau. Le tableau contiendra les résultats obtenus pour différents coefficients de pondération (0.5, 0.8, 0.2). Ensuite on fera une étude comparative afin d'évaluer le comportement du système cas le cas étudié.

À noter que dans cette expérience, le vecteur chemin entier est variable (vecteur point, vitesse et accélération).

Voici le tableau des résultats :

Fitness =		$0,5*T + 0,5*E$	$0,8*E + 0,2*T$	$0,8*T + 0,2*E$
Paramètre	Ensemble de points			
E	7	23165.548	25351.894	22582.31
	10	24777.47	24784.25	22967.98
	15	42417.36	41968.67	42285.80
	20	58511.52	61016.367	59174.875
T	7	21.200	23.550	20.95
	10	23.35	23.9	21.95
	15	37.25	35.85	37.25
	20	51.800	53.25	52,5

Tableau IV. 1. Impacte des facteurs de pondération

Interprétation

La comparaison passera par deux étapes, la première est la discussion sur le bilan énergétique et la deuxième sera la discussion sur le temps de cycle.

Pour le bilan énergétique :

1. On remarque pour les ensembles de 7 et 10 points que parmi les trois combinaisons, lorsqu'on met davantage l'optimisation du temps avec ($0,8*T$ et $0,2*E$) la consommation d'énergie est plus faible que lorsqu'on met les coefficients de pondérations égales ($0,5*T$, $0,5*E$), et ces deux dernières combinaisons donnent de meilleurs résultats que la combinaison qui met davantage l'énergie ($0,8*E$, $0,2*T$).
2. Pour la combinaison contenant 15 points, la combinaison qui donne l'avantage à l'énergie a donné le meilleur résultat en comparant avec les deux autres combinaisons.
3. Pour l'ensemble de 20 points, on remarque que le meilleur résultat a été obtenu de la combinaison qui met les coefficients égaux.

Pour le temps de cycle :

1. On remarque pour les ensembles de 7 et 10 points, que le meilleur temps d'exécution a été obtenu par la combinaison qui met davantage le temps ($0,8*T$, $0,2*E$). Puis le meilleur temps suivant été obtenu par la combinaison des coefficients égaux ; et le résultat le moins bon a été obtenu par la combinaison qui met davantage l'énergie et qui met en désavantage le temps.
2. On remarque pour la combinaison de 15 points, que la combinaison des coefficients égaux et la combinaison qui met davantage le temps ont donné le même résultats, un résultat moins bon que celui qui a été calculé par la combinaison qui met davantage l'énergie.
3. Pour l'ensemble de 20 points, on remarque que les coefficients de pondérations égales ont donné le meilleur résultat que les deux autres combinaisons.

Conclusion

D'après l'analyse et l'interprétation des résultats présentés sur le tableau, on a tiré la conclusion suivante :

La modification des coefficients de pondération agit sur le comportement du système et cause un changement sur les résultats obtenus par ce dernier. Le système devient imprédictible lorsqu'il s'agit d'une modification des coefficients mentionnés. L'imprédictibilité du système se résume dans le fait où on ne pas savoir quel paramètre va être avantageux dans l'optimisation.

IV.3. Conclusion

Dans ce chapitre, nous avons présenté les résultats obtenus par le modèle d'optimisation développé au cours de ce projet. Nous avons également fourni des interprétations des résultats suivis par des conclusions afin d'essayer de mieux expliquer et comprendre les résultats obtenus dans ce travail.

Un protocole d'expérimentation a été suivi afin d'effectuer les différents tests sur notre application développée. Nous avons effectué des tests de validation sur le modèle d'optimisation, afin d'assurer le bon fonctionnement de ce dernier. Ensuite nous avons montré l'impact de la vitesse et l'accélération sur les résultats du travail de l'optimisation. Pour finir, nous avons aussi montré l'impact de la modification des coefficients de pondérations sur le résultats obtenus.

Conclusion

général

Conclusion générale

La problématique qui a été traitée dans notre travail est portée sur l'implémentation d'un modèle d'optimisation. Ce modèle a comme critère, optimiser l'énergie et le temps d'un cobot lors de son parcours d'un ensemble de points introduit par un utilisateur. Notre modèle peut être adapté à n'importe quelle cobot, nous avons choisi le cobot « kuka lbr iiwa 7 ».

Pour notre projet, on a utilisé en premier lieu le travail [15] qui a été déjà fait et qui se base sur le développement d'un système de reconfiguration des tâches commandé par le robot KUKA LBR IIWA, dans le but d'une commande simple et conviviale, sans arrêt du robot ni utilisation de son interface habituelle de programmation. Grâce à ce travail, nous avons fait un guidage manuel du robot, c'est-à-dire nous guidons le robot vers le point souhaité de manière manuelle, ensuite on récupère les angles de chaque axe via une communication socket entre le robot et un ordinateur.

En second lieu, après avoir récupéré les différents points, nous implémentons notre modèle d'optimisation qui est basé sur les algorithmes métaheuristique, plus précisément l'algorithme génétique.

Le modèle d'optimisation commence par la génération d'un chemin qui contient les vecteurs points, vitesse et accélération. Ensuite il envoie ce chemin au simulateur « CoppeliaSim », afin de simuler ce dernier et de calculer l'énergie consommée et le temps d'exécution d'un parcours du robot. Par la suite, ce simulateur envoie l'énergie et le temps à l'algorithme génétique.

Après l'implantation de notre modèle nous avons obtenu des résultats satisfaisants, on a pu répondre à notre objectif en optimisant le temps et l'énergie pour différentes combinaisons de points.

Comme perspectives, on propose de :

- Développer un modèle de sécurité pour l'être humain et aussi pour le robot (par exemple, on peut implémenter un modèle d'évitement d'obstacle à l'aide d'une caméra).
- Faire une optimisation multi objectifs au lieu d'utiliser la somme pondérée de l'énergie et du temps
- Appliquer la cinématique inverse au lieu du guidage manuel du robot.
- Utiliser un modèle basé sur les réseaux de neurones ou le reinforcement learning au lieu d'un algorithme génétique.

Références bibliographiques

- [1] C. DINE, S. MECHOUAR. « Développement d'un système d'apprentissage/commande robotique », Mémoire de Master, Option Automatique et Système. USTHB. (2021)
- [2] F. GEANDARME, visiativ solution, « Industrie 4.0, définition et mise en œuvre vers l'usine de production connectée », [site web], consulté le (25.05.2022),
[visiativ-solutions.fr]
- [3] J. FRANKFIELD, investopedia, « Artificial Intelligence », [site web], consulté le (25.05.2022),
[investopedia.com]
- [4] I. BERCY, economie.gouv entreprises, « blockchain-définition-avantage-utilisation-application », [site web], consulté le (25.05.2022),
[economie.gouv.fr]
- [5] D. MCGINNIS, salesforce, blog, « what-is-the-fourth-industrial-revolution », [site web], consulté le (25.05.2022),
[https://www.salesforce.com/]
- [6] VEEPLY, « industrie-4.0-definition », [site web], consulté le (25.05.2022),
[yeePLY.com]
- [7] SAP, insights, « what-is-industry-4.0 », [site web], consulté le (25.05.2022),
[https://www.sap.com/insights/what-is-industry-4-0.html]
- [8] M. CASCELL, Iblog-qhse, « robotisation industriel vers l'infini et au delà », [site web], consulté le (27.05.2022),
[blog-qhse.com]
- [9] INTEL, « Type de robot et applications industrielles de la technologie », [site web], consulté le (27.05.2022),
[intel.fr]
- [10] P. LAURIN, mga-technologies, « Tout comprendre sur les cobots ou robots collaboratifs », [site web], consulté le (27.05.2022),
[https://expertise.boschrexroth.fr/tout-comprendre-sur-les-cobots-ou-robots-collaboratifs/]
- [11] MGA-TECHNOLOGIES, « La robotique collaborative, son objectif et ses avantages », [site web], consulté le (12.05.2022),
[mga-technologies.fr]
- [12] WEGFRANCE, « industrie 4.0 et les gains en efficacité énergétique pour les entreprises », [site web], consulté le (12.05.2022),
[wegfrance.news]
- [13] Adrian O, Serban O, Mihai CN, « Some Proper Methods for Optimization in Robotics ». International Journal of Modeling and Optimization, Vol. 7, No. 4, Aout (2017)

[14] Mohammed A, Schmidt B, Wang LH, Gao L. « Minimizing energy consumption for robot arm movement». Procedia CIRP 2014 ;25 :400–5. (2014)

[15]: KATLEEN BLANCHET, « Au cœur de l'interaction humain-robot collaboratif : Comment concevoir une assistance personnalisée au profil utilisateur ? ». Thèse de Doctorat, option informatique, robotique. (2021)

[16] W. Djekoune, CDTA, « recherche et technologie, division productique et robotique », [site web], consulté le (02.06.2022),

[www.cdta.dz/fr/recherche-et-technologie/divisions/division-productique-et-robotique/]

[17] : Bugmann, G., Siegel, M. Burcin, R., « A Role for Robotics in Sustainable Development», In: Proceedings of the IEEE Africon, (2011), p. 13-15.

[18] : Chemnitz, M., Schreck, G., Kruger, J., « Analyzing energy consumption of industrial robots», In: Proceedings of the IEEE ETFA, (2011)

[19] : Meike, D., Ribickis, L., « Energy Efficient Use of Robotics in the Automobile Industry», In: Proceedings of the 15th International Conference on Advanced Robotics, (2011), p.507-511.

[20] : Vergnano, A., Thorstensson, C., Lennartson, B., Falkman, P., Pellicciari, M., Leali, F., Biller, S., « Modeling and Optimization of Energy Consumption in Cooperative Multi-Robot Systems», IEEE Transactions on Automation Science and Engineering, (2012).

[21] : Meike, D., Ribickis, L., « Recuperated Energy Savings Potential and Approaches in Industrial Robotics», In: Proceedings of IEEE International Conference on Automation Science and Engineering, (2011), p. 299-303.

[22] ACADEMIA, « Meta-heuristics», [site web], consulté le (05.04.2022),

[<https://www.academia.edu/download/59334769/Meta-heuristics20190520-36938-anxarc.pdf>]

[23] WIKIPEDIA sous licence CC-BY-SA 3.0, techno-science, « Métaheuristique – définition et explication », [site web], (05.04.2022)

[techno-science.net]

[24] I. BOUSSAID, « PERFECTIONNEMENT DE MÉTAHEURISTIQUES POUR L'OPTIMISATION CONTINUE ». Thèse de Doctorat, option informatique. (29-06-2013), disponible sur le site web

[[Perfectionnement de métaheuristiques pour l'optimisation continue \(archives-ouvertes.fr\)](http://Perfectionnement%20de%20métaheuristiques%20pour%20l'optimisation%20continue%20(archives-ouvertes.fr))]

[25] KHAYYAM, developpez, « Algorithme génétiques », [site web], consulté le (05.04.2022), [developpez.com]

[26] J. HAO, C. SOLNON, « cours Algorithme génétique », disponible sur le site web

[[P:/M_helios/Hao/PUBLIS/HAO/Chapitre IA/v1Bis.dvi](http://P:/M_helios/Hao/PUBLIS/HAO/Chapitre%20IA/v1Bis.dvi) (univ-angers.fr)]

[27] V. MAGNIN, igm.univ-mlv, « Algorithme génétique », [site web], consulté le (05.04.2022),

[\[Algorithmes génétiques \(univ-mlv.fr\)\]](#)

[28] SPYDER, [site web], consulté le (06.06.2022),

[\[https://www.spyder-ide.org/\]](https://www.spyder-ide.org/)

[29] MULESOFT, ressours, api, « what is an api », [site web], consulté le (06.06.2022),

[\[https://www.mulesoft.com/fr/resources/api/what-is-an-api#:~:text=API%20est%20l'acronyme%20d,applications%20de%20communiquer%20entre%20elles\]](https://www.mulesoft.com/fr/resources/api/what-is-an-api#:~:text=API%20est%20l'acronyme%20d,applications%20de%20communiquer%20entre%20elles)

[30] DOCS.PYTHON, « les classes », [site web], consulté le (06.06.2022),

[\[https://docs.python.org/fr/3/tutorial/classes.html\]](https://docs.python.org/fr/3/tutorial/classes.html)

[31] DATABIRD, python, « bibliothèque python », [site web], consulté le (06.06.2022),

[\[https://www.data-bird.co/python/bibliotheque-python#:~:text=En%20langage%20Python%2C%20une%20librairie,selon%20le%20classement%20Tiobe%202021\]](https://www.data-bird.co/python/bibliotheque-python#:~:text=En%20langage%20Python%2C%20une%20librairie,selon%20le%20classement%20Tiobe%202021)

[32] P. LAURA, datascientest, « Numpy », [site web], consulté le (06.06.2022),

[\[https://datascientest.com/numpy\]](https://datascientest.com/numpy)

[33] A. RONQUILLO, realpython, « python time module », [site web], consulté le (06.06.2022),

[\[https://realpython.com/python-time-module/\]](https://realpython.com/python-time-module/)

[34] DOCS.PYTHON, library, « random », [site web], consulté le (06.06.2022),

[\[https://docs.python.org/3/library/random.html\]](https://docs.python.org/3/library/random.html)

[35] COPPELIAROBOTICS, helpFiles, « remoteApiOverview », [site web], consulté le (21.03.2022),

[\[https://www.coppeliarobotics.com/helpFiles/en/remoteApiOverview.htm\]](https://www.coppeliarobotics.com/helpFiles/en/remoteApiOverview.htm)

[36] J. HUNTER, D. DALE, E. FIRING, M. DROETTBOOM, « matplotlib », [site web], consulté le (06.06.2022),

[\[https://matplotlib.org/3.5.0/tutorials/introductory/pyplot.html\]](https://matplotlib.org/3.5.0/tutorials/introductory/pyplot.html)

[37] E. STOLTZ, « towardsdatascience, Evolution of a salesman a complete genetic algorithm tutorial for python », [site web], consulté le (19.03.2022),

[\[https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35\]](https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35)

[38] COPPELIAROBOTICS, « CoppeliaSim user manual », [site web], consulté le (21.03.2022),

[\[coppeliarobotics.com\]](https://coppeliarobotics.com)

[39] : M. Safeea, P. Neto, (2020), « KUKA Sunrise Toolbox: Interfacing Collaborative Robots with MATLAB», [fichier pdf], pp,5 disponible sur: [https://arxiv.org/ftp/arxiv/papers/1709/1709.01438.pdf?fbclid=IwAR3wRmzwTiBBwpqVa3eP5PVVOuw1CrZce0I9WsBFyJCQKff2nj-Gi-Esws]

[40] : M. SAFEEA, P. NETO, R. BEAREE, (octobre 2019), « Precise hand-guiding of redundant manipulators with null space control for in-contact obstacle navigation», [fichier pdf], disponible sur [http://dx.doi.org/10.1109/IECON.2019.8927766]

Annexe

Annexe 1 : Caractéristique du robot

Le tableau suivants contient les différents caractéristiques principaux des différents axes du robot.

« LBR IIWA 7 R800 »

Axes	Plage de mouvement angulaire pour chaque robot (en degré)	Couples généré par chaque axe trame par trame [N.m]
1	± 170	± 176
2	± 120	± 176
3	± 170	± 110
4	± 120	± 110
5	± 170	± 110
6	± 120	± 40
7	± 175	± 40

Guidage manuel (Hand-guiding)

Le guidage manuel est une fonctionnalité représentative des robots collaboratifs. Il permet aux utilisateurs non qualifiés d'interagir et de programmer un robot rapidement de manière intuitive[39]. Pour le hand-guiding le robot se déplace conformément aux instructions de l'utilisateur en appliquant une force (couple) externe sur la structure du robot (comme montré sur la figure), plus exactement sur l'effecteur [40]



Figure 1.1. Enseignement par guidage manuel de cinq points cibles.

Annexe 2 : Code Python

voici le script de l'algorithme génétique :

```
# Final GA code :
import numpy as np, random, operator, pandas as pd, matplotlib.pyplot as plt
from typing import Iterable
import Final_KUKA
import Transformation
import time
start_time = time.time()
"Initialisation : "
T = []
E = []
Velocity = []
Acceleration = []
DATA = []
cityList = []
class Fitness1:
    def __init__(self, route):
        self.route = route
        self.fit = 0.0
        self.fitness = 0.0

    def CalculFit(self):
        Fit = 0.0
        P = self.route
        v = P
        #print("L'algorithme a donné le résultat suivant: ", v) # Si vous voulez voir l'individu généré a chaque fois

        Fit = Final_KUKA.main(v)
        print("Fitness = ", Fit)

        Fitt = 0.8 * Transformation.TimeTransfo(Fit[0]) + 0.2 * Transformation.EnergyTransfo(Fit[1])
        Fitt = 1/Fitt
        Fitt = 1/Fitt
        k = [Fitt, Fit]
        DATA.append(k)
        # print("Data = ", DATA)
        self.fit = Fitt
        return self.fit

    def routeFitness(self):
        if self.fitness == 0:
            self.fitness = 1 / float(self.CalculFit())
        return self.fitness

def flatten(items):
    for x in items:
        if isinstance(x, Iterable) and not isinstance(x, (str, bytes)):
            for sub_x in flatten(x):
                yield sub_x
        else:
            yield x
```



```
"Choisir l'ensemble de points "
```

```
""""
```

```
"Test pour 20 Points aléatoire"
```

```
p1=[-59.58761069, 14.32394488, -25.78310078, -84.79775368, 6.30253575, 82.5059225, -74.48451337]
p2=[-114.01860123, 54.43099054, -5.15662016, -52.71211715, 5.15662016, 73.91155557, -108.28902328]
p3=[-95.11099399, 20.05352283, -29.79380535, -65.89014644, 9.74028252, 96.82986738, -103.70536092]
p4=[-105.4242343, 52.71211715, -5.72957795, -41.25296125, 5.15662016, 86.51662706, -91.10028943]
p5=[-102.55944533, 43.54479243, -11.4591559, -83.65183809, 10.31324031, 55.00394833, -89.95437384]
p6=[-61.30648408, 10.88619811, -33.23155212, -79.64113352, 6.30253575, 91.10028943, -69.90085101]
p7=[ 30.93972094, 36.66929889, 11.4591559, -71.61972439, -7.44845134, 72.19268219, 9.74028252]
p8=[ 52.13915936, 62.45239967, 0.5729578, -61.87944187, -6.30253575, 56.72282172, 32.08563653]
p9=[ 0. , 41.25296125, 0. , -74.48451337, -0. , 64.74423085, 0.5729578 ]
p10=[ 0.34377468, 60.16056849, 1.14591559, -56.72282172, -0.5729578, 63.59831526, 0.5729578 ]
p11=[-29.79380535, 56.14986392, -0.5729578, -56.72282172, 0.5729578, 68.18197762, -26.92901637]
p12=[-13.17802929, 20.62648062, -1.71887339, -77.34930234, 0.5729578, 81.9329647, -26.35605858]
p13=[-71.61972439, 22.91831181, -20.62648062, -95.68395179, 9.74028252, 63.59831526, -81.9329647]
p14=[-119.74817918, 57.86873731, -8.02140913, -72.19268219, 9.74028252, 52.71211715, -92.81916281]
p15=[-85.37071147, 22.34535401, -18.33464944, -66.46310424, 7.44845134, 92.24620502, -115.73747462]
p16=[-36.09634109, 17.76169165, -9.16732472, -67.60901983, 2.86478898, 95.11099399, -68.75493542]
p17=[ 52.13915936, 29.79380535, 16.61577606, -79.06817573, -8.59436693, 72.19268219, 45.26366582]
p18=[ 29.79380535, 57.29577951, 0.5729578, -70.4738088, -1.14591559, 52.71211715, 27.50197417]
p19=[ 15.46986047, 21.19943842, 2.86478898, -104.27831871, -1.71887339, 55.00394833, 27.50197417]
p20=[ -6.30253575, 60.73352628, -0.5729578, -64.74423085, 0.5729578, 55.00394833, -1.71887339]
p21=[ 26.92901637, 21.77239621, 9.16732472, -52.71211715, -3.43774677, 105.9971921, 18.33464944]
p22=[ 48.70141259, 50.42028597, 8.02140913, -91.10028943, -9.74028252, 40.10704566, 24.0642274 ]
```

```
cityList = [p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15,p16,p17,p18,p19,p20]
```

```
""""
```

```
" Initialisation Liste des vecteurs"
```

```
""""
```

```
"Test pour 7 Points aléatoire"
```

```
p1=[-103.88,50 ,1.26 ,-92.32 ,-1.45 ,37.67 ,4.4 ]
p2=[-7.27 ,36.23 ,1.26 ,-55.72,-0.66 ,88.05 ,-4.55 ]
p3=[116.22 ,34.79 ,1.26 ,-87.50,-0.78 ,57.66 ,50.49 ]
p4=[-40.10 ,53.34 ,1.26 ,-66.46,-1.09 ,60.27 ,37.13 ]
p5=[-56.97 ,28.88 ,1.26 ,-40.06,-0.6 ,111.0 ,50.24 ]
p6=[-144.61 ,42.07 ,1.26 ,-48.58,-0.78 ,89.42 ,-37.34]
p7=[-21.31 ,34.79 ,1.26 ,-87.51,-0.79 ,57.66 ,50.49 ]
```

```
cityList = [p1,p2,p3,p4,p5,p6,p7]
```

```
""""
```

```
# Velocity vector
```

```
cityList2 = [0.10,0.20,0.30,0.40,0.50,0.60,0.70,0.80,0.90,1.00]
```

```
# Acceleration vector
```

```
cityList3 = [0.50,0.70,0.90,1.0,1.10,1.20,1.30,1.40,1.50,1.60,1.70,1.80,1.90,2.0]
```

```
"""" Création de population """"
```

```
def createRoute(cityList):
```

```
    route = random.sample(cityList, len(cityList)) # Vecteur des points
```

```
    Route = random.choices(cityList2,k = len(route)) # Vecteur vitesse
```

```
    RRoute = random.choices(cityList3,k = len(route)) # Vecteur acceleration
```

```
    route.append(Route)
```

```
route.append(RRoute)
global lengthRoute
lengthRoute= len(Route)
```

```
# Envoyer le chemin à vrep pour le simuler
return route
```

""" La fonction Create Route ne produit qu'un seul individu, nous avons donc créé la fonction InitialPopulation pour avoir autant de routes que nous voulons pour notre population """

```
def initialPopulation(popSize, cityList):
    population = []

    for i in range(0, popSize):
        population.append(createRoute(cityList))
    return population
```

""" Fitness fonction """

```
def rankRoutes(population):
    fitnessResults = {}
    for i in range(0, len(population)):
        fitnessResults[i] = Fitness1(population[i]).routeFitness()
    print("Classement des individus = ", sorted(fitnessResults.items(), key = operator.itemgetter(1), reverse = True))
    return sorted(fitnessResults.items(), key = operator.itemgetter(1), reverse = True)
```

""" Selection the mating pool """

```
def selection(popRanked, eliteSize):
    selectionResults = []
    df = pd.DataFrame(np.array(popRanked), columns=["Index", "Fitness"])
    df['cum_sum'] = df.Fitness.cumsum()
    df['cum_perc'] = 100*df.cum_sum/df.Fitness.sum()
    print("Selection is here : ")
    print(df)
    for i in range(0, eliteSize):
        selectionResults.append(popRanked[i][0])
    for i in range(0, len(popRanked) - eliteSize):
        pick = 100*random.random()
        for i in range(0, len(popRanked)):
            if pick <= df.iat[i,3]:
                selectionResults.append(popRanked[i][0])
                break
    return selectionResults
```

```
def matingPool(population, selectionResults):
    matingpool = []
    for i in range(0, len(selectionResults)):
        index = selectionResults[i]
        matingpool.append(population[index])
    return matingpool
```

""" Bredding function aka children creation """

```
def PREBREED_Vitesse(parent1, parent2):
    B = parent1[len(parent1)-2 :len(parent1)-1 ]
```

```

B = list(flatten(B))

A = parent2[len(parent2)-2 : len(parent2)-1]
A = list(flatten(A))

KID = []

index = int(len(parent1)/2)

for i in range(index):
    KID.append(B[i])

for i in range(index, len(A)):
    KID.append(A[i])
    if len(KID) > len(A):
        break

return KID

def PREBREED_Accel(parent1, parent2):
    BB = parent1[len(parent1)-1 : len(parent1)]
    BB = list(flatten(BB))

    AA = parent2[len(parent2)-1 : len(parent2)]
    AA = list(flatten(AA))

    kid = []

    index = int(len(BB)/2)

    for i in range(index):
        kid.append(BB[i])

    for i in range(index, len(AA)):
        kid.append(AA[i])
        if len(kid) > len(AA):
            break

    return kid

def breed(parent1, parent2):

    child = []
    childP1 = []
    childP2 = []
    P1 = parent1
    P2 = parent2

    parent1 = parent1[:len(parent1)-2]
    parent2 = parent2[:len(parent2)-2]

    geneA = int(random.random() * len(parent1))
    geneB = int(random.random() * len(parent1))

    startGene = min(geneA, geneB)

```

```

endGene = max(geneA, geneB)

for i in range(startGene, endGene):
    childP1.append(parent1[i])

childP2 = [item for item in parent2 if item not in childP1]

child = childP1 + childP2      # Permutation des points

x = PREBREED_Vitesse(P1,P2)    # Permutation Vitesse
y = PREBREED_Accel(P1,P2)     # Permutation Acceleration
child.append(x)
child.append(y)
return child

def breedPopulation(matingpool, eliteSize):
    children = []
    length = len(matingpool) - eliteSize
    pool = random.sample(matingpool, len(matingpool))

    for i in range(0, eliteSize):
        children.append(matingpool[i])

    for i in range(0, length):
        child = breed(pool[i], pool[len(matingpool)-i-1])
        children.append(child)
    return children

""" Mutation """
def Amutate(vector, mutationRate):
    vector = list(flatten(vector))
    for i in range (lengthRoute):

        if random.random() < mutationRate:
            if random.random() < 0.5:
                vector[i] += 0.1
            else:
                vector[i] -= 0.1
        while True:
            if vector[i] > 2:

                vector[i] = 2
                break
            elif vector[i] < 0.5:
                vector [i] = 0.5
                break
            else:
                break

    return vector

def Vmutate(V, mutationRate):
    V = list(flatten(V))
    for i in range (lengthRoute):

        if random.random() < mutationRate:
            if random.random() < 0.5:

```

```

        V[i] += 0.10
    else:
        V[i] -= 0.10
    while True:
        if V[i] > 1:

            V[i] = 1
            break
        elif V[i] <= 0.10:
            V[i] = 0.10
            break
        else:
            break

    return V

```

```
def mutate(individual, mutationRate):
```

```

    Avector = individual[len(individual)-1:len(individual)] # Acceleration
    VVector = individual[len(individual)-2:len(individual)-1] # Velocity

```

```
    individual = individual[:len(individual)-2]
```

```

    for swapped in range(len(individual)):
        if(random.random() < mutationRate):
            swapWith = int(random.random() * len(individual))

```

```

        city1 = individual[swapped]
        city2 = individual[swapWith]

```

```

        individual[swapped] = city2
        individual[swapWith] = city1

```

```

    x = list(flatten(Vmutate(VVector,mutationRate))) # Vecteur vutesse après mutation
    y = list(flatten(Amutate(Avector,mutationRate))) # Vecteur accélération après mutation
    individual.append(x)
    individual.append(y)

```

```
    return individual # Vecteur chemin après mutation
```

```
def mutatePopulation(population, mutationRate, eliteSize):
```

```
    mutatedPop = []
```

```

    for i in range(0,eliteSize):
        mutatedPop.append(population[i])
    for ind in range(0, len(population)):
        mutatedInd = mutate(population[ind], mutationRate)
        mutatedPop.append(mutatedInd)
    return mutatedPop

```

```
""" Repeating the Process """
```

```
def nextGeneration(currentGen, eliteSize, mutationRate):
```

```

popRanked = rankRoutes(currentGen)
selectionResults = selection(popRanked, eliteSize)
matingpool = matingPool(currentGen, selectionResults)
children = breedPopulation(matingpool, eliteSize)
nextGeneration = mutatePopulation(children, mutationRate, eliteSize)
return nextGeneration

""" Genetic Algorithm Function """
def geneticAlgorithm(population, popSize, eliteSize, mutationRate, generations):
    pop = initialPopulation(popSize, population)
    print("Initial distance: " + str(1 / rankRoutes(pop)[0][1])) # distance is the inverse of the fitness

    for i in range(0, generations):
        pop = nextGeneration(pop, eliteSize, mutationRate)

    print("Final distance: " + str(1 / rankRoutes(pop)[0][1]))
    bestRouteIndex = rankRoutes(pop)[0][0]
    bestRoute = pop[bestRouteIndex] # Best route using indexes
    print("best route is ", bestRoute)
    return bestRoute

""" Plotting The Results """
def TE(A,P):
    Time = []
    Energy = []
    S=[]
    for i in range(len(P)):

        for j in range (len(A)):

            if A[j][0] == P[i]:
                S = A[j][1]
                # print("S = ",S)

                Time.append(S[0])
                Energy.append(S[1])
                break
    plt.plot(P)
    plt.ylabel('Fitness')
    plt.xlabel('Generation')
    plt.show()

    plt.plot(Time)
    plt.ylabel('Time')
    plt.xlabel('Generation')
    plt.show()
    print('Time = ',Time)
    plt.plot(Energy)
    plt.ylabel('Energy')
    plt.xlabel('Generation')
    plt.show()
    print('Energy = ',Energy)
    return

def geneticAlgorithmPlot(population, popSize, eliteSize, mutationRate, generations):
    pop = initialPopulation(popSize, population)
    print("Fitnessse initiale : " + str(1 / rankRoutes(pop)[0][1]))

```

```

progress = []
progress.append(1 / rankRoutes(pop)[0][1])

for i in range(0, generations):
    print("progress [" ,i+1, "] = ", progress)
    pop = nextGeneration(pop, eliteSize, mutationRate)
    progress.append(1 / rankRoutes(pop)[0][1])

print("Data = ", DATA)
print("Progresse finale=", progress)
TE(DATA,progress)

print("Fitness finale:" + str(1 / rankRoutes(pop)[0][1]))
bestRouteIndex = rankRoutes(pop)[0][0]
bestRoute = pop[bestRouteIndex]          # best route to take
print("Meilleure résultat est : ",bestRoute)
print("Simulation time = ", time.time()-start_time)

#####

""" Launching the Algorithm """

n = input('Do you want to plot the results ? 1 for YES Or 0 for NO : ')
g = input('Number Of Generations Desired : ')
if (int(n) == 1):
    geneticAlgorithmPlot(population=cityList, popSize=15, eliteSize=2, mutationRate=0.02, generations=int(g))
else:
    geneticAlgorithm(population=cityList, popSize=15, eliteSize=2, mutationRate=0.02, generations=int(g))

```

Voici le script du simulateur « CoppeliaSim »

```

SS = false
fin = {}
Energy =0
Time = 0
E = 0
sim.setBoolParameter(sim.boolparam_display_enabled,false)
function sysCall_init()
    corout=coroutine.create(coroutineMain)
end

function sysCall_actuation()
    if coroutine.status(corout)~='dead' then
        local ok,errorMsg=coroutine.resume(corout)
        if errorMsg then
            error(debug.traceback(corout,errorMsg),2)
        end
    end
end

```

```

    end
end

function movCallback(config,vel,accel,handles)
    for i=1,#handles,1 do
        if          sim.getJointMode(handles[i])==sim.jointmode_force          and
sim.isDynamicallyEnabled(handles[i]) then
            sim.setJointTargetPosition(handles[i],config[i])
            -- Calcul de Couple généré frame by frame
            a = math.abs(sim.getJointForce(handles[i]))
            E = E + a
        else
            sim.setJointPosition(handles[i],config[i])
        end
    end
end

function myFunction (inInts,inFloats,inStrings,inBuffer)
    print(inInts)
    indexVell=inInts[1]-1
    indexAccel=inInts[1]
    len = (inInts[1] - 2 )*7
    lenpoint=inInts[1]-2
    P=inFloats
    print("P = ",#P)
    inInts = 0
    D = {}
    M = {}
    X = {}
    q = 0
    b = 1
    Vell = {}
    Accel = {}
    for i=1,len,1 do
        q=q+1
    end
end

```



```

    D[q]=P[i]
    if #D == 7 then
        X = D
        M[b] = X

        b = b + 1
        q = 0
        D = {}
    end
end
for i=len+1,#P,1 do
    q=q+1
    D[q]=P[i]
    if #D == lenpoint then
        X = D
        M[b] = X
        b = b + 1
        q = 0
        D = {}
    end
    Vell=M[indexVell]
    Accel=M[indexAccel]
end
print("length M = "..#M)
print("Vell = ",Vell)
SS = true
return inInts,inFloats,inStrings,inBuffer
end

```

```

function moveToConfig(handles,maxVel,maxAccel,maxJerk,targetConf)
    local currentConf={}

```

```

    for i=1,#handles,1 do
        currentConf[i]=sim.getJointPosition(handles[i])
    end
    sim.moveToConfig(-
1,currentConf,nil,nil,maxVel,maxAccel,maxJerk,targetConf,nil,movCallback,handles)
end

T = {}
function coroutineMain()
    z = 0
    local jointHandles={}
    for i=1,7,1 do
        jointHandles[i]=sim.getObject('/joint',{index=i-1})
    end
    local vel=110
    local accel=80
    local jerk=80

    local
maxVel={vel*math.pi/180,vel*math.pi/180,vel*math.pi/180,vel*math.pi/180,vel*math.pi/180,vel
*math.pi/180,vel*math.pi/180}
    local
maxAccel={accel*math.pi/180,accel*math.pi/180,accel*math.pi/180,accel*math.pi/180,accel*ma
th.pi/180,accel*math.pi/180,accel*math.pi/180}
    local
maxJerk={jerk*math.pi/180,jerk*math.pi/180,jerk*math.pi/180,jerk*math.pi/180,jerk*math.pi/18
0,jerk*math.pi/180,jerk*math.pi/180}

    local          p1={-7.27*math.pi/180,36.23*math.pi/180,100.26*math.pi/180          ,-
55.72*math.pi/180,120.66*math.pi/180,88.05*math.pi/180,-4.55*math.pi/180  }
    local
p2={0*math.pi/180,0*math.pi/180,0*math.pi/180,0*math.pi/180,0*math.pi/180,0*math.pi/180,0*
math.pi/180}

```

```

local                                     p3={90*math.pi/180,90*math.pi/180,170*math.pi/180,-
90*math.pi/180,90*math.pi/180,90*math.pi/180,0}

p={p1,p2,p3}
e = 0
while true do
    if SS==true then
        e1 = E
        for j = 1,2,1 do
            moveToConfig(jointHandles,maxVel,maxAccel,maxJerk,p[j])
            T[j]= E
            if j== 2 then
                V = E
            end
        end
        print("maxvel avant :",maxVel[1])
        print("maxAccel avant :",maxAccel[1])
        L = E-e1
        V = E

        Tstart = sim.getSimulationTime()
        for j = 1,lenpoint,1 do
            vel=110* Vell[j]
            if vel <= 0.10 then
                vel = 0.1918621
            end
            maxVel={vel,vel,vel,vel,vel,vel,vel}
            accel=80* Accel[j]
            maxAccel={accel,accel,accel,accel,accel,accel,accel}

            print('VELOCITY ['.j..']='..vel)
            print('Acceleration ['.j..']='..accel)

            moveToConfig(jointHandles,maxVel,maxAccel,maxJerk,M[j])

```

```

        print("M[..j..]"=,M[j])
    end

    Tfinish=sim.getSimulationTime()
    Time = Tfinish - Tstart
    Energy = E - V
    print(" Simulation time = " , Time)
    print(' Energy consumed=' , Energy)
    SS=false
    TT = Time
    FF = Energy
    print("Energy = "..FF)
    print("Time = "..TT)
    z = 1
end
end
end

function VREP_PYTHON(inInts1,inFloats1,inStrings1,inBuffer1)
    if z == 1 then
        fin = {TT,FF}
        inFloats1 = fin
        inInts1 = {1}
    end
    return inInts1,inFloats1,inStrings1,inBuffer1
end

```