UNIVERSITÉ DE MONTRÉAL

ANALYSE DE SCÈNE RAPIDE UTILISANT LA VISION ET L'INTELLIGENCE ARTIFICIELLE POUR LA PRÉHENSION D'OBJETS PAR UN ROBOT D'ASSISTANCE

CHRISTOPHER BOUSQUET-JETTÉ

DÉPARTEMENT DE GÉNIE MÉCANIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE MÉCANIQUE)

AOÛT 2016

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

ANALYSE DE SCÈNE RAPIDE UTILISANT LA VISION ET L'INTELLIGENCE ARTIFICIELLE POUR LA PRÉHENSION D'OBJETS PAR UN ROBOT D'ASSISTANCE

présenté par : BOUSQUET-JETTÉ Christopher

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. SAUSSIÉ David, Ph. D., président

M. ACHICHE Sofiane, Ph. D., membre et directeur de recherche

M. RAISON Maxime, Doctorat, membre et codirecteur de recherche

Mme CHERIET Farida, Ph. D., membre

DÉDICACE

À mes parents, pour leur soutien.

À ma copine, pour son écoute et son encouragement.

REMERCIEMENTS

Je remercie énormément mon directeur Sofiane Achiche et mon codirecteur Maxime Raison de m’avoir fait confiance, d’avoir pris de leur temps pour m’épauler et de m’avoir montré comment acquérir une démarche scientifique. Sans eux rien n’aurait été possible. Je remercie tous mes collèges de maîtrise pour leur soutien. Je remercie les professeurs David Saussié et Farida Cheriet d’avoir accepté de participer à mon jury de maîtrise recherche.

RÉSUMÉ

L'assistance robotisée à l'aide de la vision est en pleine effervescence, notamment pour les personnes âgées en perte de mobilité et les personnes atteintes de troubles musculo-squelettiques. Ce mémoire met en lumière les solutions développées dans le cadre d'une maîtrise recherche du département de génie mécanique de l'École Polytechnique de Montréal.

Dans ce contexte, la Kinect V2 a permis l'acquisition surfacique de scènes amenant alors le projet à la détection d'objets. Les méthodes de détection actuelles les plus robustes prennent encore beaucoup de temps de calcul, empêchant l'automatisation de la préhension d'objets par des robots dans un temps acceptable pour l'assistance des utilisateurs au quotidien.

L'objectif est alors de développer un système d'analyse de scène rapide utilisant la vision et l'intelligence artificielle pour la préhension d'objets par un robot d'assistance. Ce système doit permettre de répondre à l'ensemble des questions suivantes plus rapidement que les méthodes existantes : 1. Combien y a-t-il d'objets et où se situent-ils? 2. Comment les saisir, c.-à-d. quels sont les endroits de préhension efficaces et quelle est la phase d'approche à donner au robot? 3. Quels sont ces objets de la scène, reconnus à partir d'un apprentissage neuronal sur un ensemble de données acquis avec une caméra active?

Avec l'acquisition d'un ensemble de données de 180 scènes comprenant un objet chacun, la solution a été développée en 3 étapes : 1. La détection d'objets comprenant la transformation des scènes brutes acquises en données matricielles et la segmentation 3D des scènes pour trouver les objets à l'aide d'un algorithme innovant de « palpage par le haut » suivi de l'élimination des points indésirables par calcul de leur gradient. 2. Apprentissage supervisé de l'ensemble des données suite aux algorithmes de détection d'objets des scènes. 3. Analyse de scène des objets comprenant l'identification des endroits de préhension des objets et la phase d'approche du bras robotique à l'aide d'un arbre de décision simple, puis l'utilisation d'un réseau neuronal combinant deux caractéristiques dont la surface et la couleur RGB nous permettant d'obtenir 83 % de performance dans un espace connu pour la reconnaissance d'objets.

Cette étude démontre que l'analyse de scène rapide utilisant la vision et l'intelligence artificielle pour la préhension d'objets par un robot d'assistance en coopération avec un utilisateur peut être réalisée en un temps efficace. En effet, le système prend en moyenne 0,6 seconde pour l'analyse d'un objet dans une scène. Avec le bras robotique d'assistance Jaco, le système peut prendre un

objet désiré en 15 secondes en se déplaçant à 50 mm/s et le temps peut être grandement diminué en augmentant la vitesse. Ce système combiné à un robot d’assistance a un grand potentiel afin de permettre aux utilisateurs d'être plus autonomes, indépendants et de mieux adhérer à l’utilisation de leur aide technique.

ABSTRACT

Vision-assisted robotic aid is a rapidly expanding field, particularly solutions developed for people affected by age-related loss of mobility and for people subject to musculoskeletal disorders. This thesis presents the series of the solutions developed in the context of a research master at the Mechanical Engineering Department of École Polytechnique de Montréal.

In this context, the Kinect V2 allows for rapid surface acquisition of scenes bringing the project to focus on objects detect. The current detection methods available need a lot of computing time, preventing the full automation of prehending objects by robots, in an acceptable time, for the assistance of target users in their everyday activities.

The objective of this study is therefore to develop algorithm for fast automated scene analysis and object prehension. The developed algorithm must provide answers to all the following questions faster than existing methods do: 1. How many of the objects are there, and where are they located? 2. Which coordinates on the objects are effective prehension targets and what is the favored path of approach for the robot? 3. What are the objects in the scene, as identified by a neural network on data from an active camera?

With the acquisition of a dataset composed of 180 scenes with an object in each scene, the solution was developed following three stages: 1. Object detection involving transformation of raw scenes into data matrices and 3D scene segmentation to find the objects, by means of a novel algorithm for “top-down probing”. This is followed by elimination of undesirable points based on their gradients. 2. After object detection, supervised learning is performed on the objects in the dataset. 3. Scenes containing the objects are analyzed, which includes identification of grasping targets on the objects using a simple decision tree, and selection of the approach path of the robotic arm for full prehension.

Subsequently, a neural network performs object recognition utilizing surface geometry and RGB color, yielding 83% performance in a controlled environment.

This study has shown that fast scene analysis for robotic prehension of objects in cooperation with a user can be performed with effective promptness. Indeed, the system requires on average 0.6 seconds to analyze an object in a scene. With the JACO robotic assistance arm, the system can pick up a requested object in 15 seconds while moving at 50 mm/s, which may be greatly improved

upon using faster robots. This system, in combination with an assistive robot, has great potential for providing potential users with improved autonomy and independence, and for encouraging sustained usage of technical aids.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

3D	Trois dimensions dans l'espace
OBJ	Format d'enregistrement d'un objet 3D
PLY	Format d'enregistrement *Polygon File Format*
RGB	Couleur R (rouge), G (vert) et B (bleu)
RGB-D	RGB avec la profondeur D
SDK	Software Development Kit
STL	Format d'enregistrement *Standard Triangle Language*
SVM	*Support Vector Machine*
XYZ	Coordonnées 3D dans l'espace cartésien X, Y et Z

LISTE DES ANNEXES

CHAPITRE 1 INTRODUCTION

L’assistance robotisée à l’aide de la vision est en pleine effervescence, notamment pour les personnes âgées en perte de mobilité et les personnes atteintes de troubles musculo-squelettiques. Ces robots sont souvent dotés de préhenseurs pour augmenter l'autonomie et l'habileté des utilisateurs ayant des limitations motrices aux membres supérieurs dans leurs tâches quotidiennes telles que saisir des objets, ouvrir une porte ou contrôler un interrupteur.

Dans un contexte d’assistance technique pour des personnes atteintes de dystrophie musculaire ou de paralysie cérébrale, Kinova robotics inc (2016) [1], une jeune entreprise œuvrant dans le domaine de la robotique, a développé un robot possédant six degrés de liberté servant à prendre des objets du quotidien. Le bras robotique est monté sur une chaise adaptée pour des utilisateurs à mobilité réduite comme illustré sur la Figure 1.1. Ce robot est doté d'un préhenseur à trois doigts fixés à son extrémité pour augmenter l'autonomie et l'habileté quotidienne des utilisateurs ayant des déficiences motrices aux membres supérieurs.

Figure 1.1 : Chaise adaptée pour un utilisateur avec le bras Jaco de Kinova (2016) [2].

La commande des robots peut se faire, entre autres, à l'aide d’une manette fixée sur l'appui-bras des fauteuils motorisés. L’efficacité d’utilisation d’un robot dans ce genre de contexte est évaluée dans le travail présenté dans (Maheu et al. (2011)) [3] et la conclusion est donc que son utilisation

reste très bénéfique pour les utilisateurs. Par contre, elle requiert un bon niveau de dextérité manuelle et peut ainsi demander beaucoup trop d'énergie pour certains. Cela peut rendre le temps d'atteinte des objets trop long et les manœuvres trop complexes engendrant ainsi fatigue et frustrations. Or, la facilité d'utilisation de ce type d'assistance robotique et la capacité à atteindre les objets dans un temps raisonnable sont des critères primordiaux afin d'assurer l'adhérence des usagers à leur aide technique.

L'objectif est de développer un moyen de faciliter la préhension d'objets et d'optimiser le temps d'atteinte avec l'aide d'un système d'analyse de scène rapide utilisant la vision et l'intelligence artificielle pour la préhension d'objets par un robot d'assistance.

Le mémoire par articles est choisi pour la présentation du projet de recherche. Le Chapitre 2 présente une revue critique de la littérature. Le Chapitre 3 présente la rationnelle du projet de recherche. Le Chapitre 4 présente les aspects méthodologiques/résultats complémentaires. Le Chapitre 5 présente l'article *Fast Scene Analysis Using Vision and Artificial Intelligence for Object Grasping by an Assistive Robot* soumis dans le Journal Engineering Applications of Artificial Intelligence. Le Chapitre 6 présente la discussion générale. Le Chapitre 7 présente la conclusion et les recommandations.

CHAPITRE 2 REVUE CRITIQUE DE LA LITTÉRATURE

Les problèmes de détection, de préhension et de reconnaissance d'objets par un robot sont d'actualité de nos jours. De nombreux groupes de recherche travaillent pour découvrir une approche pratique, rapide et avec le moins d'erreurs possible. Selon les méthodes utilisées dans le domaine, il y a différents problèmes à traiter. Certains analysent les scènes avec des images et d'autres plus récemment avec des acquisitions 3D. La segmentation de scène avec des images a beaucoup été étudiée et explorée dans le passé. La segmentation de scène 3D émerge ces dernières années avec les avancements de la technologie comme la caméra Kinect et semble promise à un bel avenir.

La littérature possède de nombreux ouvrages proposant des solutions techniques sur le sujet, pour ce qui est de la vision par ordinateur, le livre Computer Vision : A Modern Approach (Forsyth et Ponce (2012)) [4] est une référence dans le domaine de la vision artificielle, le livre d'intelligence artificielle (Russell et al. (2010)) [5] est aussi une référence répertoriant de notions pratiques pour résoudre des problèmes à partir d'algorithmes complexes.

Nous explorerons dans les prochaines sous-sections de ce chapitre différentes revues de la littérature qui ont permis de converger vers la solution afin de répondre à l'objectif du projet de recherche se divisant en quatre soit 1.Détection d'objets, 2. Préhension d'objets, 3. Reconnaissance d'objets et 4. Articles comprenant une analyse globale.

2.1 Détection d'objets

Commençons avec la capture d'images 2D et leur segmentation correspondant à un traitement de bas niveau où elle regarde la relation entre les pixels, la correction et le seuillage d'histogramme, le traitement de bruit et l'application de masque. Après plusieurs expérimentations avec des logiciels de traitements d'images, nous nous sommes rendu compte que la segmentation d'objets dans une image était très difficile afin d'être robuste pour tous les types de scènes. C'est pourquoi nous avons décidé de prendre une approche plus moderne d'acquisition de scènes 3D. À notre sens, c'est une bonne initiative puisqu'une scène est en fait un environnement 3D et non 2D. Au commencement, nous avons eu des craintes sur l'utilisation de l'analyse 3D, car, de façon logique, plus nous augmentons la dimension ou la mémoire d'enregistrement de données, plus il y a de l'information à traiter, ce qui amène des temps de calcul plus importants. Le temps de calcul est un

aspect primordial dans notre recherche de solutions et doit être le plus court possible. Cela amène à l'exploration de techniques pour la détection d'objets 3D dans des scènes.

(Medeiros Brito et al. (2008)) [6] ont expliqué comment reconstruire une surface 3D constituée d'un nuage de points avec un apprentissage adaptatif par un algorithme nommé Kohonen et un maillage adaptatif de l'ensemble pour obtenir une surface 3D d'un objet. Cela n'était qu'un exemple de nombreuses théories pour la reconstruction 3D.

Dans (Zhang et al. (2008)) [7], il y a eu le développement d'un système d'acquisition de scènes intérieures avec une plateforme rotative créant le déplacement d'une caméra stéréovision. Ils ont présenté la calibration, la reconstruction 3D et les expérimentations de leur système. Cette technique a permis une reconstruction 3D flexible de bonne qualité, mais elle n'était pas robuste pour des objets possédant très peu de texture, ce qui était un gros problème de la stéréovision.

Suite aux progrès technologiques (2010) [8], la Kinect V1 a fait son apparition en novembre 2010 pour la console Xbox 360 permettant d'acquérir des données de natures différentes, soit une image couleur RGB et une image de profondeur D d'où la provenance du terme RGB-D. Cette caméra active a été un outil flexible utilisé particulièrement dans l'industrie du jeu et de la robotique, ce qui a particulièrement capté notre attention.

Dans (Anwer et al. (2015)) [9], une application intéressante avec la Kinect V1 dans le domaine de la robotique a présenté le calcul des dimensions d'objets réels en utilisant une résolution dynamique pour les manipuler ou pour les éviter en convertissant les valeurs des pixels en valeurs physiques, ce qui pourrait être intéressant à utiliser comme technique.

(Yan Cui et al. (2013)) [10] ont développé un algorithme pour le balayage de formes 3D avec une caméra de profondeur pour un objet statique et une caméra en mouvement autour de l'objet pour le balayer. Puisque les techniques antérieures de la littérature n'atteignaient pas les objectifs de balayage souhaité, cet algorithme basé sur une nouvelle méthode a combiné la super-résolution 3D avec une approche d'alignement probabiliste qui prenait en compte les bruits du capteur. Cette technique a amélioré grandement le rendu d'une acquisition 3D et a permis le développement de nouveaux capteurs d'acquisition 3D.

En juillet 2014, la Kinect V2 est arrivée sur le marché. (Lachat et al. (2015)) [11] ont passé une série de tests pour regarder les performances de cette version afin de voir si elle pouvait être une alternative pour les caméras laser de faible gamme. Ils ont énoncé que ce nouveau capteur possède

quelques limitations dans l'acquisition de l'environnement avec des erreurs causées par le système lui-même. Par contre, la conclusion des tests était prometteuse considérant la précision augmentée de la mesure de profondeur et l'efficacité en général. La Kinect V2 était considérée comme un progrès réel pour le traitement de vision par ordinateur. On peut voir sur la Figure 2.1 les spécifications de la Kinect V2.

Figure 2.1 : Capteur Kinect V2 par Microsoft [8].

La Kinect V2 a été utilisée dans le travail (Yang et al. (2015)) [12], où l'évaluation de la mesure de profondeur sur la distribution de la précision dans un cône de vision y est présentée. Les résultats ont démontré que la Kinect V2 a une précision acceptable lorsque l'acquisition reste dans le cône de vision. Par contre, cet article a rapporté que les objets avec matériaux réfléchissants peuvent conduire à un problème avec la lumière émise par le capteur rendant les valeurs de profondeur moindrement fiables. Des objets noir carbone absorbant la lumière causaient moins de lumière infrarouge réfléchie vers la caméra et le bloc d'alimentation qui requérait encore trop d'énergie étaient aussi des limitations supplémentaires.

Cependant, la Kinect V2 reste notre choix car elle est facilement disponible, simple d'utilisation et aussi très abordable. La Kinect de deuxième génération ne vient pas seule, le Software Development Kit (SDK) 2.0 (logiciel gratuit) comprend plusieurs codes de base en C++ et en C# pour commencer à utiliser la Kinect V2 et ses caractéristiques rapidement.

Kinect Fusion est un programme d'acquisition 3D qui est présenté par (Izadi et al. (2011)) [13] que l'on retrouve dans le SDK 2.0 (2016) [14]. Ce programme est utilisé pour obtenir l'acquisition surfacique des scènes dans notre étude. Il donne directement un nuage de points d'un

environnement enregistrable en trois formats différents, soit STL, OBJ ou PLY. L'interface de Kinect Fusion C# est présentée à la Figure 2.2.

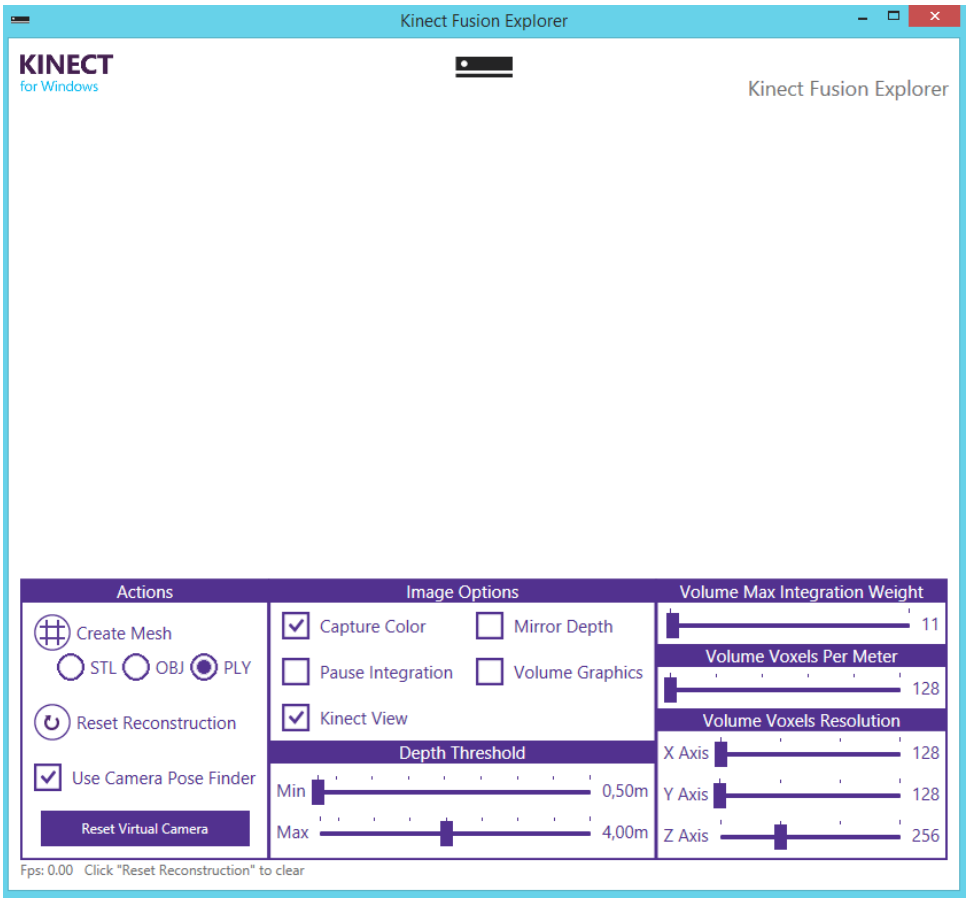

Figure 2.2 : Interface de Kinect Fusion C#.

L'utilisation de Kinect Fusion nécessite l'ajustement de paramètres comme le seuil de profondeur (la distance entre la Kinect V2 et la scène) qui varie de 0.5 à 4.5 mètres et les paramètres volumiques dont le volume maximum d'intégration des poids, le volume de voxels par mètre et le volume de résolution de voxels en XYZ. Après quelques tests, en vérifiant nos besoins pour le calcul en temps réel, nous avons choisi d'enregistrer les acquisitions surfaciques dans le format PLY. Ce format contient six informations essentielles pour un point dans l'espace, soit les trois coordonnées XYZ de l'objet et ses trois couleurs RGB afin d'obtenir toutes les informations pertinentes d'une acquisition RGB-D. Après avoir observé les changements de ces paramètres en opération, nous avons choisi les paramètres présentés à la Figure 2.3.

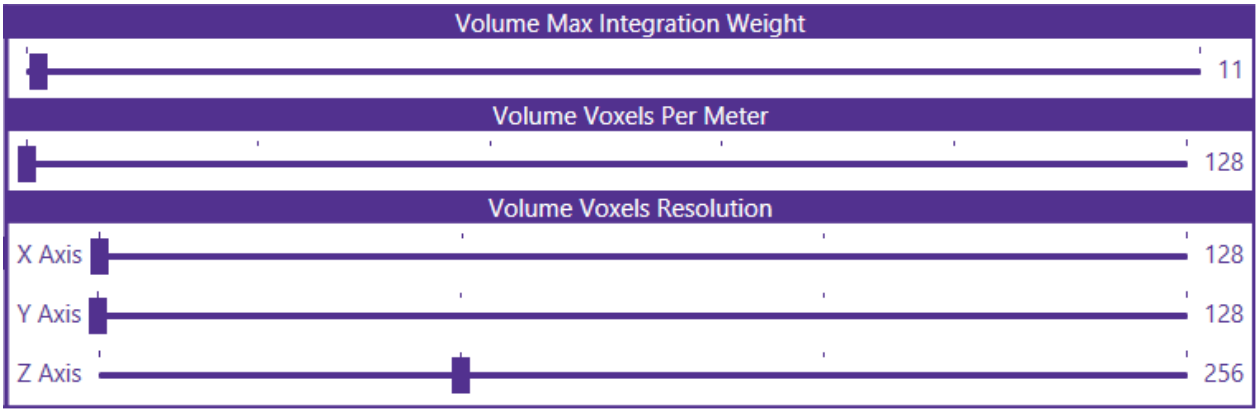

Figure 2.3 : Paramètres utilisés avec Kinect Fusion C#.

Après l'acquisition 3D, (Anguelov et al. (2005)) [15] ont présenté la méthode de Markov Random Fields permettant de détecter et de segmenter des objets complexes par apprentissage de surfaces et de caractéristiques volumiques. Cette méthode a permis de trouver une cohérence statistique permettant de différencier des objets qui n'ont pas de parties similaires comme un arbre, un bâtiment, etc. Par contre, cette méthode n'a pas été retenue, car elle a de la difficulté avec la segmentation des objets qui se ressemblent.

(Mian et al. (2005)) [16] ont présenté un résumé comparatif des algorithmes employés dans la littérature (Random Sample Consensus, Graph Matching, Spin-Image Maching, Huber's Framework, etc.) pour la segmentation 3D à partir de critères comme la capacité de fonctionner sur des objets à forme libre, l'efficacité du temps de calcul et bien d'autres. Ils ont démontré qu'aucun algorithme actuel ne peut répondre aux critères et surtout aucun ne le fait en un temps efficace. Selon cette référence, le meilleur algorithme serait Huber's Framework.

(Rusu et al. (2009)) [17] ont proposé la segmentation d'un nuage de points 3D sur des surfaces de support horizontal comme une table pour soutenir des objets dans un environnement domestique. Ils ont estimé les formes primitives d'un objet comme les sphères, les cylindres, les cônes et les prismes. La reconstruction 3D des objets, selon les primitives, a permis d'inférer des parties non visibles durant l'acquisition due à des occlusions afin d'augmenter la précision de reconnaissance et de faciliter leur préhension par un robot. Concevoir une base de données d'objets 3D pour aider la détection, la segmentation et la reconnaissance a permis de contrer les parties non visibles des objets. Par contre, faire une recherche dans une grande base de données prenait beaucoup de temps, alors cette méthode n'a pas été retenue.

Un résumé comparatif récent (Nguyen et Le (2013)) [18] a exposé les différentes méthodes proposées pour la segmentation 3D de nuages de points en présentant leurs avantages et désavantages. Ils ont reporté que la segmentation 3D en temps réel n'est pas encore un succès total, mais en général les chercheurs utilisent deux approches.

La première approche utilise des modèles purement mathématiques et des techniques de raisonnement géométrique tel que la région de croissance ou l'ajustement de modèles, en combinant avec des estimateurs robustes pour s'adapter à des modèles linéaires et non linéaires de nuage de points. Cette approche est rapide et permet d'obtenir de bons résultats dans des scènes simples. Par contre, il est difficile de choisir la dimension du modèle quand l'ajustement d'objets et/ou la sensibilité aux bruits ne fonctionnent pas bien dans des scènes complexes.

La seconde approche extrait des caractéristiques 3D de nuage de points en utilisant des descripteurs de caractéristiques et utilise des techniques d'apprentissage machine pour apprendre les classes de différents objets et utiliser le modèle résultant pour la classification de données. Cette deuxième approche surpasse la première selon la référence, car trouver et s'ajuster aux géométries de primitives complexes d'objets en raison du bruit et des occlusions dans les acquisitions rend la première approche très difficile à implémenter. En nous basant sur cette référence, nous nous sommes orientés vers la segmentation 3D et l'extraction de caractéristiques des objets dans une scène pour les présenter à un classificateur avec une technique d'apprentissage supervisé tel un réseau de neurones.

2.2 Préhension d'objets

(Calli et al. (2011)) [19] ont présenté un algorithme pour prendre des objets dans un environnement non structuré avec l'utilisation de la vision active et des informations de maximisation de courbure des objets. Ils ont utilisé une boucle d'asservissement pour contrôler l'approche de l'effecteur du robot avec une caméra alimentant l'analyse Elliptic Fourier Descriptor, transformant l'image en modèle 2D, calculant la courbure pour positionner les endroits de préhension et envoyant aux actionneurs du robot les commandes. Cet algorithme a été assez rapide pour être utilisé en temps réel selon la référence, mais ils n'ont pas donné la valeur chiffrée et leur méthode semble complexe pour attraper des objets.

(Lippiello et al. (2013)) [20] ont présenté une méthode de reconstruction visuelle élastique des surfaces de préhension jusqu'à ce que cette reconstruction s'ajuste à l'enveloppe d'un objet. Leur méthode utilisait un algorithme de préhension qui fonctionnait en parallèle pour minimiser le temps de préhension en bougeant en même temps qu'il analysait l'acquisition. Les modèles mathématiques ont été modélisés de façon à représenter chaque point comme étant une masse soutenue par des ressorts et un amortisseur vers d'autres points afin d'épouser la forme d'un objet. L'algorithme déplaçait les doigts de manière à respecter la prise de l'objet en fonction de la répartition du poids et de la configuration cinématique de l'effecteur. Cette méthode s'adaptait complètement aux objets et prenait en considération la configuration cinématique de l'effecteur, mais elle semblait excessive pour simplement prendre un objet au quotidien.

Pourquoi utiliser un modèle analytique complexe pour la préhension d'objets? Nous explorons, dans le cadre du projet de recherche, une méthode plus simple basée sur l'analyse des dimensions combinées avec le centre géométrique de l'objet pour leur préhension. Cela permettra de voir si une telle méthode peut être aussi opérationnelle que les méthodes complexes de la littérature.

2.3 Reconnaissance d'objets

La reconnaissance d'objets est un problème d'apprentissage supervisé. Ce processus est généralement connu sous le nom d'apprentissage inductif. L'objectif est d'apprendre un ensemble de données pour en tirer une règle de décision non linéaire en optimisant certains critères. Ces critères sont le nombre d'époques maximum, le taux d'apprentissage, le nombre de couches cachées et bien d'autres paramètres avec un réseau de neurones dans notre étude. Déjà en 1990, les réseaux de neurones démontraient leur potentiel pour la classification.

Dans (Lee et al. (1990)) [21], ils ont présenté l'utilisation de la texture comme une caractéristique aidant grandement la classification. De plus, ils ont démontré que plus un modèle est non linéaire, moins l'apprentissage supervisé nécessite de données pour l'entrainement comparé à une approche linéaire.

ImageNet a utilisé un ensemble de données comprenant plus de 1.2 million d'images à haute résolution avec 1000 différentes classes et a été entrainé par un réseau de neurones à convolution profonde. Ce travail développé dans (Krizhevsky et al. (2012)) [22] a démontré qu'un tel réseau a été capable de dépasser les records de classification en utilisant uniquement l'apprentissage

supervisé. Ils ont noté que modifier la structure du réseau en enlevant des convolutions dégrade le résultat de la classification. Nous apprenons de cette référence que les réseaux de neurones sont adaptés pour reconnaitre des données après un apprentissage supervisé intensif pour généraliser des modèles non linéaires avec des fonctions de prise de décision. Nous apprenons aussi que l'interconnexion entre les neurones est un aspect important pour l'apprentissage.

(Shin et al. (2016)) [23] ont démontré que les progrès dans la reconnaissance avec un réseau de neurones à convolution profonde ont été très utiles et performants pour des applications pratiques comme le diagnostic d'imagerie médicale.

Ces références ont montré que la reconnaissance d'objets avec un apprentissage supervisé par un réseau de neurones est d'actualité et est très avantageux pour la classification d'un grand ensemble de données pour plus d'un domaine. C'est pourquoi nous utilisons ce type de classificateur pour classer nos données.

2.4 Articles comprenant une analyse globale

Pour la détection et l'obtention de caractéristiques d'objets, (Johnson et Hebert (1999)) [24] ont présenté une technique innovante de descriptions d'objets, le spin-images, pour créer un grand ensemble de données d'objets 3D de manière efficace. Cette méthode a permis d'aider à résoudre des problèmes de superpositions, de bruits, de désordres et d'occlusions lors de la détection d'objets en trouvant de bons descripteurs pour les surfaces. Après la compression du spin-images, leur algorithme utilisait une analyse de composantes principales pour déterminer les caractéristiques les plus pertinentes des images et ainsi diminuer l'ordre de complexité. Par contre, le temps de calcul de cette technique était long et variait grandement dépendamment de la surface analysée.

Dans (Huber et al. (2004)) [25], ils ont présenté une méthode intéressante sur la détermination de parties de base d'un objet 3D dans une scène. Chaque partie d'un objet a été représentée comme une classe comparativement à l'approche classique qui identifie un objet comme une classe générale. Ils ont utilisé une extraction de parties des objets à partir d'un entrainement et ils ont regroupé les parties avec un algorithme de classement hiérarchique. Par la suite, ils ont combiné les parties pour reconnaitre l'objet dans son ensemble. Cette décomposition des objets classés en plusieurs sous-classes pourrait être intéressante, mais complexe et difficile à implémenter.

(Mian et al. (2006)) [26] ont présenté une manière plus efficace d'obtenir des vues multiples des objets 3D pour former un grand ensemble de données à partir de modèles synthétiques et réels. La librairie des modèles contenait les modèles 3D ainsi que leurs vues multiples dans une table. Ils ont reporté également une reconnaissance de 95 % et leur algorithme n'était pas sensible à la dimension d'un modèle de librairie comme avec l'algorithme du spin-images. En utilisant un ordinateur et une scène similaire aux articles présentant le spin-images, ils ont obtenu un temps de 6 minutes comparé à 480 minutes, ce qui a diminué le temps de calcul et a rendu l'algorithme plus efficace. Dans une scène complexe, leur méthode, pour détecter et segmenter un objet, prenait moins de 2 minutes, ce qui reste tout de même élevé pour notre application.

Dans (Rusu et al. (2010)) [27], ils ont utilisé un descripteur de caractéristiques 3D nommé Viewpoint Feature Histogram encodant la géométrie et les points de vue d'un nuage de points. Cette méthode fonctionnait avec des acquisitions bruitées et/ou un manque d'information de profondeur en utilisant la stéréovision pour détecter la géométrie et la pose des objets. Les auteurs ont obtenu 98,5 % pour la reconnaissance d'objets, ce qui serait un but à atteindre. Cependant, ils n'ont pas rapporté le coût en temps de calcul, mais nous supposons que le processus d'analyse doit prendre un temps élevé.

(Lai et al. (2011)) [28] ont présenté un arbre hiérarchique de 51 classes d'objets acquis en RGB-D avec la Kinect V1. Ils ont démontré que la combinaison de la couleur et de la profondeur ont été des informations pertinentes pour obtenir de bons résultats de reconnaissance d'objets, ce qui nous a intéressés. Ils ont créé leur ensemble de données en utilisant une table tournante, en supprimant l'arrière-plan et en appliquant le Random Sample Consensus pour trouver la surface de leur plateau rotatoire. Puis, ils ont utilisé l'algorithme de spin-images et le descripteur Scale-invariant Feature Transform pour obtenir les caractéristiques des images. Ensuite, Linear SVM, Gaussian Kernel SVM et Random Forest ont été utilisés pour classifier les objets et ils ont obtenu 90.5 % de reconnaissance. Leur algorithme prenait un temps de 10 secondes, ce qui était encore un peu trop long pour l'analyse.

(Tang et al. (2012)) [29] ont présenté un système pipeline comprenant l'acquisition 3D, la segmentation des objets sur une table, la transformation en maillage, l'extraction de caractéristiques avec la Scale-Invariant Feature Transform vers un histogramme et une vérification de la pose. Ils ont obtenu une reconnaissance de l'ordre de 90 % pour des scènes complexes. Par contre, rouler

leur pipeline d'analyse d'objets prenait 20 secondes sur une scène avec un ordinateur 6-cœur 3.2 GHz i7 de 24 GB de RAM prenant trop de temps.

(Lai et al. (2014)) [30] ont présenté un algorithme de reconnaissance d'objets par apprentissage de caractéristiques non supervisé, Hierarchical Matching Pursuit for 3D Voxel Data. Leur algorithme prenait en compte une acquisition RGB-D et traitait deux caractéristiques, soit la couleur et la profondeur avec la méthode Markov Random Fields [15]. Ils ont expliqué qu'utiliser une image 2D était intéressant, car capturer un objet en avant de son arrière-plan donnait des informations contextuelles et qu'utiliser une acquisition 3D donnait une information spatiale importante. Leur précision globale pour la classification d'objets était de 95,3 % et le temps pour extraire les caractéristiques d'une scène était de 4 secondes.

Ce dernier temps de calcul est raisonnable et la performance de la reconnaissance est bonne, ce qui nous permet de croire à l'obtention de la réalisation de l'objectif d'analyse en temps réel qui devrait être opérationnel dans un futur proche.

CHAPITRE 3 RATIONNELLE DU PROJET DE RECHERCHE

En résumé, certaines méthodes de la littérature sont déjà très prometteuses et avancées, leur détection d'objets peut détourner les occlusions sur des scènes complexes et offrir une reconstruction de formes pour améliorer la préhension et la reconnaissance des objets. Certains innovent dans des modèles complexes et opérationnels pour la préhension d'objets et d'autres ont obtenu de bons résultats pouvant aller jusqu'à 98,5 % de performance pour la reconnaissance d'objets.

3.1 Problématiques

Dans la littérature, une des problématiques est que les méthodes de détection d'objets actuelles les plus robustes prennent encore beaucoup de temps de calcul, empêchant l'automatisation de la préhension d'objets par des robots dans un temps acceptable pour l'assistance des utilisateurs au quotidien. La plupart des méthodes de détection d'objets utilisent des algorithmes statistiques coûteux en calcul machine rendant la détection non praticable en temps réel. Il n'existe aucune méthode de détection d'objets qui soit rapide, simple et robuste. Également, après l'analyse des objets détectés, certains groupes de chercheurs innovent dans des modèles complexes pour la préhension d'objets qui sont opérationnels, mais très difficiles à mettre en œuvre et certains obtiennent de bons résultats pour la reconnaissance d'objets sur des données.

3.2 Objectif général

L'objectif général de cette étude est de développer un système d'analyse de scène rapide utilisant la vision et l'intelligence artificielle pour la préhension d'objets par un robot d'assistance. Les algorithmes internes doivent permettre de répondre à l'ensemble des questions suivantes plus rapidement que les méthodes existantes : 1. Combien y a-t-il d'objets et où se situent-ils? 2. Comment les saisir, c.-à-d. quels sont les endroits de préhension efficaces et quelle est la phase d'approche à donner au robot? 3. Quels sont ces objets de la scène, reconnus à partir d'un apprentissage neuronal sur un ensemble de données acquis avec une caméra active?

3.2.1 Objectifs spécifiques

1. Développer une technique de détection d'objets plus rapide et simple.
2. Développer une technique d'apprentissage supervisé par un réseau de neurones.
3. Développer une technique de préhension d'objets simple, rapide et efficace.
4. Développer une technique de reconnaissance d'objets aussi performante que la littérature.

3.3 Hypothèse générale

L'hypothèse générale de recherche repose sur le fait que le temps nécessaire pour la détection, la préhension et la reconnaissance d'objets par un robot peut être grandement diminué en analysant plus efficacement une scène. Cette hypothèse sera réfutée si le temps de calcul de ces analyses n'est pas au moins deux fois plus petit que le temps avec les méthodes traditionnelles, qui est à notre connaissance de 4 secondes pour une scène [30].

CHAPITRE 4 ASPECTS MÉTHODOLOGIQUES/RÉSULTATS COMPLÉMENTAIRES

4.1 Solution technologique

Dans ce contexte, un des défis actuels majeurs concerne l'automatisation de la détection, de la préhension et de la reconnaissance d'objets par les robots dans des scènes. Aujourd'hui, l'utilisation de la segmentation de scène 3D se démocratise de plus en plus, par exemple les travaux de Zug et al. (2012) [31] présentant des applications pour la construction de cartes, la localisation et les évitements d'obstacles utilisant la Kinect. La Kinect a ouvert de nouveaux horizons vers la reconnaissance d'objets, la réalité augmentée et bien d'autres domaines comme l'expliquait la revue présentée dans (Cruz et al. (2012)) [32]. Cette technologie a permis d'acquérir des nuages de points de scènes possédant de l'information sur les coordonnées de couleur *RGB* et de profondeur *D* d'où la provenance du terme *RGB-D*.

Pour obtenir nos acquisitions 3D, la caméra active Kinect V2 (Microsoft, USA) sortie en juillet 2014 a été utilisée pour notre système. Cette caméra est dédiée à la base pour le monde du loisir et du jeu vidéo avec la console Xbox One. Cependant, la composition de cette caméra active, soit une caméra couleur haute résolution et un capteur de balayage infrarouge pour la profondeur, la rend très intéressante pour la recherche et le développement.

En comparaison, la technologie utilisée pour obtenir la profondeur avec la Kinect V1 est la lumière structurée tandis que pour la Kinect V2, le principe est basé sur le temps de vol. Le temps de vol est une technique balayant la scène mesurée par une lumière infrarouge en calculant le temps que prend cette lumière pour effectuer le trajet entre la surface de la scène et le capteur. Le temps de vol est proportionnel à la distance entre la caméra et la scène mesurée en mètre. Cette mesure est faite sur chaque point d'une surface, ce qui permet d'obtenir une représentation 3D surfacique complète de la scène. La Figure 4.1 montre l'apparence des deux versions de Kinect et le Tableau 4.1 présente des spécifications comparatives.

Figure 4.1 : Comparatif entre les deux versions de la Kinect.

Tableau 4.1: Spécifications comparatives entre les deux versions de la Kinect

Caractéristique	Kinect pour Windows 1	Kinect pour Windows 2
Caméra couleur	640 x 480 @ 30 fps	1920 x 1080 @ 30 fps
Caméra profondeur	320 x 240	512 x 424
Distance max en profondeur	~ 4.5 m	~ 4.5 m
Distance min en profondeur	40 cm	50 cm
Angle de vue horizontale	57 degrés	70 degrés
Angle de vue vertical	43 degrés	60 degrés
Moteur rotation	oui	non
Joints définis d'un squelette	20 joints	26 joints
Squelette complétement suivi	2	6
Standard USB	2.0	3.0
OS supporté	Win 7, Win 8	Win 8
Prix	299 $	160 $

La version Kinect V2 avec de meilleures performances que la première version amène alors à concevoir une solution rapide et fonctionnelle pour l'acquisition 3D de scènes pour en détecter les objets à prendre.

4.2 Programmes et résultats

La Kinect V2 possédait déjà des qualités remarquables pour l'acquisition surfacique de scènes en trois dimensions avec son système de balayage laser. Le capteur nous a permis de débuter le développement de programmes pour exploiter sa puissance. Différents programmes ont été conçus dans la cadre du projet de recherche afin d'obtenir une solution rapide et simple.

4.2.1 Analyse de formes et de dimensions 3D

L'algorithme d'analyse de formes et de dimensions 3D permet d'analyser un nuage de points STL d'un objet quelconque en déterminant une forme qui lui correspond le mieux parmi une sphère, un cylindre ou un prisme rectangulaire en calculant la différence d'erreur volumique. Puis avec la forme, l'algorithme détermine les dimensions de cet objet en fonction du volume correspondant et enregistre le tout dans une base de données.

Prenons un exemple pour illustrer un résultat obtenu avec l'algorithme de formes et de dimensions 3D. Voici le nuage de points acquis d'une boite de jus reconstruite en 3D sur la Figure 4.2. La forme réelle de cet objet est un prisme rectangulaire et de dimensions $50 \times 40 \times 120$ mm.

Figure 4.2 : Nuage de points d'une boite de jus versus l'image d'origine.

Voici les graphiques sur les correspondances des trois formes (cylindre, prisme rectangulaire et sphère) sur la boite de jus, ainsi que le résultat obtenu.

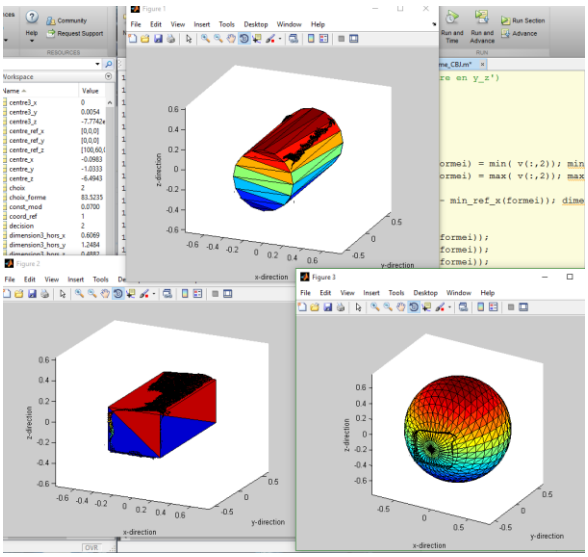

Figure 4.3 : Graphiques de l'algorithme de formes et de dimensions 3D.

```
Command Window
  Fonction: lire_STL()     Version 1.0
  Fichier: volume_base\cylindre_d20h200.stl;
  Fonction: lire_STL()     Version 1.0
  Fichier: volume_base\prismeRect.stl;
  Fonction: lire_STL()     Version 1.0
  Fichier: volume_base\sphere.stl;

  erreur_volume =

    170.8228   83.5235  824.7964


  La forme moyenne est un Prisme Rectangulaire
  De cote: en x: 53.685410 mm, en y: 117.839193 et en z: 41.822013 mm
```

Figure 4.4 : Résultats de la décision de l'algorithme de formes et de dimensions 3D.

En observant la fenêtre de commande des résultats, on peut observer le calcul des erreurs de volume pour chacune des formes dans cet ordre, cylindre, prisme rectangulaire et sphère. On remarque que la décision qui en découle est que la forme de l'objet analysé est un prisme rectangulaire de dimensions 53.5 × 41.8 × 117.8 mm, ce qui est assez proche des dimensions exactes de la boite de jus réel. En effet, après plusieurs tests, il est possible de voir une généralisation d'une erreur maximale de 5 mm sur les dimensions des objets. Cela est satisfaisant pour la suite du système. Le code de cet algorithme est observable dans l'Annexes

Annexe A.

4.2.2 Base de données simple

La base de données simple est un support pour l'algorithme précédant et suivant. Ce programme avait pour but de conserver et d'ajouter de nouveaux objets acquis et de faire le lien entre les autres algorithmes. La base de données simple comprend le id, le nom, la forme, les dimensions dépendamment de la forme, les données des points dans l'espace, les données des triangles du nuage de points composant tous les points du nuage analysés par l'algorithme d'analyse de formes et de dimensions 3D. Le code de cette base de données simple est observable dans l'Annexes

Annexe B.

Une fois analysées, les caractéristiques de l'objet sont enregistrées dans une base de données simple sous la forme suivante :

1	'Test'	'cylindre'	70	120	0	0	0
2	'boite jus'	'prisme'	0.5148	1.2164	0.4879	<7664x3 do...	<15076x3 d...
3	'bouteille h...	'cylindre'	0.0725	0.2457	0	<10732x3 d...	<21244x3 d...
4	'cube'	'prisme'	5.1069	6.3175	6.1234	<4377x3 do...	<8521x3 do...
5	'balle tennis'	'sphere'	0.0751	0	0	<26x3 doub...	<48x3 doub...
6	'balle_tennis'	'sphere'	0.0704	0	0	<6245x3 do...	<12094x3 d...

Figure 4.5 : Base de données simple.

4.2.3 Algorithme de convolution 3D

Les étapes précédentes complètent l'ajout d'objets dans la base de données simple nécessaire pour le système. Les étapes suivantes devraient pouvoir être exécutées en temps réel afin d'assister la phase d'approche de préhension d'un objet en position et en orientation. Pour la préhension, après l'activation du système par l'utilisateur, l'acquisition de la scène par la Kinect V2 et la communication de l'objet désiré avec une interface quelconque, l'algorithme de convolution 3D analyse la scène pour retourner une matrice de position/orientation de l'effecteur du robot.

L'algorithme de convolution 3D débute par le chargement d'un nuage de points d'une scène. Ensuite, elle utilise une fonction de transformation du nuage de points en une matrice de 0 et de 1, où 1 représente un point dans l'espace. Pour ce faire, l'algorithme trouve le centre du nuage de points et le convertit en le centre de la matrice de 0 et de 1. Puis, il convertit de façon linéaire les points du nuage en points vers la matrice remplie de 0 en mettant un 1 au bon endroit selon un pas de précision qui dépend de la résolution des voxels d'acquisition des nuages. Une fois la

transformation complétée, selon l'objet désiré d'un utilisateur, le programme va rechercher dans la base de données simple le nuage de points de l'objet. Ensuite, il applique une convolution stratégique en fonction de la forme de l'objet désiré pour retrouver la position et l'orientation de cet objet dans la scène.

L'analyse de l'orientation et de la position dans l'espace par l'algorithme de convolution 3D donne le résultat suivant pour un exemple d'une bouteille sur une table :

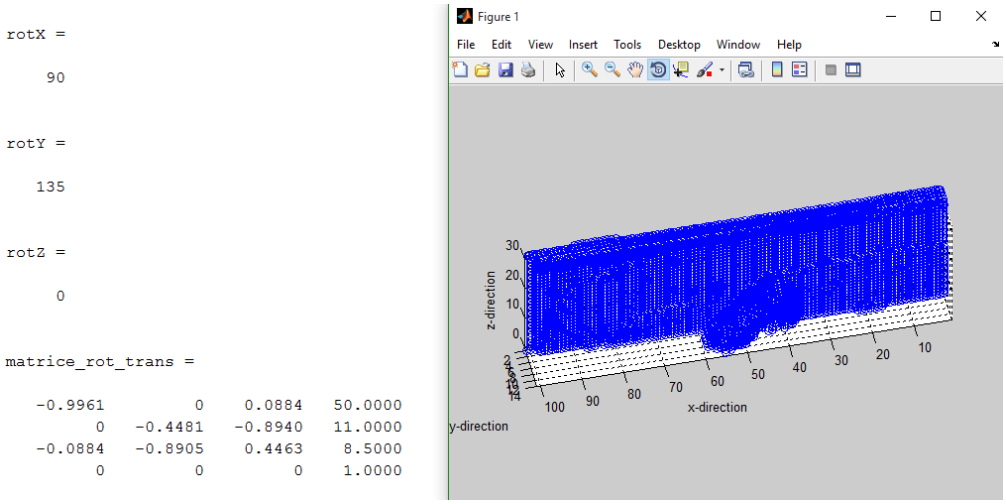

Figure 4.6 : Résultats de l'algorithme de convolution 3D.

Nous pouvons voir l'affichage de la matrice de 0 et de 1 de la scène. Dans cette situation, la bouteille est couchée sur la table et tournée d'un angle de 135 degrés. Nous pouvons remarquer dans la fenêtre de commande l'analyse suivante, l'objet a fait une rotation de 90 degrés autour de l'axe X pour spécifier que la bouteille est couchée sur la table et une rotation de 135 degrés autour de l'axe Y, ce qui est un bon résultat. Également, nous pouvons observer la matrice de rotation/translation pour l'effecteur du robot afin qu'il puisse aller prendre l'objet sur la table par le centre de l'objet dans la scène. Dans le cas montré, 50 en x, 11 en Y et 8.5 en Z sont des coordonnées relatives qui pourront être retrouvées avec le robot en appliquant des changements de repères. Le code de cet algorithme est observable dans l'Annexes

Annexe C.

4.2.4 Analyse de scène rapide

Le système était alors opérationnel, mais la recherche dans la base de données simple et la convolution 3D prenait un temps trop élevé pour conserver cette solution. Une nouvelle solution

émergea de ce problème pour obtenir la conception du système actuel qui est présentée en détail dans la méthodologie du Chapitre 5. Cette solution réutilise la fonction de transformation précédemment utilisée pour l'algorithme de convolution 3D durant l'acquisition d'une scène pour la détection d'objets afin de faire une détection d'objets innovante, le palpage par le haut. Ensuite, nous avons ajouté une analyse de scène des objets détectés avec un arbre de décisions pour trouver les endroits de préhension adaptés aux objets et un apprentissage supervisé par un réseau de neurones pour la reconnaissance des objets dans une scène faisant partie d'un ensemble de données connu. Le code de l'algorithme d'analyse de scène rapide est observable dans l'Annexes

Annexe D.

CHAPITRE 5 ARTICLE 1: FAST SCENE ANALYSIS USING VISION AND ARTIFICIAL INTELLIGENCE FOR OBJECT GRASPING BY AN ASSISTIVE ROBOT

C. BOUSQUET-JETTÉ, S. ACHICHE and M. RAISON

Soumis le 2 août 2016 au journal *Engineering Applications of Artificial Intelligence*

5.1 Abstract

Vision-assisted robotic aid is flourishing, particularly for people affected by age-related loss of mobility and people subject to musculoskeletal disorders. In this context, a major current challenge remains automated scene analysis and robotic prehension of objects. Specifically, the most robust of current segmentation methods are still computationally intensive, preventing the automation of robotic object prehension in a manner quick enough to be acceptable for everyday aid.

The objective of this study is to develop a fast scene analysis using vision and artificial intelligence for object grasping by an assistive robot. The algorithm must provide answers to all the following questions more quickly than existing methods do: 1. How many of the objects are there, and where are they located? 2. Which coordinates on the objects are effective prehension targets and what is the favored path of approach for the robot? 3. What are the objects in the scene, as identified by a neural network on data from an active camera?

The solution has been developed using the Kinect V2 vision system (Microsoft, USA) and the JACO robotic assistance arm (Kinova, Canada). Following the acquisition of scene data (180 scenes) with an object in each scene, the solution was developed in three stages: 1. Object detection involving transformation of raw scenes into data matrices and 3D scene segmentation to find the objects, by means of a novel algorithm for “top-down probing”. This is followed by elimination of undesirable points based on their gradients. 2. After object detection, supervised learning is performed on the objects in the dataset. 3. Scenes containing the objects are analyzed, which includes identification of prehension targets on the objects using a simple decision tree, and selection of the path of approach of the robotic arm. Subsequently, a neural network performs

object recognition utilizing surface geometry and RGB color, yielding 83% performance in a controlled environment.

This study has shown that scene analysis for robotic prehension of objects in cooperation with a user can be performed with effective promptness. Indeed, the system requires on average 0.6 s to analyze an object in a scene. With the JACO robotic assistance arm, the system can pick up a requested object in 15 s while moving at 50 mm/s, which may be greatly improved upon using faster robots. The system performance averages 83% for object recognition and is able to use a decision tree select a simple path for the robot effector to approach a requested object. This system, in combination with an assistive robot, has great potential for providing users with improved autonomy and independence, and for encouraging sustained usage of technical aid.

Abbreviations:

3D	Three spatial dimensions
OBJ	3D object file format
PLY	“Polygon File Format”
RGB	Red (R), green (G), and blue (B)
RGB-D	RGB with depth D
SDK	Software Development Kit
STL	“Standard Triangle Language” file format
XYZ	3D Cartesian spatial coordinates

5.2 Introduction

Vision-assisted robotic assistance is flourishing, particularly for people affected by age-related loss of mobility and people subject to musculoskeletal disorders. Such robots are often equipped with grippers to improve the autonomy and the capabilities of users who are subject to upper limb motor limitations. Users may be assisted in daily tasks such as manipulating objects, opening doors, and operating electrical switches. To control the robots arm the user must manipulate a joystick attached to the armrest of a motorized wheelchair. The effectiveness of robot usage in this kind of context was evaluated by (Maheu et al., 2011), which concludes that such robots are very useful to

users. However, the operation of such robots requires good manual dexterity and may therefore be demanding for certain users, which may cause object manipulation to be too slow and make complex operations tiring and frustrating. Indeed, ease of use of this type of robotic aid and the ability to manipulate objects reasonably quickly are essential criteria for sustained usage of robotic aids. Consequently, it is imperative to design a means of facilitating object prehension and of optimizing the time required for operations that makes use of computer vision and artificial intelligence.

In this context, the current challenge is to automate fast scene analysis using vision and artificial intelligence for object grasping by an assistive robot. 3D scene segmentation is increasingly widespread, with technological advancements becoming available such as the Kinect V1 active camera (Microsoft, USA) introduced in November 2010, which still has a promising future. This camera is a flexible tool used mostly in the gaming industry but also in robotics, as reported by (Zug et al., 2012), which presents applications in mapping, object localization, and obstacle avoidance. This technology opens new avenues for object recognition, augmented reality, and many other fields, as reported by (Cruz et al., 2012).

Some existing methods are very promising and advanced, with 3D segmentation able to avoid occluding obstacles in complex scenes and provide shape reconstruction for improved prehension. Some methods develop innovative, complex, functional models for object prehension, and some have obtained good results for object recognition with up to 98.5% performance. Nonetheless, the most robust of the current segmentation methods are still computationally intensive, preventing automated object prehension by robots quickly enough to be acceptable for everyday aid.

For example, the Markov Random Fields method (Anguelov et al., 2005) enables detection and segmentation of complex objects by learning surface and volumetric features to establish a statistical model that allows objects with dissimilar parts to be distinguished, such as a tree, a building, and so on. The Markov Random Fields method does however have difficulty distinguishing objects that are similar to one another. (Mian et al., 2005) presents a comparative summary of the algorithms used in the literature (Random Sample Consensus, Graph Matching, Spin Image Matching, Huber's Framework, etc.) for 3D segmentation on the basis of criteria including the ability to function on free-form objects, computational efficiency, and many others.

The article shows that no current algorithm satisfies the criteria and that none are quick enough. According to the reference, the best current algorithm uses Huber's Framework.

Given that no quick, simple, automated method exists for a fast scene analysis using vision and artificial intelligence for object grasping by an assistive robot, accordingly, the **objective** of this study is to develop such an algorithm. The algorithm must provide answers to all the following questions more quickly than existing methods do: 1. How many of the objects are there, and where? 2. How should the objects be grasped, i.e., which prehension targets on the objects are effective and what is the favored path of approach for the robot? 3. What are the objects in the scene, as identified by a neural network using data from an active camera?

The research hypothesis is that the time required for a robot to prehend an object may be greatly reduced by improving the efficiencies of scene analysis and analysis of prehension mode. This hypothesis would be refuted if, in comparison with traditional methods, diminishing the computation times of these analyses by half were not achieved. As to our knowledge, the best analysis time for one scene was 4 s (Lai et al., 2014), the time required should be therefore less than 2 s.

5.3 Literature Review

This section presents the state of the art in object detection, object prehension, and object recognition.

(Johnson and Hebert, 1999) presented an innovative method known as the spin-images method, which efficiently generated large datasets for 3D objects. This was done by identifying good descriptors of surfaces, for the sake of object detection and in identification of object features, namely, superposition, noise, disorder, and occlusion. After the spin-images compression, the algorithm used principal component analysis to determine the most significant features of the images and thereby lowers the order of complexity. However, the computation time with this technique was long and highly variable depending on the surface analyzed.

(Mian et al., 2006) presented a more efficient way of obtaining multiple views of 3D objects to generate a large dataset from real and synthetic models. The library contains a table of 3D models with multiple views. The authors of the reference report 95% recognition, and unlike the spin-images method, their algorithm is insensitive to the dimensions of library models. They obtained a

computation time of 6 minutes compared to the 480 minutes needed for spin-images using similar computer and scene. The algorithm still requires at least 2 minutes for a complex scene, which we consider too long for our application.

(Rusu et al., 2009) proposed a method of segmenting a cloud of 3D points located on a horizontal support surface, such as a table in a domestic environment. That reference estimated the primitive shapes of an object: spheres, cylinders, cones, and prisms. The reconstruction of 3D objects in terms of primitive shapes allowed occluded or otherwise non-visible parts to be inferred during acquisition to facilitate precision and robotic prehension. Designing a 3D object database to aid detection, segmentation, and recognition helped contend with non-visible parts of objects. Nevertheless, performing searches in a large database took time.

(Rusu et al., 2010) included a descriptor for 3D features called Viewpoint Feature Histogram, which works with data that is noisy and lacks depth information. This was done by detecting object geometry and pose using stereovision. The authors of that reference report 98.5% object recognition. They do not report the computational weight, but we surmise that the computation time is lengthy.

For prehension, (Lippiello et al., 2013) presented a method for elastic visual reconstruction of prehension surfaces which ran until the reconstruction fitted the object envelope. Furthermore, the authors proposed a prehension algorithm that ran in parallel with the scene analysis to minimize the total time. Mathematically, the method modeled the points as masses linked by springs and dampers, so as to marry the points to the object envelope. The algorithm moved the robot fingers based on the effector kinematic configuration and the weight distribution of the grasped object. This method was completely adaptive to objects and took effector kinematic configuration into consideration. However, this method used a complex and time-consuming analytical model, which seems excessive to transfer to our daily robotic application.

Further, for recognition, ImageNet use a dataset including more than 1.2 million high-resolution images categorized into 1,000 classes by a deep convolutional neural network. (Krizhevsky et al., 2012) showed that such a network was able to exceed classification records using only supervised learning. It should be remarked that modifying the network structure by removing convolutions degrades the classification results. That reference illustrated the fact that neural networks are adapted to recognition of certain patterns following intensive supervised learning, generalizing

nonlinear models with decision-making capabilities. This reference also explained that the structure of the neural connections is an important aspect of the training process.

(Lai et al., 2011) present a hierarchical tree of 51 object classes acquired in RGB-D with the Kinect V1. They showed that combined color and depth information contribute significantly to obtaining good recognition results. This is very relevant to our case. They generated their dataset using a turntable, removing the background and applying the Random Sample Consensus method to identify the surface of the turntable. Then, they employed the spin-images algorithm and the Scale-Invariant Feature Transform descriptor to obtain the image features. Following that, they applied the Linear Support Vector Machine, Gaussian Kernel, and Random Forest methods to classify the objects. They obtained 90.5% recognition in 10 s, which we deem too lengthy for our purposes.

(Tang et al., 2012) presents a system incorporating 3D acquisition, segmentation of objects on a table, meshing, creation of a histogram of features extracted using the Scale-Invariant Feature Transform, and a pose validation. They obtained recognition of the order of 90% for complex scenes. However, their series of object analyses requires 20 s per scene using a hexa core, 3.2 GHz i7 processor and 24 GB of RAM, which is not fast enough.

Finally, an object recognition algorithm by unsupervised learning is presented by (Lai et al., 2014), called Hierarchical Matching Pursuit for 3D Voxel Data. The authors' algorithm considers RGB-D data and treats both color and depth information with the Markov Random Fields method. They explain that use of a 2D image has merit, because capturing an object in front of its background may yield contextual information and that a 3D capture contains important spatial information. Globally, their object classification success rate is 95.3%, with scene features being extracted over 4 s. This delay is reasonable and the recognition rate is acceptable, which supports our belief that our analysis objective should be achievable and operational soon.

In summary, some of the above methods are promising and advanced, with 3D segmentation able to avoid occluding obstacles in complex scenes and provide shape reconstruction for improved prehension. Some methods develop innovative, complex, functional models for object prehension, and some have obtained good results for object recognition with up to 98.5% performance. Nevertheless, no method currently exists for scene analysis and object prehension that is both fast and simple.

5.4 Methodology

This section describes the system conceptual steps and the equipment required.

5.4.1 Acquisition of Scene Surfaces

The solution was developed using:

- The Kinect V2 vision system (Microsoft, USA).
- The JACO robotic arm (Kinova, Canada), intended for assisting people subject to musculoskeletal disorders such as muscular dystrophy and cerebral palsy.
- A set of 12 different objects (Figure 5.1) to be assimilated and recognized in various positions and orientations.

Figure 5.1 : The objects in the dataset.

The training dataset included the following objects: a tennis ball, a volleyball, a box of cookies, a juice box, an eyeglass case, a tissue box, a red box, a coffee cup, a blue container, a metal container, a cardboard box, and a yogurt container.

The active Kinect V2 camera was chosen to acquire the images from the 3D environment for the excellent quality afforded for its price. Further, this camera is practical for computer vision and 3D acquisition because its RGB-D acquisition is quick.

The Kinect V2 is composed of a high-resolution color camera (1920p×1080p at 30 frames per second) and an active infrared sensor for depth (512p×424p at 30 frames per second). The infrared sensor acquires depth as a time-of-flight camera using infrared illumination. The "flight time" is proportional to the distance between the camera and the target object. By measuring the distance to each point in the scene, a 3D representation of the scene is constructed. Finally, the Kinect enables to record a point cloud from a scene by using RGB color and depth D (RGB-D data).

The Kinect V2 has been used successfully in several research projects as shown in (Lachat et al. 2015) who concluded that Kinect V2 was promising considering the improved depth precision and general efficiency; nonetheless, the authors reported some acquisition limitations including errors internal to the system. Secondly, (Yang et al., 2015) presented depth measurement precision in the vision cone. Their results showed that the Kinect V2 precision is acceptable as long as acquisition is performed within a certain vision cone. However, the authors reported that reflective objects may cause acquisition problems. Similarly, dark black, light-absorbing materials reduce signal strength, which leads to increased illumination power requirements and brings about additional limitations.

In spite of these limitations, the Kinect V2 remains our choice because it is widely available, simple to use and very affordable. Indeed, the Software Development Kit (SDK) 2.0 (free software) includes multiple C++ and C# templates meant for quickly getting started using the Kinect V2 and its capabilities. Kinect Fusion is a 3D acquisition program presented by (Izadi et al., 2011) and is included in the SDK 2.0 ("Download Kinect for Windows SDK 2.0 from Official Microsoft Download Center"). The SDK is the program utilized for surface acquisition in our study. The SDK produces the point cloud from an acquirable environment in one of three formats directly: STL, OBJ, or PLY. Using the Kinect Fusion requires parameter tuning, such as setting the depth threshold (the distance between the Kinect V2 and the scene), which varies from 0.5 to 4.5 meters. The volumetric parameters also require tuning: the maximum point integration volume, the linear density of voxels, and the voxel resolution volume in XYZ. Starting from our real-time computing requirements, we performed some tests based on which we opted to record the surface data in PLY format. That format contains six essential data for a point in space: its three XYZ coordinates and the three elements of its RGB color triplet, which together represent the required RGB-D data.

Each surface acquired from a scene is recorded as a cloud of points for object segmentation and for cleansing of points not associated with those objects. Figure 5.2 illustrates a scene to be analyzed and will serve as our discussion reference.

Figure 5.2 : Reference scene in RGB color.

Subsequent to surface acquisition, the acquired data points are stored in a table by RGB color and XYZ spatial coordinates. Following the process detailed below, once segmentation is completed by the first part of the system, each independent object matrix is analyzed using a decision tree to determine where to grasp the object with the robot effector. It is assumed that the effector has two fingers, but it is possible to adapt the analysis to other effector designs. For each object in the scene, the decision tree returns a list describing the effector approach path, the coordinates of the center and the coordinates of the two prehension targets.

In this study, we considered a dataset composed of 12 objects to be recognized (illustrated in Figure 5.1). Accordingly, for each object class considered, the belonging of each object to each class was represented as a vector of 12 Boolean values composed of 0 or 1 that specify belonging in that class. For each of the classes that a given object belongs to, we acquired 15 different positions and orientations within a basic scene on table, without clutter. This totals 180 object scenes to be assimilated by the neural network and 12 object classes. After several tests varying number of hidden layers and number of neurons per layer, we obtained a 95% confusion matrix for object classification with a single hidden layer and 100 neurons.

5.4.2 Conceptual Process System

The principle of the system is presented in Figure 5.3 and is detailed in the following subsections. The methodology employed to develop our solution is explained in detail in this section, and includes three principal steps for conceptualization:

1. Object detection in a scene (transformation and segmentation) (Section 5.4.2.1).

2. Supervised learning by a neural network on a dataset (Section 5.4.2.2).

3. Scene analysis (prehension and recognition) (Section 5.4.2.3).

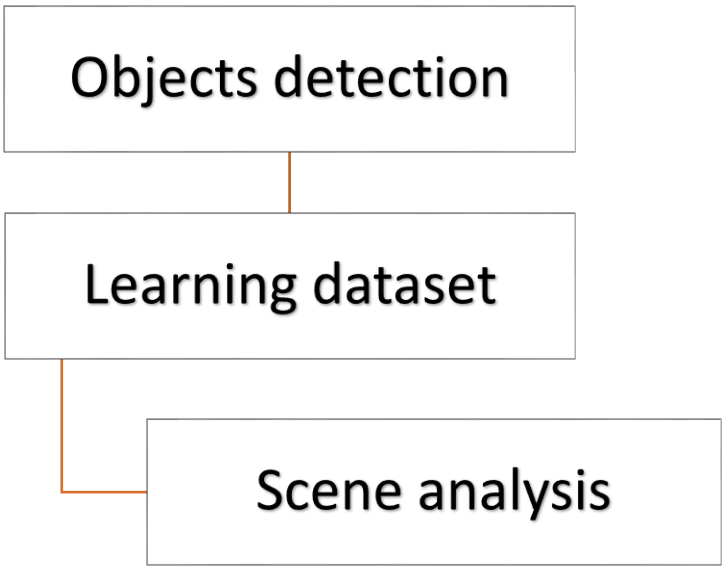

Figure 5.3: Conceptual process of the system.

5.4.2.1 Object Detection in a Scene (Transformation and Segmentation)

The algorithm for object detection in a scene is comprised of two steps:

Step A: Transformation of the point cloud.

Step B: 3D segmentation.

Step B.1: “Top-down probing” for object detection.

Step B.2: Elimination of undesirable points.

Step B.3: Counting of the objects in the scene.

Step A: Transformation of the point cloud

Once acquisition of the scene is complete and the PLY file has been saved, the program immediately reads the text file containing the table of six components of information per point. The program rotates the point cloud based on the angle from which the Kinect V2 views the scene. The angle of rotation is based on the angle at which the Kinect V2 is mechanically fixed, for the purpose of righting the scene vertical axis and simplifying the analysis. Figure 5.4 shows the surface acquired from the reference scene (Figure 5.2), before transformation. The number of points is very large and the center of the point cloud is at coordinates (0, 0, 0), causing negative coordinates and coordinates with decimals.

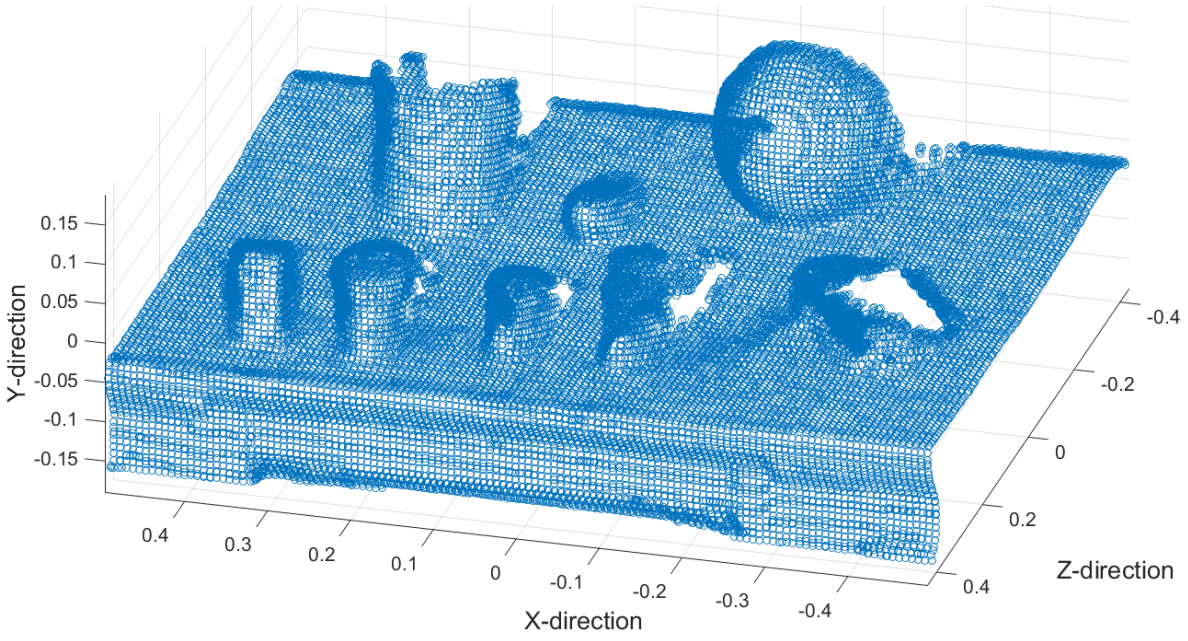

Figure 5.4 : Scene surface acquisition with the Kinect V2.

The algorithm transforms the scene to send a corner to the origin and to reduce the number of points to a specific sampling step (p). These changes are intended to reduce the computation time by reducing the number of points and to work only with whole, positive numbers.

Transforming the point cloud to an efficiently analyzable 3D matrix of the scene is comprised of three steps:

1) Obtaining the minimum ($posMin_{xyz}$) and maximum ($posMax_{xyz}$) in the three XYZ directions.

2) Computing the XYZ dimensions (dim_{xyz}).

$$dim_{xyz} = posMax_{xyz} - posMin_{xyz} \quad (1)$$

3) Computing the number of XYZ divisions ($nDiv_{xyz}$) of the scene in terms of the step (p) to minimize the number of conserved points and reduce computation time. Knowing these divisions, the 3D matrix of the 3D scene may be allocated.

$$nDiv_{xyz} = ((dim_{xyz})/p) + 1 \quad (2)$$

$$M3D_{dim} = [nDiv_x, nDiv_y, nDiv_z] \quad (3)$$

4) Determine the center of the 3D matrix ($CM3D_{xyz}$).

$$CM3D_{xyz} = nDiv_{xyz}/2 \quad (4)$$

5) Calculate the slopes in X, Y, and Z (m_{xyz}) and convert the positions of the points in the cloud (pc_{xyz}) to coordinates in the 3D matrix ($M3D_{xyz}$) of the scene. Here, the color information is not utilized. The function assigns value of 1 to points to be conserved and a value of 0 to empty spaces. Equations 5, 6, and 7 reflect the above principles.

$$m_{xyz} = nDiv_{xyz}/dim_{xyz} \quad (5)$$

$$M3D_{xyz} = m_{xyz} * pc_{xyz} + CM3D_{xyz} \quad (6)$$

$$M3D\left(M3D_x, M3D_y, M3D_z\right) = 1 \quad (7)$$

The 3D matrix of the scene obtained is represented in Figure 5.5, which is expressed in whole, positive Cartesian coordinates.

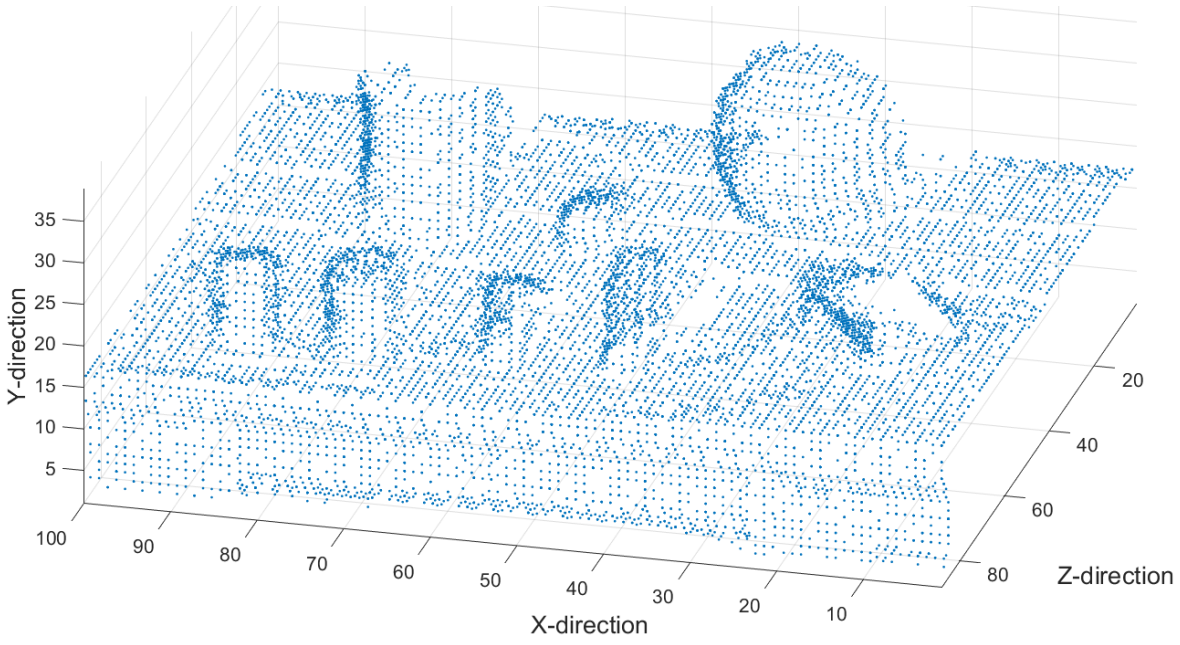

Figure 5.5 : 3D Matrix of the scene after transformation.

Step B: 3D segmentation

(Nguyen and Le, 2013) published a review of the various methods for 3D segmentation of point clouds and their relative pros and cons. It is reported that 3D segmentation in real time is not yet a complete success, and that overall, researchers globally use the following two approaches.

The first approach uses purely mathematical models and geometrical reasoning techniques such as region growth or model fitting, combined with robust estimators for adaptation to linear or nonlinear models data points. This approach is fast and yields good results for simple scenes. However, selecting the dimension of the model is difficult when noise sensitivity and object fitting are problematic in complex scenes.

The second approach extracts 3D features from point clouds using feature descriptors and makes use of machine learning techniques to assimilate the classes of different objects. The resulting model is used for data classification. According to the article, this second approach supersedes the first approach because noise and occlusions make identifying and adapting to the geometries of complex primitives very hard to implement using the first approach. Starting from the work of (Nguyen and Le, 2013), we directed our work towards 3D segmentation and extraction of features

from scenes to present them to a classifier, which would be based on a technique such as a neural network. In the following sections, we present our method for scene segmentation from transformed point clouds that were acquired with the Kinect V2.

Step B.1: "Top-down probing" for object detection

Using the data from the scene 3D matrix, we developed a solution inspired by mechanical probing. Mechanical probing is a measurement technique used in mechanical engineering to analyze surfaces and detect the dimensions of parts. Our approach enables us quickly to extract the dimensional features of surfaces using a virtual mechanical probe by considering the topmost points in the 3D matrix and identifying the tops of objects. This technique is intended to minimize computation time when determining how many objects are present in a scene and their locations. For this purpose, we established a table of elevations ("heights") ($tabH$) which records the coordinates of upper points. In the case that the virtual probe descends and does not detect any points, the algorithm records the value 1 for elevation Y (H) at coordinates XZ, which is the bottom of the scene 3D matrix. The algorithm then proceeds to the next XZ coordinates (see the principle illustrated in Figure 5.6).

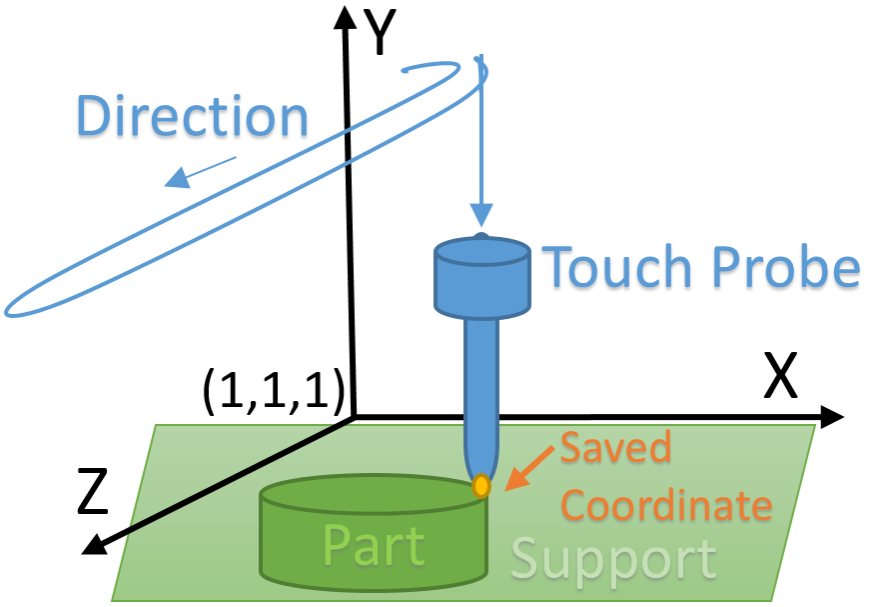

Figure 5.6: Virtual probe.

With the elevation Y (H) and their position XZ data recorded, the algorithm creates a table ($countH$) for counting heights, after which the system counts the number of points with the same elevation (H) and records them. To do this, the algorithm gives the counting table ($countH$) as many rows as there are elevations (H) available in the scene. Then, the algorithm goes through the table of elevations ($tabH$) and adds counter ($countH(H_{ligne})$) when the elevation (H) of a row in the table having elevation (H_{ligne}) is equal to a row of the counting table ($countH$).

$$CREATE\ countH(ligne_i) = (Hscene_i) \quad (8)$$

$$FOR\ (tabH = 1\ to\ max(H)) \quad (9)$$

$$IF\ tabH(ligne) = H,\ Then\ countH(H_{ligne}) = countH(H_{ligne}) + 1 \quad (10)$$

This allows identification of a surface susceptible of being a horizontal surface that supports objects, such as a table or a floor. The elevation (H) shared by the most points is deduced to be the elevation (H) of the table surface. Once (H) is determined, we eliminate all the points below(H). Clearly, the technique is fallible, but it is very simple and fast in most cases. The potential problem with this technique is that in the case that the elevation shared by the most points is not the table surface, the table is identified incorrectly. In the future, logical countermeasures could mitigate this problem.

Furthermore, during surface acquisition using the Kinect V2, undesirable points that do not correspond to real points in the scene are erroneously generated, but only in attempting to close the acquired surface. Figure 5.7 shows the meaningful points on the objects without the table, and undesirable points are circled in red.

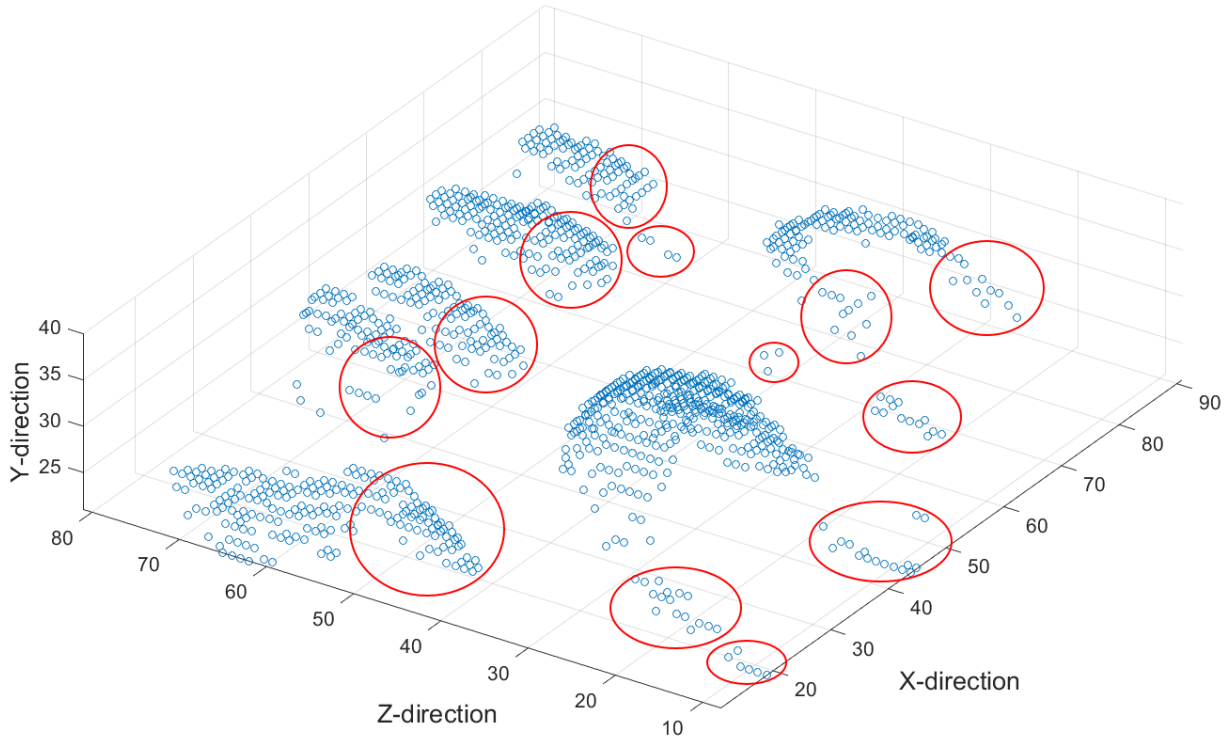

Figure 5.7: 3D matrix without the table where undesirable points are circled (in red).

Step B.2: Elimination of undesirable points

The algorithm for eliminating undesirable points follows the five following steps:

1) The Y-coordinates of the elevations table (H) are used to create a 2D matrix ($M2D(H_x, H_z)$) containing the elevations of the topmost points to analyze the topography suggested by these points (see Figure 5.8) by assigning the elevation values (H) to positions X (H_x) and Z (H_z).

$$M2D(H_x, H_z) = H \tag{11}$$

2) The algorithm uses the padarray (fct) computer vision functions to add contours to the 2D matrix $M2D(H_x, H_z)$, to obtain the augmented 2D matrix $M2D(H_x, H_z)_+$ and close holes, making the surface smoother and more uniform in elevation.

$$M2D(H_x, H_z)_+ = fct(M2D(H_x, H_z)) \tag{12}$$

3) The algorithm calculates the gradient along Z (∇_Z), determining the differences in the numerical values of elevation (H), and at each point averaging the three points behind with a 2D convolution ($conv$). The convolution is applied three times to obtain the needed value, that is, the average gradient along Z using three convolutions ($\mathrm{C}\nabla moy_z$).

$$\nabla_Z = gradient(M2D_+) \quad (13)$$

$$\mathrm{C}\nabla moy_z = conv(\nabla_Z) \quad (14)$$

4) In the event that ($C\nabla moy_z$) of a particular coordinate in the 2D elevation matrix ($M2D(H_x, H_z)$) exceeds a decision threshold (DT), then that coordinate is considered one of the undesirable points (pts) and is deleted. In this work, the threshold is set to 11, based on the mean gradients obtained following three convolutions.

$$DELETE\ pts\ \ IF\ (C\nabla moy_z \geq\ DT) \quad (15)$$

5) After identifying and eliminating all the undesirable points (pts), there remain only the coordinates corresponding to objects in the scene as seen from above. Figure 5.8 shows the result of eliminating the undesirable points (pts).

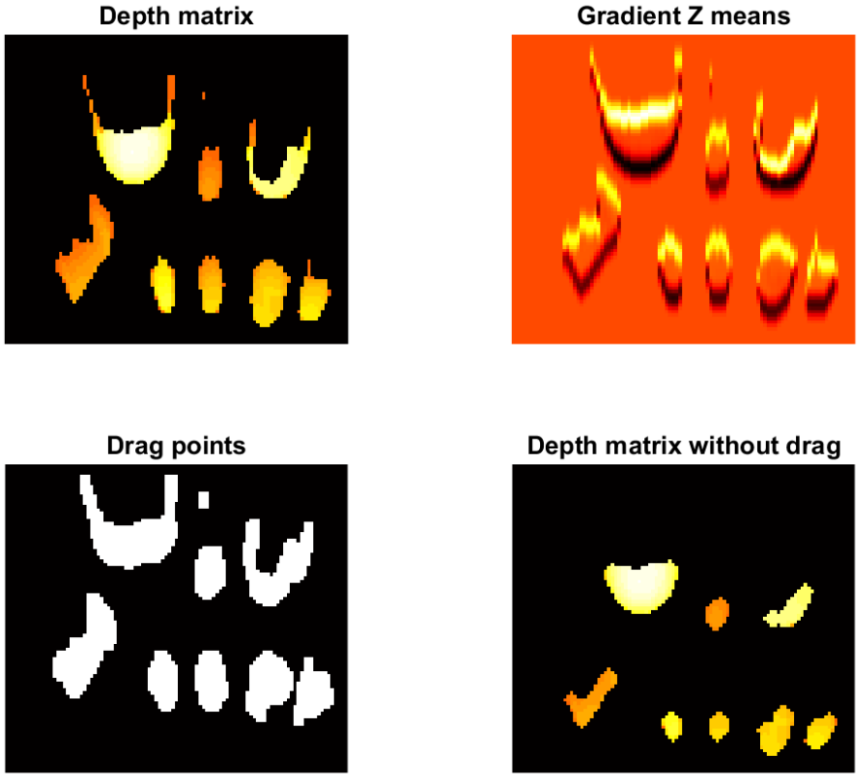

Figure 5.8 : The algorithm elimination of undesirable points.

Step B.3: Counting the objects in the scene

In this step, subsequently to 3D segmentation, the objects in the scene are counted. This step is crucial to the remainder of the process, because it enables determination of the number of treatments to be performed by the decision tree and neural network for prehension analysis. Each object in the scene is assigned a reference number to index the objects consistently during the remainder of the analysis.

To count the objects in the scene, we use the Sobel filter ($edge_{Sobel}$) ("Sobel operator") to identify the object contours ($cont$). Then, the contours ($cont$) are enhanced to improve clarity, dilating and extending the remaining objects (see Figure 5.9).

$$cont = edge_{Sobel}(M2D_{clean}) \quad (16)$$

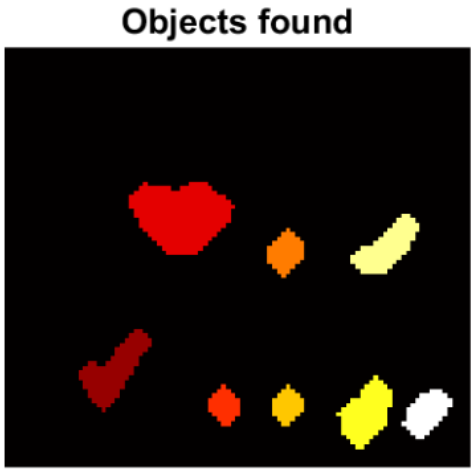

Figure 5.9 : Objets identified in the scene, which may be counted.

Using a Matlab function (regionprops), we count the objects in the scene, their dimensions, and their XZ centroids. After which, we consider the XYZ centroids of the objects to be located at their XZ centroids, at a height which is half the elevation Y at XZ.

5.4.2.2 Supervised Learning of a Dataset by a Neural Network

Training for object recognition was done using a neural network. We consider a supervised classification problem in which the program knows the objects' classes in advance and must minimize the error of the network output. This process is generally known as inductive learning. Thus, we have a set of manually labeled output vectors that the classifier employs to develop its nonlinear model for class identification by creating a decision rule. The decision rule is created by

optimizing for certain criteria of the learning algorithm, such as the number of epoch and the learning rate. (Lee et al., 1990), presented the surface texture as a feature to a neural network, which greatly helps class recognition. For neural network object recognition, we therefore employed a similar approach, using RGB-D texture, which is comprised of the colors and spatial positions of each of the points making up the objects. (Ciresan et al., 2011) explains that neural networks are a technique for adaptive classification suitable for complex problems like object recognition.

We utilized a conventional neural network called multi-perceptron by back-propagation with a single hidden layer to obtain a second order model and binary classification output, because this type of network is appropriate for classification problems (see the example in (Lee et al., 1990)). A neural network involves input values, time-dependent learning weights, hidden layers of neurons, a number of neurons per layer, and output classes. In our case, it is back-propagation of error that modifies the variable learning weights in the neural network. The system computes the error between the labeled binary output vector of classes and the neural network output vector. For cases with large numbers of classes, as with the 1,000 classes in (Krizhevsky et al., 2012), deeper networks (having additional hidden layers) become a sound means of increasing the order of complexity of the model and improve class identification.

The principle is simple: each numerical input value (X_i) is multiplied by a learning weight (W_i) that varies in time to minimize output classification error following back-propagation. Once multiplied (see Equation 17), the sum (Sum_i) is used as the input for an artificial neuron. (Sum_i) is passed to an activation function (Fa_i), which may be activated or not depending on the function used to interpret the data. In general, this activation function is a sigmoid in which Sum_i is used as an exponent as in Equation 18. Each activation brings about changes to the network and attempts to obtain labeled binary output vectors. Figure 5.10 shows a completely connected prototypical neural network in its simplest form. Note that each connection implies a learning weight (W_i).

$$Sum_i = \sum_{i=1}^{n} X_i * W_i \tag{17}$$

$$Fa_i = \frac{1}{1+e^{-Sum_i}} \tag{18}$$

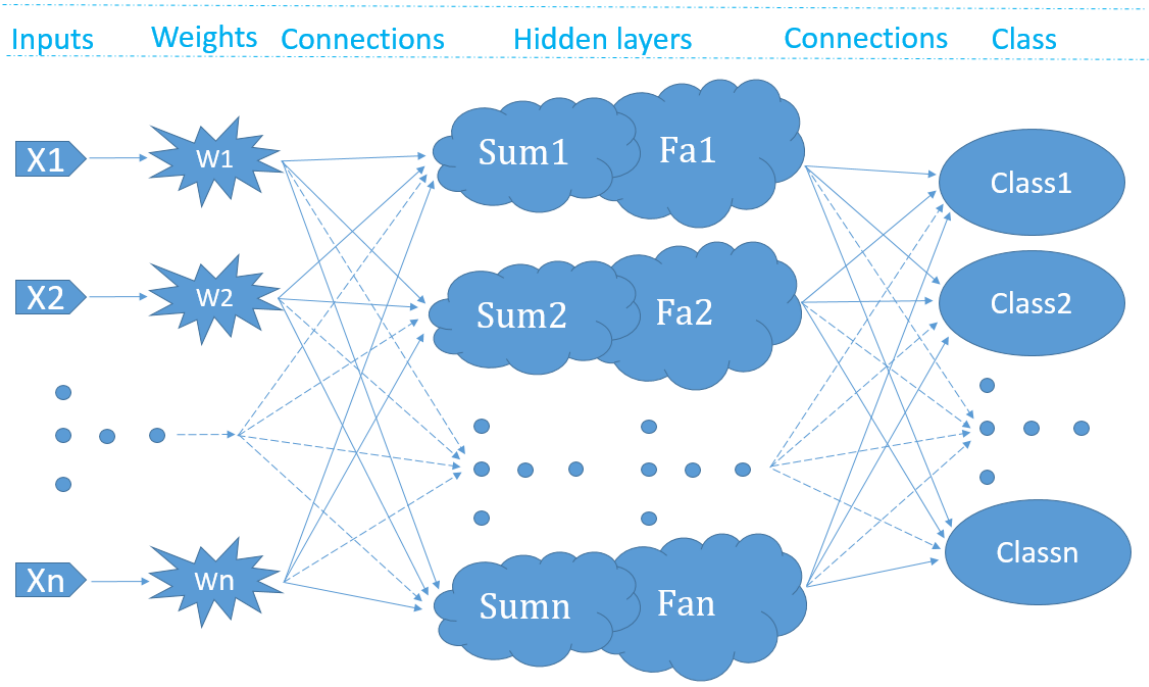

Figure 5.10 : Neural network.

For object recognition, we used two object features as network inputs: 1. The acquired surface expressing the geometric shape and possessing size and spatial position information pertaining to an object. 2. The surface texture, which possesses RGB color with a depth of 256 levels per color.

During the training stage, each vector input to the neural network comes from the 3D sub-matrix of an object as extracted from a scene by 3D segmentation. This 3D sub-matrix is transformed into a vector containing the RGB-D information pertaining to the segmented object. More precisely, Equation 19 yields a unique number describing the color composition (C_{num}) of a point in space:

$$C_{num} = 256^2 * R + 256 * G + B \quad (19)$$

where R, G, and B are the values obtained from the Kinect acquisition. Knowing the point coordinates, the number is assigned to the appropriate element of the vector.

5.4.2.3 Scene Analysis (Prehension and Recognition)

The analysis algorithm has two steps: prehension and recognition of objects.

Step C: Object prehension.

Step C.1: Separation of 3D sub-matrices after segmentation.

Step C.2: Utilization of the decision tree for approach and selection of prehension targets.

Step D: Object recognition.

Step D.1: Generation of the neural network input vectors comprising numerical color information of points based on their coordinates within an object.

Step D.2: Object classification by the neural network.

Step C: Object prehension

In this step, prehension targets for grasping are identified, i.e., it is decided where to grasp upon the envelope of a 3D object. This is done using a decision tree.

For object prehension, (Calli et al., 2011) presents an algorithm for object grasping in a multiple-geometry environment. Their method uses active vision and maximal object curvature information. The algorithm performs the effector approach using a control closed loop and a camera which feeds the Elliptic Fourier Descriptor analysis of the scene to the system. In other terms, the algorithm transforms the image to a 2D model, computes curvature to select prehension targets, and sends the corresponding commands to the servos. However, (Calli et al., 2011) does not indicate the execution time for prehension. They report that it is plausible to get real time by developing an implementation of the process that makes use of parallel programming. Although this mathematical method seems effective, we explored a dimensional method that is significantly simpler, to determine whether comparable effectiveness is achievable in a simpler manner. Accordingly, in the following step, the system segregates the detected objects into 3D sub-matrices and passes them to a dimensional decision tree to select prehension targets on the objects.

Step C.1: Separation of 3D sub-matrices after segmentation.

Once the XYZ object centroids are recorded after segmentation (step B), an algorithm identifies subsets of points corresponding to objects, and assigns them to 3D sub-matrices. This is done using XZ delimitation boxes and object heights (H).

During generation and segregation of the 3D sub-matrices of the objects, the algorithm adds the acquired RGB colors, giving 6 dimensions for each point of an object: (X, Y, Z, R, G, B). Then,

the centroids of the objects' 3D sub-matrices are recalculated to compensate for rounding errors incurred during segregation.

Step C.2: Utilization of the decision tree for approach and selection of prehension targets.

Once the separation of the 3D sub-matrices is complete, the algorithm employs the decision tree, described below, to generate an adequate approach path for the robot effector. The algorithm also selects spots on the objects to serve as practical targets for prehension. These operations are performed for each object in the scene.

- ***Condition 1:*** The height (H), along axis Y, of the object is compared to the object width and length along axes X and Z. In the case that height is greater than width and length, the robot employs an approach from the side; otherwise, the robot grasps the object from above.
- ***Condition 2:*** It must be determined which is greater of the width X and the length Z. In the case that the dimension in X is greater than in Z $(D(X) > D(Z))$, the robot positions the fingers of its effector parallel to the Z axis to grasp the object along its smallest dimension. Doing so facilitates prehension; alternatively, the robot effector may not be able to open wide enough to grasp the object. In the same manner, in the case that the dimension in X is inferior to that in Z, $(D(X) < D(Z))$, the robot positions its effector fingers parallel to the X axis, again so as to grasp the object along its smallest dimension. Figure 5.11 shows the decision tree for object prehension using two fingers.

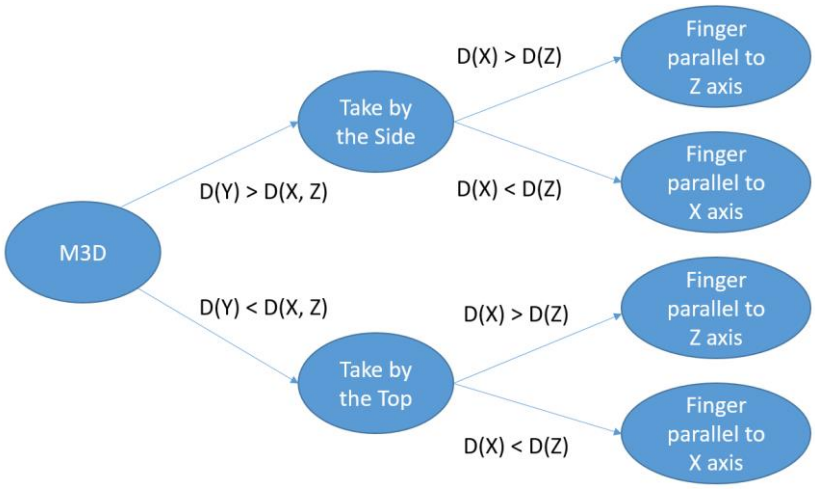

Figure 5.11 : The decision tree for object prehension using two fingers.

Once the fingers have been oriented, the algorithm uses the XYZ coordinates of the centroid of the object 3D sub-matrix and the dimensions of the object to grasp the object at the computed prehension targets. Naturally, in the case that the object smallest dimension is too large for the effector to grasp, the algorithm return indicates that prehension of the object is impossible.

Step D: Object recognition

Step C having encompassed determination and analysis of the prehension targets, step D broaches object classification (recognition) in terms of the assimilated training dataset. Classification is performed for the sake of identifying the object that the user wishes the robot to grasp, and consequently, to compute the prehension parameters with which to command the robotic assistance arm. Step D is divided into two substeps:

Step D.1: Generation of the neural network input vectors comprising numerical color information of points based on their coordinates within an object

This step consists of creating the input vectors intended for the neural network, which are composed of numerical values reflecting the color information and spatial coordinates of the points. After prehension analysis, the algorithm transforms this information ((X, Y, Z, R, B, G)) into vectors representing object features for input to the neural network. As explained in Section 5.4.2.2 on the methodology of the neural network supervised learning, those features describe a surface in terms of the distribution of its points, the size of the object, and the point colors.

Step D.2: Object classification by the neural network

Our system loads the weights learned by the neural network as in Section 5.4.2.2 (‘Supervised Learning of a Dataset by a Neural Network’), then iterates over the input vectors that describe the features of the scene objects. The neural network takes each vector as input and outputs the probability of each object belonging to each of the 12 object classes assimilated from the training dataset. Each object is assigned its most likely class. Our solution considers greatest probability lesser than 50% to be a case of inadequate recognition and that in such cases the object is not reliably recognized.

This classification procedure identifies the object requested by the user, enabling generation of the appropriate prehension parameters to command of the assistance arm. The parameters passed to

the robot are the effector approach path, the appropriate opening of the jaws, the object center and the two prehension targets upon the user-requested object.

5.5 Results

In this section, we present the results obtained with our system. These include object recognition training of a neural network from a dataset, results from the decision tree selection of prehension targets on the objects of a scene, and user operation of the system.

The reference scene used in this article contains 8 objects (see Figure 5.2). The system takes 4 s to detect objects (transformation and segmentation) and analyze the scene (prehension and recognition) using a 3.60 GHz Intel® Core™ i7-4790 CPU, 16 GB of RAM, and an NVIDIA GeForce® GTX 760 graphical processing unit. The four second interval implies a mean duration of 0.6 s per object (see Table 5.1).

In supervised learning, the confusion matrix qualifies the accuracy of the classification. The rows correspond to reference classes and each column corresponds to an evaluated object. The purpose of the confusion matrix is concisely to summarize the system classification results to identify correct recognition as well as false positives. As shown in Figure 5.12, the confusion matrix achieves 95% performance, which leads us to believe that the system recognition algorithm will perform well in environments similar to the training environment (in terms of positions, orientations, and distances).

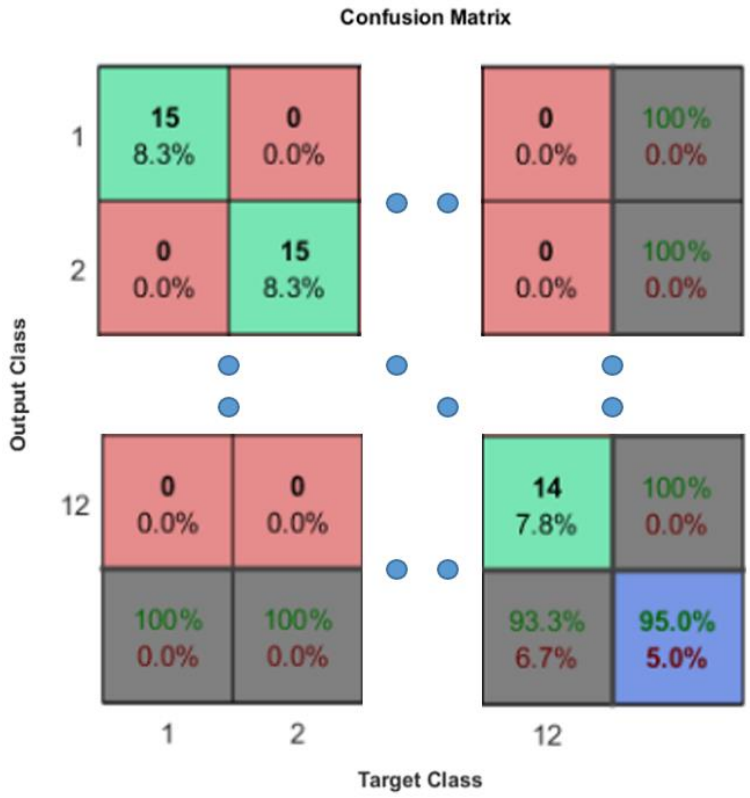

Figure 5.12 : The confusion matrix of our neural network.

As in Figure 5.13, following object detection (transformation and segmentation) and segregation of the objects' sub-matrices, the decision tree selects, for each object in a scene, two coordinates to serve as prehension targets.

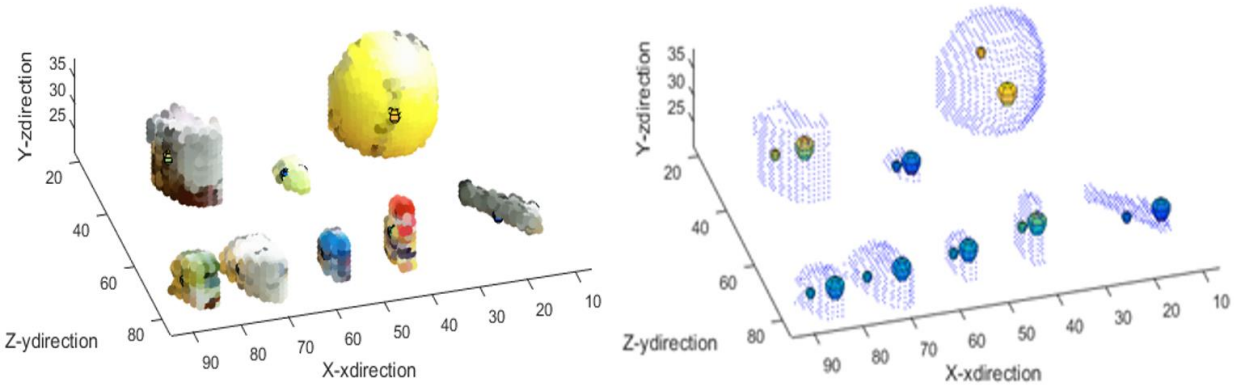

Figure 5.13 : (Left) Prehension scene, in color. (Right) Prehension targets on each object.

The information gathered from the scene analysis algorithm is shown in Figure 5.14: the number of objects in the scene, the object classes (names) of maximal recognition probability, the suggested path for effector approach, the appropriate opening of the effector jaws, the coordinates of the object center, and the two coordinates selected as prehension targets.

Scene analysis

The scene contains 8 objects

Scene description

…

The third object is a: Yogurt, with 90 % confidence.

Grasp the object by the: side.

Open your effector to: 3 unit.

Center :	X = 43	Y = 26	Z = 72
First place :	X = 41	Y = 26	Z = 72
Second place :	X = 44	Y = 26	Z = 72

…

Figure 5.14 : The scene description output by the scene analysis algorithm.

Table 5.1 illustrates the 83% global performance obtained, based on 10 different scenes in a controlled environment. No doubt, that success rate can be improved upon by employing a larger set of class data for object recognition training. As well, the same table lists the required Analysis Times, which are the times taken to analyze 10 scenes, and the Process Times, which are, for each scene, the sum of the time taken to load the scene and the time taken to analyze it. The difference between Process Time and Analysis Time is the time taken to load a scene, and averages 0.7 s for 7 MB scenes in PLY format. Furthermore, it is seen that the mean time required to analyze a scene for one object is 0.6 s.

Table 5.1: Recognition success rates and analysis times for ten scenes.

Scene	Objects Count (Qty)	Success Count (Qty)	Success Rate (%)	Process Time (s)	Analysis Time (s)	Time per Object (s)
1	4	4	100	2.6	1.9	0.5
2	2	2	100	2.8	2.1	1.1
3	3	2	67	2.3	1.6	0.5
4	4	3	75	3.3	2.7	0.7
5	2	2	100	2.3	1.6	0.8
6	3	2	67	3.4	2.7	0.9
7	6	5	83	3.2	2.5	0.4
8	6	5	83	2.7	2.0	0.3
9	6	6	100	3.0	2.3	0.4
10	7	4	57	3.8	3.1	0.4
Mean	4.3	3.5	**83**	2.9	2.3	**0.6**

Finally, user operation was tested using the JACO arm, for a total mean time of 15 s required to analyze and grasp a requested object. Note that duration could be greatly diminished by increasing the speed at which the robot moves, which we limited to 50 mm/s in compliance with safety standards.

To begin, the user positions himself before a scene containing the object of interest. The user activates the system, launching the Kinect V2 surface acquisition. The data acquired is recorded in a text file which the system reads. Scene analysis yields the object count, the effector approach

paths, the appropriate effector opening, the object centers, the prehension targets upon the objects, and the object classes (names). Subsequently, the user requests that an object be grasped by means of some interface and the robot then moves its arm and orients its effector in accordance with the coordinates output by the analysis. Figure 5.15 shows the process with user operation and Figure 5.16 shows the final results for real object prehension utilizing the JACO arm effector ("Kinova Robotics web site").

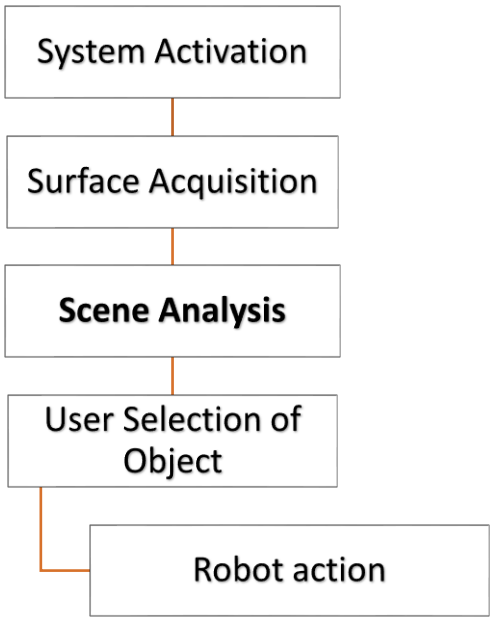

Figure 5.15 : Process of user operation.

Figure 5.16 : Real prehension scene following robot action.

5.6 Discussion

The developed system demonstrated that it is possible to analyze an object in a scene in about 0.6 s and prehend an object in 15 s. Naturally, the global computation time depends on the complexity of the scene. Indeed, computation time increases with scene complexity, as in Table 5.1. Nevertheless, the mean computation time our system requires is less than the time taken by traditional object detection methods (transformation, segmentation), which to the best of our knowledge require at least 4 s (Lai et al., 2014). The improvement in estimated time required for object detection, in comparison with the classical approaches common in the literature such as Randsac and Spin-images (Mian et al., 2005), is the result of the eminent efficiency of our top-down probing method. Naturally, the traditional methods are robust; however, they are slow to converge. For now, those methods have greater flexibility in the context of complex scenes involving occlusions and superpositions, but only the use of more powerful computers can garner quicker convergence from them. Our 3D segmentation algorithm is simple and fast and can be improved to deal with occlusions and superpositions by adapting other existing techniques (see Limitations and Prospects, below). Computation time depends crucially upon scene complexity, the choice of system for scene acquisition and recording, hardware computing power, and the algorithms employed.

Nevertheless, the use of a dimensional analysis for prehension and a pre-trained neural network for recognition enables fast object detection and analysis in real-time.

Our decision tree for object prehension is able to select a simple approach path for the robot effector, following four directions. This method is less robust than that of (Lippiello et al., 2013), which is able to ply a mathematical model of object envelopes to adapt in a complex manner to object geometry and effector model. Nonetheless, our system dimensional decision algorithm performs very well, simply, and very rapidly.

As shown in Table 5.1, the system recognition performance averages 83%, which is lower than the listed values found in literature. The literature values go up to 98.5% (Rusu et al., 2010); still, our method has broader applications and is based on more than solely using neural networks.

5.7 Limitations and Prospects

As well, 3D segmentation could be made more robust with respect to object superposition and adjacency, and the presence of walls and ceilings could also be considered. This could be done by combining the top-down virtual probing method with lateral probing equivalents, virtually probing from the front and the sides. Segmentation could also be improved by exploiting the color information when detecting and separating objects. Additionally, segmentation could be interrupted upon recognition of the user-requested object, saving time.

To improve our system prehension abilities, which are already quick and simple, the decision tree employed could be augmented using real-time control loop concepts. Additional branches could consider alternative grasping options and approach paths. Real-time obstacle avoidance during robot motion could also be implemented.

Object recognition using deep convolutional neural networks may be worth developing provided large training datasets are available, for the sake of including more object classes. To develop such a system, an algorithm will have to be designed that will automatically incorporate a large number of object class datasets. This would be done to test recognition in terms of color and surface features because manual tests are too time-consuming. Indeed, to obtain good supervised learning performance, the availability of a broad set of examples is paramount; the system ability to generalize the data tendencies hinges upon it. In the literature, the number of data may reach more than one million.

Despite the limitations and prospects discussed above, our results show that the system developed in the present study is already suitable for numerous everyday applications, such as in the computer vision employed by robotic aids for people affected by musculoskeletal disorders or loss of autonomy.

5.8 Conclusion

This study showed that a fast scene analysis using vision and artificial intelligence for object grasping by an assistive robot in cooperation with a human user may be performed with effective promptness. Indeed, the system requires, on average, 0.6 s to analyze an object in a scene. Using a robotic arm, the system may grasp a requested object in 15 s. The system object recognition performance averages 83% and is able to use a simple decision tree to select an approach path for

the robot effector to follow towards a requested object. This system, in combination with an assistance robot, has great potential for providing users with improved autonomy and independence, and encouraging sustained usage of their technical aid.

5.9 References

A. Krizhevsky, I. Sutskever, G. E. Hinton, 2012. ImageNet Classification with Deep Convolutional Neural Networks. Adv. Neural Inf. Process. Syst. 25 Curran Associates, Inc., PP.1097–1105.

Anguelov, D., Taskar, B., Chatalbashev, V., Koller, D., Gupta, D., Heitz, G., Ng, A., 2005. Discriminative Learning of Markov Random Fields for Segmentation of 3D Scan Data. IEEE, pp. 169–176. doi:10.1109/CVPR.2005.133

Calli, B., Wisse, M., Jonker, P., 2011. Grasping of unknown objects via curvature maximization using active vision. IEEE, pp. 995–1001. doi:10.1109/IROS.2011.6094686

Cruz, L., Lucio, D., Velho, L., 2012. Kinect and RGBD Images: Challenges and Applications. IEEE, pp. 36–49. doi:10.1109/SIBGRAPI-T.2012.13

Dan C. Ciresan, Ueli Meier, Jonathan Masci,, Luca M. Gambardella and Jurgen Schmidhuber, 2011. High-Performance Neural Networks for Visual Object Classification.

Download Kinect for Windows SDK 2.0 from Official Microsoft Download Center [WWW Document], n.d. URL https://www.microsoft.com/en-us/download/details.aspx?id=44561 (accessed 4.27.16).

E. Lachat, H. Macher, M.-A. Mittet, T. Landes, P. Grussenmeyer, n.d. FIRST EXPERIENCES WITH KINECT V2 SENSOR FOR CLOSE RANGE 3D MODELLING. ISPRS - Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. Vol. XL-5W4 2015 Pp93-100.

Jie Tang, Miller, S., Singh, A., Abbeel, P., 2012. A textured object recognition pipeline for color and depth image data. IEEE, pp. 3467–3474. doi:10.1109/ICRA.2012.6224891

Johnson, A.E., Hebert, M., 1999. Using spin images for efficient object recognition in cluttered 3D scenes. IEEE Trans. Pattern Anal. Mach. Intell. 21, 433–449. doi:10.1109/34.765655

Kinova Robotics web site [WWW Document], n.d. URL http://www.kinovarobotics.com/fr/ (accessed 4.27.16).

Lai, K., Bo, L., Fox, D., 2014. Unsupervised feature learning for 3D scene labeling. IEEE, pp. 3050–3057. doi:10.1109/ICRA.2014.6907298

Lai, K., Bo, L., Ren, X., Fox, D., 2011. A large-scale hierarchical multi-view RGB-D object dataset. IEEE, pp. 1817–1824. doi:10.1109/ICRA.2011.5980382

Lee, J., Weger, R.C., Sengupta, S.K., Welch, R.M., 1990. A neural network approach to cloud classification. IEEE Trans. Geosci. Remote Sens. 28, 846–855. doi:10.1109/36.58972

Lippiello, V., Ruggiero, F., Siciliano, B., Villani, L., 2013. Visual Grasp Planning for Unknown Objects Using a Multifingered Robotic Hand. IEEEASME Trans. Mechatron. 18, 1050–1059. doi:10.1109/TMECH.2012.2195500

Maheu, V., Frappier, J., Archambault, P.S., Routhier, F., 2011. Evaluation of the JACO robotic arm: Clinico-economic study for powered wheelchair users with upper-extremity disabilities. IEEE, pp. 1–5. doi:10.1109/ICORR.2011.5975397

Mian, A.S., Bennamoun, M., Owens, R., 2006. Three-Dimensional Model-Based Object Recognition and Segmentation in Cluttered Scenes. IEEE Trans. Pattern Anal. Mach. Intell. 28, 1584–1601. doi:10.1109/TPAMI.2006.213

Mian, A.S., Bennamoun, M., Owens, R.A., 2005. AUTOMATIC CORRESPONDENCE FOR 3D MODELING: AN EXTENSIVE REVIEW. Int. J. Shape Model. 11, 253–291. doi:10.1142/S0218654305000797

Nguyen, A., Le, B., 2013. 3D point cloud segmentation: A survey. IEEE, pp. 225–230. doi:10.1109/RAM.2013.6758588

Rusu, R.B., Blodow, N., Marton, Z.C., Beetz, M., 2009. Close-range scene segmentation and reconstruction of 3D point cloud maps for mobile manipulation in domestic environments. IEEE, pp. 1–6. doi:10.1109/IROS.2009.5354683

Rusu, R.B., Bradski, G., Thibaux, R., Hsu, J., 2010. Fast 3D recognition and pose using the Viewpoint Feature Histogram. IEEE, pp. 2155–2162. doi:10.1109/IROS.2010.5651280

S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, A. Fitzgibbon, 2011. KinectFusion: Realtime 3D Reconstruction and Interaction Using a Moving Depth Camera*. UIST'11 Oct. 16–19 2011 St. Barbara CA USA.

Sobel operator - Wikipedia, the free encyclopedia [WWW Document], n.d. URL https://en.wikipedia.org/wiki/Sobel_operator (accessed 6.12.16).

Yang, L., Zhang, L., Dong, H., Alelaiwi, A., Saddik, A.E., 2015. Evaluating and Improving the Depth Accuracy of Kinect for Windows v2. IEEE Sens. J. 15, 4275–4285. doi:10.1109/JSEN.2015.2416651

Zug, S., Penzlin, F., Dietrich, A., Nguyen, T.T., Albert, S., 2012. Are laser scanners replaceable by Kinect sensors in robotic applications? IEEE, pp. 144–149. doi:10.1109/ROSE.2012.6402619

CHAPITRE 6 DISCUSSION GÉNÉRALE

La solution (système) développée dans le cadre de cette maîtrise représente un premier jet bien fonctionnel, et il y a plusieurs améliorations qui devront y être apportés. Un ajout possible durant une acquisition serait que le système puisse analyser une scène complexe contenant plusieurs étages de soutien d'objets. Il faudrait donc que le système soit capable de diviser l'acquisition en sous-scènes, de faire le traitement en parallèle pour chaque sous-scène et de ramener le tout en une analyse unique de toute l'acquisition.

Également, si la Kinect V2 change d'angle de support en temps réel, il faudrait ajouter une automatisation de la rotation du nuage de points pour ramener la scène à l'horizontale. La procédure consisterait à ajouter un capteur de position (potentiomètre et/ou gyroscope) pour connaitre la position de la caméra par rapport à l'horizontale et modifier en temps réel la rotation dans l'algorithme.

De plus, la détection d'objets pourrait devenir plus robuste afin de détecter par exemple des objets superposés, collés entre eux, des éléments tels que les murs et le plafond, afin de généraliser l'acquisition en utilisant une combinaison de palpage par le haut, par en avant et par le côté. L'utilisation de la couleur comme caractéristique pourrait être utilisée pour séparer et détecter les objets. De plus, la détection pourrait être interrompue si un objet désiré par un utilisateur est reconnu avant que la segmentation ou l'analyse soit complétée pour permettre de sauver du temps de calcul.

Pour améliorer la préhension de notre système avec l'arbre de décision qui est déjà simple et efficace, on pourrait combiner des concepts de boucle d'asservissement en temps réel. Nous pouvons également ajouter des branches supplémentaires qui prendraient en considération plus d'options de prise d'objets et de phase d'approche. Nous pourrions aussi ajouter l'évitement d'obstacles lors d'une trajectoire entreprise par l'effecteur du robot en temps réel pour la préhension d'un objet désiré.

Dans (Ciresan et al. (2011)) [34], leur réseau de neurones à convolution est testé sur trois ensembles d'images différentes soit NORB, CIFAR10 et MNIST. Les erreurs de tests sont excellentes, soit 2.4 %, 1,0 % et 0,5 % pour les trois ensembles respectivement. Leur structure utilise un entrainement par descente du gradient, une architecture de rétro-propagation de l'erreur et des connexions aléatoires qui reflètent davantage la biologie du cerveau humain. Cette structure semble

plus efficace que les réseaux de neurones à convolution complètement connectés à chaque neurone dans chaque couche cachée. Nous pourrions alors envisager de concevoir notre propre architecture de réseau de neurones plutôt que de prendre un réseau totalement connecté dans le futur.

Également, (Cao et al. (2015)) [35] développe une nouvelle approche pour effectuer la classification d'images avec un réseau de neurones à convolution capturant la sémantique de haut niveau en projetant l'information sur une représentation schématique utile pour la reconnaissance et la localisation. Leur concept est inspiré du processus de vision humaine qui est davantage porté sur les rétroactions que sur les connexions de rétro-propagation grandement utilisées dans le domaine de la vision par ordinateur. La possibilité d'ajouter des liens logiques entre un environnement et les objets, c.-à-d. de la sémantique, permettrait une reconnaissance plus flexible et mieux adaptée.

La reconnaissance d'objets avec un réseau de neurones à convolution profonde peut être intéressante à développer dans le futur à condition d'avoir un grand ensemble de données afin d'ajouter plusieurs autres classes. Pour ce faire, il faudrait développer un algorithme qui permettrait d'ajouter automatiquement une grande quantité d'ensembles de données de classes d'objets pour tester la reconnaissance avec les caractéristiques utilisées soit la surface et la couleur, car le faire manuellement prend beaucoup trop de temps. En effet, pour obtenir une bonne performance par apprentissage supervisé, il faut absolument avoir un grand nombre d'exemples à lui transmettre pour qu'il généralise mieux la tendance des données. Dans la littérature, le nombre de données peut aller jusqu'à plus d'un million.

Après l'énoncé de ces limitations et de ces perspectives, rappelons que le système développé dans cette étude peut déjà être utilisé dans de nombreuses applications du quotidien actuellement, tel que l'asservissement à la vision de robots d'assistance pour les personnes atteintes de troubles musculo-squelettiques ou de perte d'autonomie comme présenté dans les résultats du Chapitre 5.

Le système développé respecte plusieurs objectifs énoncés dans le Chapitre 3 soit d'avoir une technique de détection d'objets plus rapide et simple, une technique d'apprentissage supervisé par un réseau de neurones, et une technique de préhension d'objets simple, rapide et efficace. L'objectif qui reste à mieux développer est une technique de reconnaissance d'objets aussi performante que la littérature, mais nous avons pris connaissance des outils et des moyens pour atteindre cet objectif dans le futur. En somme, l'hypothèse générale est respectée car notre temps de calcul pour l'analyse

d’une scène est au moins deux fois plus petit que le temps d’analyse de scènes dans la littérature selon nos connaissances.

En effet, le système développé peut être embarqué sur une chaise adaptée pour les utilisateurs et comprendra le robot d’assistance Jaco, une manette de commande sur l’appui-bras, une interface-écran, une batterie de puissance pour une économie prolongée, un ordinateur haute performance pour le traitement de calcul de données en temps réel et une Kinect V2 positionnée à environ 150 cm avec un angle de 30 degrés par rapport à l’horizontale pour l’acquisition surfacique de l’environnement d’intérêt.

De plus, ce système intelligent peut être utilisé dans de nombreuses autres applications que celles énoncées dans ce mémoire. Pour en nommer qu’une seule, la réalisation d’un robot humanoïde qui aiderait toute personne au quotidien à la maison ou pour l’exécution d’un travail à risque pour l’homme. Ce robot pourrait permettre de répondre à plusieurs besoins afin d’avoir un potentiel diversifié. C’est dans ce sens que les systèmes intelligents robotiques doivent évoluer et se fusionner avec d’autres programmes pour concevoir un système globalement performant et pratique pour l’homme.

CHAPITRE 7 CONCLUSION ET RECOMMANDATIONS

En définitive, ce mémoire a démontré que l'analyse de scène rapide utilisant la vision et l'intelligence artificielle pour la préhension d'objets par un robot d'assistance en coopération avec un humain peut être réalisée en un temps efficace. En effet, le système prend en moyenne 0.6 seconde pour la détection et l'analyse d'un objet dans une scène. De plus, avec un bras robotique, il peut prendre un objet désiré en moins de 15 secondes. Le système a une performance moyenne de 83 % pour la reconnaissance d'objets et est capable de choisir une phase d'approche simple de l'effecteur du robot vers un objet désiré en utilisant un arbre de décision. Ce système combiné à un robot d'assistance a un grand potentiel afin de permettre aux utilisateurs d'être plus autonomes, indépendants et de mieux adhérer à l'utilisation de leur aide technique.

Une recommandation serait de développer une nouvelle approche pour augmenter les facultés de la reconnaissance. Les techniques actuellement utilisées en intelligence artificielle sont des méthodes d'apprentissage par accumulation intense du nombre d'exemples à assimiler en un court laps de temps comme les réseaux de neurones ou autres techniques d'apprentissage supervisé. Par contre, est-ce qu'un humain a besoin de plusieurs exemples pour reconnaitre un objet et savoir son utilité? La réponse est définitivement non. Il faudrait développer des programmes qui ne sont pas uniquement basés sur des techniques calculatoires et statistiques qui cherchent uniquement à reproduire par cœur un savoir comme le présentent de nombreux ouvrages en intelligence artificielle.

Il faudrait plutôt se remettre en question et se questionner sur ce qui nous pousse à nous surpasser, car l'apprentissage est un fil conducteur à travers nos expériences de vie et nos buts recherchés. Apprendre tout en un instant ne permet pas à un agent dit intelligent d'avoir la possibilité de faire ses propres réflexions et ses propres choix. Comprendre et faire des liens logiques entre différentes entités comme savoir ce qu'est un objet, à quoi il ressemble, à quoi il sert et comment les détecter dans l'environnement sans pour autant fournir des lignes de code de transformation et un grand ensemble de données est le prochain défi. Un agent ou programme soumis aux entrées et sorties que nous lui donnons et rien de plus ne peut certainement pas comprendre réellement.

La voie suggérée pour le développement d'un système vraiment intelligent est qu'il puisse apprendre en temps réel durant ses expériences de vie et non de manière précalculée. Le but ultime serait de transmettre la faculté d'avoir une curiosité et une soif d'apprentissage pour obtenir des

connaissances afin de réaliser ses buts. Dans le cas du projet de recherche, le but est de reconnaitre le plus d'objets possible afin de mieux assister les utilisateurs. La combinaison d'algorithmes génétiques avec des réseaux de neurones est peut-être une voie vers la solution, la recherche se poursuit!

BIBLIOGRAPHIE

[1] « Kinova Robotics web site ». [En ligne]. Disponible à: http://www.kinovarobotics.com/fr/. [Consulté le: 27-avr-2016].

[2] « JACO^{TM}, The New Robotic Manipulator Arm by Kinova - RobotShop Blog ». [En ligne]. Disponible à: http://www.robotshop.com/blog/en/jaco-the-new-robotic-manipulator-arm-by-kinova-932. [Consulté le: 27-avr-2016].

[3] V. Maheu, J. Frappier, P. S. Archambault, et F. Routhier, « Evaluation of the JACO robotic arm: Clinico-economic study for powered wheelchair users with upper-extremity disabilities », 2011, p. 1-5.

[4] D. Forsyth et J. Ponce, *Computer vision: a modern approach*, 2nd ed. Boston: Pearson, 2012.

[5] S. J. Russell, P. Norvig, et F. Popineau, *Intelligence artificielle*. Paris: Pearson education, 2010.

[6] A. de Medeiros Brito, A. D. Doria Neto, J. Dantas de Melo, et L. M. Garcia Goncalves, « An Adaptive Learning Approach for 3-D Surface Reconstruction From Point Clouds », *IEEE Trans. Neural Netw.*, vol. 19, n^{o} 6, p. 1130-1140, juin 2008.

[7] F. Zhang, L. Shi, Z. Xu, et Z. Hu, « A 3D Reconstruction System of Indoor Scenes with Rotating Platform », 2008, p. 554-558.

[8] « programming-with-kinect-v2-7-638.jpg (Image JPEG, 638 × 359 pixels) ». [En ligne]. Disponible à: http://image.slidesharecdn.com/programmingwithkinectv2-150603164425-lva1-app6892/95/programming-with-kinect-v2-7-638.jpg?cb=1433349971. [Consulté le: 14-mai-2016].

[9] A. Anwer, A. Baig, et R. Nawaz, « Calculating real world object dimensions from Kinect RGB-D image using dynamic resolution », 2015, p. 198-203.

[10] Yan Cui, S. Schuon, S. Thrun, D. Stricker, et C. Theobalt, « Algorithms for 3D Shape Scanning with a Depth Camera », *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, n^{o} 5, p. 1039-1050, mai 2013.

[11] E. Lachat, H. Macher, M.-A. Mittet, T. Landes, P. Grussenmeyer, « FIRST EXPERIENCES WITH KINECT V2 SENSOR FOR CLOSE RANGE 3D MODELLING », *ISPRS - Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. Vol. XL-5W4 2015 Pp93-100*.

[12] L. Yang, L. Zhang, H. Dong, A. Alelaiwi, et A. E. Saddik, « Evaluating and Improving the Depth Accuracy of Kinect for Windows v2 », *IEEE Sens. J.*, vol. 15, nº 8, p. 4275-4285, août 2015.

[13] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, et A. Davison, A. Fitzgibbon, « KinectFusion: Realtime 3D Reconstruction and Interaction Using a Moving Depth Camera* », *UIST'11 Oct. 16–19 2011 St. Barbara CA USA*, oct. 2011.

[14] « Download Kinect for Windows SDK 2.0 from Official Microsoft Download Center ». [En ligne]. Disponible à: https://www.microsoft.com/en-us/download/details.aspx?id=44561. [Consulté le: 27-avr-2016].

[15] D. Anguelov, B. Taskar, V. Chatalbashev, D. Koller, D. Gupta, G. Heitz, et A. Ng, « Discriminative Learning of Markov Random Fields for Segmentation of 3D Scan Data », 2005, vol. 2, p. 169-176.

[16] A. S. Mian, M. Bennamoun, et R. A. Owens, « AUTOMATIC CORRESPONDENCE FOR 3D MODELING: AN EXTENSIVE REVIEW », *Int. J. Shape Model.*, vol. 11, nº 2, p. 253-291, déc. 2005.

[17] R. B. Rusu, N. Blodow, Z. C. Marton, et M. Beetz, « Close-range scene segmentation and reconstruction of 3D point cloud maps for mobile manipulation in domestic environments », 2009, p. 1-6.

[18] A. Nguyen et B. Le, « 3D point cloud segmentation: A survey », 2013, p. 225-230.

[19] B. Calli, M. Wisse, et P. Jonker, « Grasping of unknown objects via curvature maximization using active vision », 2011, p. 995-1001.

[20] V. Lippiello, F. Ruggiero, B. Siciliano, et L. Villani, « Visual Grasp Planning for Unknown Objects Using a Multifingered Robotic Hand », *IEEEASME Trans. Mechatron.*, vol. 18, nº 3, p. 1050-1059, juin 2013.

[21] J. Lee, R. C. Weger, S. K. Sengupta, et R. M. Welch, « A neural network approach to cloud classification », *IEEE Trans. Geosci. Remote Sens.*, vol. 28, nº 5, p. 846-855, sept. 1990.

[22] A. Krizhevsky, I. Sutskever, G. E. Hinton, « ImageNet Classification with Deep Convolutional Neural Networks », *Adv. Neural Inf. Process. Syst. 25*, p. Curran Associates, Inc., PP.1097–1105, 2012.

[23] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, et R. M. Summers, « Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning », *IEEE Trans. Med. Imaging*, p. 1-1, 2016.

[24] A. E. Johnson et M. Hebert, « Using spin images for efficient object recognition in cluttered 3D scenes », *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 21, n° 5, p. 433-449, mai 1999.

[25] D. Huber, A. Kapuria, R. Donamukkala, et M. Hebert, « Parts-based 3D object classification », 2004, vol. 2, p. 82-89.

[26] A. S. Mian, M. Bennamoun, et R. Owens, « Three-Dimensional Model-Based Object Recognition and Segmentation in Cluttered Scenes », *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, n° 10, p. 1584-1601, oct. 2006.

[27] R. B. Rusu, G. Bradski, R. Thibaux, et J. Hsu, « Fast 3D recognition and pose using the Viewpoint Feature Histogram », 2010, p. 2155-2162.

[28] K. Lai, L. Bo, X. Ren, et D. Fox, « A large-scale hierarchical multi-view RGB-D object dataset », 2011, p. 1817-1824.

[29] Jie Tang, S. Miller, A. Singh, et P. Abbeel, « A textured object recognition pipeline for color and depth image data », 2012, p. 3467-3474.

[30] K. Lai, L. Bo, et D. Fox, « Unsupervised feature learning for 3D scene labeling », 2014, p. 3050-3057.

[31] S. Zug, F. Penzlin, A. Dietrich, T. T. Nguyen, et S. Albert, « Are laser scanners replaceable by Kinect sensors in robotic applications? », 2012, p. 144-149.

[32] L. Cruz, D. Lucio, et L. Velho, « Kinect and RGBD Images: Challenges and Applications », 2012, p. 36-49.

[33] « Sobel operator - Wikipedia, the free encyclopedia ». [En ligne]. Disponible à: https://en.wikipedia.org/wiki/Sobel_operator. [Consulté le: 12-juin-2016].

[34] Dan C. Ciresan, Ueli Meier, Jonathan Masci, et Luca M. Gambardella and Jurgen Schmidhuber, « High-Performance Neural Networks for Visual Object Classification », January 20011.

[35] C. Cao, X. Liu, Y. Yang, Y. Yu, J. Wang, Z. Wang, Y. Huang, L. Wang, C. Huang, W. Xu, D. Ramanan, et T. S. Huang, « Look and Think Twice: Capturing Top-Down Visual Attention with Feedback Convolutional Neural Networks », 2015, p. 2956-2964.

ANNEXES

ANNEXE A – ALGORITHME D'ANALYSE DE FORMES ET DE DIMENSIONS 3D

L'algorithme d'analyse de formes et de dimensions 3D permet d'analyser un nuage de points d'un objet quelconque en déterminant une forme qui lui correspond le mieux par la différence d'erreur volumique et puis de déterminer les dimensions de cet objet en fonction du volume correspondant.

La première étape est de charger le nuage de points de l'objet qui a été filtré et de trouver les points minimums et maximums sur les trois axes (X, Y et Z). Cela permet ensuite de trouver les dimensions extérieures pour donner une idée générale de l'objet.

```
% Charger le nuage de points stl de l'objet
[f2, v2, n2] = lire_STL('volume_filtrer\scene_bouteille_GDebout.stl', false

% Extremum du volume de l'objet
min_x = min( v2(:,1)); min_y = min( v2(:,2)); min_z = min( v2(:,3));
max_x = max( v2(:,1)); max_y = max( v2(:,2)); max_z = max( v2(:,3));

% Dimension hors de l'objet
dimension_hors_x = (max_x - min_x); dimension_hors_y = (max_y - min_y); dim
```

Figure 7.1 : Code de chargement et de calcul général.

Avec ces calculs, l'algorithme approxime le centre du nuage de points et le ramène à l'origine globale pour faciliter l'analyse.

```
% Centroide de l'objet
centre_x =  min_x + dimension_hors_x/2; centre_y =  min_y + dimension_hors_

% Remettre l'objet a l'origine
v2(:,1) = v2(:,1) - centre_x;
v2(:,2) = v2(:,2) - centre_y;
v2(:,3) = v2(:,3) - centre_z;
```

Figure 7.2 : Code pour mettre à l'origine le centre du nuage de points.

Une fois le nuage centré, une boucle sur les différentes formes à analyser est démarrée afin de déterminer la forme la plus correspondante. Les trois formes utilisées dans l'analyse du programme sont le cylindre, le prisme rectangulaire et la sphère.

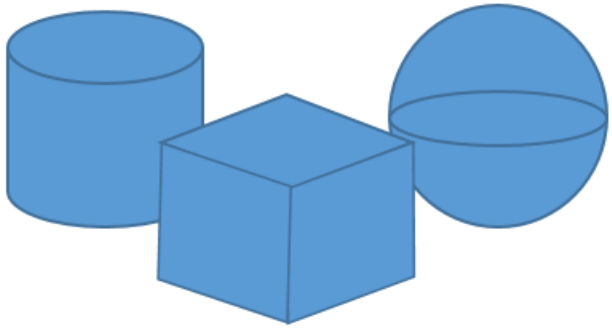

Figure 7.3 : Formes de base.

Ces trois formes ont été modélisées avec CATIA et mises sous format STL texte afin de fonctionner avec le programme d'analyse de formes.

```
switch formei
    case 1
        % Lire volume de reference stl
        [f, v, n] = lire_STL('volume_base\cylindre_d20h200.stl',false);
        forme = 'cylindre';
    case 2
        [f, v, n] = lire_STL('volume_base\prismeRect.stl',false);
        forme = 'prisme';
    case 3
        [f, v, n] = lire_STL('volume_base\sphere.stl',false);
        forme = 'sphere';
    otherwise
end
```

Figure 7.4 : Chargement des formes de base pour l'analyse.

Après le chargement d'une forme de base, on positionne son centre à l'origine. Dépendamment de la forme analysée, le programme engendre différentes lignes de code.

Pour l'analyse du cylindre, il faut réfléchir à l'orientation qu'il doit prendre pour correspondre le mieux au nuage de points acquis. La manière utilisée est de choisir les deux dimensions extérieures qui sont les plus similaires et de mettre le cercle de la base du cylindre dans ce plan.

```
% Rotation intelligente du cylindre
if (strcmp(forme,'cylindre'))
    erreur_x_y = abs(dimension3_hors_x - dimension3_hors_y);
    erreur_x_z = abs(dimension3_hors_x - dimension3_hors_z);
    erreur_y_z = abs(dimension3_hors_y - dimension3_hors_z);
    erreur_cyn = [erreur_x_y, erreur_x_z, erreur_y_z];
    [erreur_ordre, err] = sort(erreur_cyn);
    decision = err(:,1);
    switch decision
        case 1
            fprintf('met ton cercle de cylindre en x_y')
            ratio_cyl = 1;
        case 2
            fprintf('met ton cercle de cylindre en x_z')
            v = v * Rx(pi/2);
            ratio_cyl = 2;
        case 3
            fprintf('met ton cercle de cylindre en y_z')
            v = v * Ry(pi/2);
            ratio_cyl = 3;
    end
end
```

Figure 7.5 : Rotation du cylindre pour la correspondance.

Puis, nous trouvons les dimensions correspondant le mieux avec des ratios pour changer l'échelle des dimensions des nuages de base.

Voici le calcul des ratios :

```
% Calcul des ratios des changement d echelle
ratio_x = (dimension3_hors_x/dimension_ref_x(formei));
ratio_y = (dimension3_hors_y/dimension_ref_y(formei));
ratio_z = (dimension3_hors_z/dimension_ref_z(formei));
```

Figure 7.6 : Calcul des ratios.

Avec ces ratios, nous nous assurons de trouver le cylindre qui correspond aux plus petites des deux dimensions similaires et nous modifions l'échelle avec le code suivant :

```
case 'cylindre'
    switch ratio_cyl
        case 1
            ratio_intel = [ratio_x, ratio_y];
            ratio_min = min(ratio_intel);
            v(:,1) = v(:,1)* ratio_min;
            v(:,2) = v(:,2)* ratio_min;
            v(:,3) = v(:,3)* ratio_z;
        case 2
            ratio_intel = [ratio_x, ratio_z];
            ratio_min = min(ratio_intel);
            v(:,1) = v(:,1)* ratio_min;
            v(:,2) = v(:,2)* ratio_y;
            v(:,3) = v(:,3)* ratio_min;
        case 3
            ratio_intel = [ratio_y, ratio_z];
            ratio_min = min(ratio_intel);
            v(:,1) = v(:,1)* ratio_x;
            v(:,2) = v(:,2)* ratio_min;
            v(:,3) = v(:,3)* ratio_min;
    end
```

Figure 7.7 : Code de changement d'échelle pour le cylindre.

Pour la sphère, puisqu'elle n'a pas d'orientation spécifique, un simple changement d'échelle est appliqué en prenant le ratio maximum pour que la sphère de base vienne englober le nuage de points de l'objet acquis.

```
case 'sphere'
    % Pour la sphere prendre dimension max du ratio
    ratio_intel = [ratio_x, ratio_y, ratio_z];
    ratio_max = max(ratio_intel);
    v(:,1) = v(:,1)* ratio_max;
    v(:,2) = v(:,2)* ratio_max;
    v(:,3) = v(:,3)* ratio_max;
```

Figure 7.8 : Code de changement d'échelle pour la sphère.

Pour le prisme, l'orientation n'est pas importante parce que nous venons faire correspondre les surfaces. Par contre, trouver la surface qui approxime le mieux le nuage de points est critique, car elle ne peut pas correspondre uniquement au minimum ou au maximum du nuage de points parce que cela apporte trop d'imprécision si le nuage acquis est dans une orientation quelconque. Dans ce cas, nous calculons la surface moyenne avec tous les points.

Voici un extrait du code :

```
% Calcul de la surface moyenne
surf_moy_X_min = somme(1) / s(1);
surf_moy_X_max = somme(2) / s(2);
surf_moy_Y_min = somme(3) / s(3);
surf_moy_Y_max = somme(4) / s(4);
surf_moy_Z_min = somme(5) / s(5);
surf_moy_Z_max = somme(6) / s(6);

 ratio_x = ((dimension3_hors_x-((surf_moy_X_min+surf_moy_X_max)/
 ratio_y = ((dimension3_hors_y-((surf_moy_Y_min+surf_moy_Y_max)/
 ratio_z = ((dimension3_hors_z-((surf_moy_Z_min+surf_moy_Z_max)/
 v(:,1) = v(:,1)* ratio_x;
 v(:,2) = v(:,2)* ratio_y;
 v(:,3) = v(:,3)* ratio_z;
```

Figure 7.9 : Code de changement d'échelle pour le prisme.

Ensuite, nous calculons le volume approximatif du nuage de points de l'objet acquis et les volumes approximatifs des formes de base modifiées en orientation et en échelle. Le programme calcule les différences d'erreurs des volumes, puis prend la décision de la forme la plus correspondante au nuage de points.

```
% Calcul des volumes
volume_approx_ref = volume_approx(f,v)*1000; % (triangle , sommets)
volume_approx_obj = volume_approx(f2,v3)*1000; % (triangle , sommets)

% Erreur de volume
erreur_volume(formei) = abs(volume_approx_obj - volume_approx_ref);
```

Figure 7.10 : Calcul des volumes et erreurs sur les volumes.

Le calcul du volume approximatif se fait avec la formule suivante :

$$Volume = \left(\frac{1}{6}\right) * \sum\left((vi2y - vi1y) * (vi3z - vi1z) - (vi2z - vi1z) * (vi3y - vi1y)\right) * (vi1x + vi2x + vi3x)$$

où les vi sont les points de fermeture des triangles dans le sens horaire.

Voici le code de calcul du volume :

```
% calcule du volume
    term1 = (sommet2y-sommet1y)*(sommet3z-sommet1z);
    term2 = (sommet2z-sommet1z)*(sommet3y-sommet1y);
    term3 = (sommet1x+sommet2x+sommet3x);
    som = som + ((term1-term2)*term3);
end

som = som / 6;
volume = som;
```

Figure 7.11 : Code de calcul du volume.

Voici le code de la prise de décision de la forme :

```
[choix_forme, choix]= min(erreur_volume);
switch choix
    case 1
        fprintf('\n');
        fprintf('La forme moyenne est un Cylindre\n');
        forme = 'cylindre';
        fprintf('De diametre: %f mm et de hauteur: %f mm \n', dimension_re

    case 2
        fprintf('\n');
        fprintf('La forme moyenne est un Prisme Rectangulaire\n');
        forme = 'prisme';
        fprintf('De cote: en x: %f cm, en y: %f cm et en z: %f cm \n', dime

    case 3
        fprintf('\n');
        fprintf('La forme moyenne est une Sphere\n');
        forme = 'sphere';
        fprintf('De diametre: %f mm \n', dimension_max_ref(1,3)*100);
end
```

Figure 7.12 : Prise de décision sur la forme.

Enfin, toutes les données recueillies sur le nuage de points acquis sont enregistrées dans une base de données simple.

ANNEXE B – BASE DE DONNÉES SIMPLE

La base de données simple est composée de tous les nuages de points analysés par l'algorithme d'analyse de formes et de dimensions 3D. Cette base comprend le id, le nom, la forme, les dimensions dépendamment de la forme, les données des points dans l'espace et les données des triangles du nuage de points.

```
% Base de données
load ('base_de_donnees');
id = size(base_de_donnees,1) + 1;
base_de_donnees{id, 1} = id;
nom = input('nom de l''objet :'); % information donnee
base_de_donnees{id, 2} = nom;
base_de_donnees{id, 3} = forme;
switch forme
    case 'cylindre'
        base_de_donnees{id, 4} = dimension_cyl_dia;
        base_de_donnees{id, 5} = dimension_cyl_h;
        base_de_donnees{id, 6} = 0;
    case 'prisme'
        base_de_donnees{id, 4} = dimension_ref_x(1,2);
        base_de_donnees{id, 5} = dimension_ref_y(1,2);
        base_de_donnees{id, 6} = dimension_ref_z(1,2);
    case 'sphere'
        base_de_donnees{id, 4} = dimension_max_ref(1,3);
        base_de_donnees{id, 5} = 0;
        base_de_donnees{id, 6} = 0;
end
base_de_donnees{id, 7} = v2;
base_de_donnees{id, 8} = f2;
save ('base_de_donnees.mat','base_de_donnees');
```

Figure 7.13 : Code de la base de données simple pour le cylindre.

ANNEXE C – ALGORITHME DE CONVOLUTION 3D

L'algorithme de convolution débute par le chargement du nuage de points de la scène. Ensuite, elle transforme le nuage de points en matrice de 0 et de 1, où 1 représente un point réel dans l'espace de la scène acquise. Pour ce faire, l'algorithme trouve le centre du nuage de points acquis et le convertit en le centre de la matrice de 0 et de 1. Puis, il convertit de façon linéaire les points du nuage en points vers la matrice remplie de 0 en mettant un 1 au bon endroit selon un pas de précision qui dépend de la résolution des voxels d'acquisition des nuages.

```
% Centre de la matrice 3D de 0 et 1
centreX = n_divX/2; centreY = n_divY/2; centreZ = n_divZ/2;

% Pente de transformation
m_transX = (n_divX-1)/(maxB_x-minB_x);
m_transY = (n_divY-1)/(maxB_y-minB_y);
m_transZ = (n_divZ-1)/(maxB_z-minB_z);
m3DX = round(m_transX*v2(:,1) + centreX);
m3DY = round(m_transY*v2(:,2) + centreY);
m3DZ = round(m_transZ*v2(:,3) + centreZ);

for i=1:size(m3DX,1)
    matrice_0_1(m3DX(i)+1, m3DY(i)+1, m3DZ(i)+1) = 1;
end
```

Figure 7.14 : Code pour la transformation du nuage vers la matrice.

Par la suite, selon l'objet désiré de l'utilisateur, le programme va rechercher dans la base de données simple l'objet désiré avec ses caractéristiques à analyser en position et en orientation.

```
% Chargement de la base de donnees
load ('base_de_donnees');
%base_de_donnees = base_de_donnees.base_de_donnees;
nom = input('nom de l''objet :'); % information donnee
for ii = 1:size(base_de_donnees,1)
    if ( strcmp(nom, base_de_donnees{ii, 2}) )
        id = base_de_donnees{ii, 1};
        nom = base_de_donnees{ii, 2};
        forme = base_de_donnees{ii, 3};
        switch forme
            case 'cylindre'
                dimension_cyl_dia = base_de_donnees{ii, 4};
                dimension_cyl_h = base_de_donnees{ii, 5};
            case 'prisme'
                dimension_x = base_de_donnees{ii, 4};
                dimension_y = base_de_donnees{ii, 5};
                dimension_z = base_de_donnees{ii, 6};
            case 'sphere'
                dimension_sphere_dia = base_de_donnees{ii, 4};
        end
        v2 = base_de_donnees{ii, 7};
        f2 = base_de_donnees{ii, 8};
    end
end
```

Figure 7.15 : Code pour obtenir les caractéristiques de l'objet désiré.

Pour gagner du temps de calcul, le programme regarde la forme de l'objet désiré et prend une décision sur la procédure de calcul de la convolution.

La sphère prend moins de temps de calcul pour l'analyse, car elle ne possède pas d'orientation spécifique. Donc une simple translation dans l'espace est nécessaire en appliquant la convolution.

```
case 'sphere'
    % Transformation du nuage de points en matrice01
    [ objet, objX, objY, objZ ] = transformation_0_1(v2, Pas);

    % Affichage de l'objet_0_1
    figure()
    scatter3(objX,objY,objZ)
    xlabel('x-direction');ylabel('y-direction');zlabel('z-direction');
    axis equal;

    % Convolution
    c = convn(scene, objet(end:-1:1, end:-1:1, end:-1:1));
```

Figure 7.16 : Code pour le cas d'une sphère.

Le cylindre prend un peu plus de temps, car le programme doit analyser les translations et les orientations possibles dans deux plans.

```
case 'cylindre'
    % Etape 1
    % Transformation du nuage de points en matrice01
    [ objet, objX, objY, objZ ] = transformation_0_1(v2, Pas);

    % Convolution
    c = convn(scene, objet(end:-1:1, end:-1:1, end:-1:1));
    stack_c{1,1} = max(max(max(c)));
    stack_c{2,1} = c;
    stack_c{3,1} = objet;
    stack_c{4,1} = 0;
    stack_c{5,1} = 0;
    stack_c{6,1} = 0;

    % Etape 2
    % Changement d'angle
    v3 = v2 * Rx(90*pi/180);
    for jj = 2:10
        for ii = 1:6
            stack_c{ii,jj} = 0;
        end
    end
    jj = 2;
    for angle=0*pi/180:15*pi/180:180*pi/180
        % Objet cylindre coucher
        v2 = v3;
        % Rotation
        v2 = v2 * Ry(angle);
        % Transformation du nuage de points en matrice01
        [ objet, objX, objY, objZ ] = transformation_0_1(v2, Pas);

        % Convolution
        c = convn(scene, objet(end:-1:1, end:-1:1, end:-1:1));
        stack_c{1,jj} = max(max(max(c)));
        stack_c{2,jj} = c;
        stack_c{3,jj} = objet;
        stack_c{4,jj} = 90;
        stack_c{5,jj} = angle*180/pi;
        stack_c{6,jj} = 0;
        jj = jj + 1;
    end
```

Figure 7.17 : Code pour le cas d'un cylindre.

Pour le prisme, il faut prendre en considération toutes les translations avec toutes les orientations, cela peut prendre plus de temps. Le code dans ce cas est similaire au cylindre, mais il possède une étape de plus. Pour ne pas prendre trop de temps d'analyse, il faut choisir une précision de rotation. Pour toutes les formes, l'angle choisi pour l'analyse est de 22.5 degrés de rotation, ce qui donne un temps raisonnable pour le calcul d'un prisme rectangulaire.

Enfin, l'algorithme prend la décision sans appel sur le maximum de corrélation avec la convolution parmi les possibilités dépendamment du retour sur les rotations et sur les translations combinées qui se retrouvent dans une cellule tableau.

```
% Donner les coordonnees dans l'espace
stack = [];
for zz = 1:size(stack_c,2)
    stack(1,zz) = [stack_c{1,zz}];
end
[valeur, index] = max(stack);
c = stack_c{2,index};
objet = stack_c{3,index};
[ x, y, z ] = Coordonnees_espace(c, objet);
rotX = stack_c{4,index}; rotY = stack_c{5,index}; rotZ = stack_c{6,index};
```

Figure 7.18 : Code de décision pour la rotation/translation.

ANNEXE D – ANALYSE DE SCÈNE RAPIDE

Nous présentons dans cette section le code développé pour le système actuel. L'algorithme d'analyse de scène rapide permet de détecter, de prendre et de reconnaitre des objets d'une scène. Elle fournir le centre (la position), deux endroits de préhension, une phase d'approche et une reconnaissance de tous les objets d'une scène acquise avec la Kinect V2.

Variables du système.

Variables modifiables dans le programme pour la segmentation de l'ensemble de données.

```
clear all;
clc;
close all;

% Variables de fonctionnement
num_class = 1;
num_objet = 1;
total_objet = 0;
ajustement_hauteur = 1;
ajout_contour = 5;
rayon_p = 2;
rayon_m = 1;
dim_s = 5;

% Variables modifiables
Pas = 0.01;
grosseur_pince = 3;
espace01 = 80;
```

Figure 7.19 : Variables du programme.

Les paramètres modifiables sont le PAS (pour modifier le nombre de points du nuage conservé), GROSSEUR PINCE (pour modifier la grosseur d'ouverture maximum de la pince utilisée par l'effecteur) et l'ESPACE01 (pour modifier la dimension à prendre en compte d'un objet pour l'apprentissage).

Lecture automatique de l'ensemble de données des scènes acquises.

Lecture de scènes acquises pour l'apprentissage.

```
%% Lecture Automatique
vectinput{1} = [];
for num_class = 1:12
    fprintf('\nclasse:%0.0f\n',num_class)
     for num_objet = 1:15
         clear scene sceneX sceneY sceneZ sceneXYZ centroide objets_nuages

        % Chargement de la scene STL Binaire
        [Elements,varargout] = plyread(strcat('Scene_PLY\objets\P',...
            num2str(num_class),'\',num2str(num_objet),'.ply'));
```

Figure 7.20 : Lecture automatique de l'ensemble de données.

Création des vecteurs avec la couleur des points de chaque objet segmenté pour l'apprentissage avec le réseau de neurones.

```
        %% Concevoir un vecteur colonne avec la couleur
        total_objet = total_objet + 1;
        color_num = 0;
        moitier01 = espace01/2;
        for objet = 1:nb_objet
            % Transformation de coordonnee vers une matrice 3D
            inputs = [];
            matrice01 = zeros(espace01,espace01,espace01);
            for i = 1:size(objets_nuages{objet,1},1)
                color_num = 65536 * objets_nuages{objet,1}(i,4) + ...
                    256 * objets_nuages{objet,1}(i,5) + ...
                    objets_nuages{objet,1}(i,6);
                matrice01(abs(objets_nuages{objet,1}(i,1)+...
                    (moitier01-centroide(objet,1))), ....
                    abs(objets_nuages{objet,1}(i,2)+....
                    (moitier01-centroide(objet,2))), ...
                    abs(objets_nuages{objet,1}(i,3)+...
                    (moitier01-centroide(objet,3)))) = color_num;
            end
            for couche = 1:size(matrice01,3)
                mat01 = matrice01(:,:,couche);
                inputs = [inputs im2col(mat01, [1,1])];
            end
        end
        vectinput{total_objet,1} = inputs;
    end
```

Figure 7.21 : Création des vecteurs avec couleur.

Mettre tous les vecteurs ensemble pour les présenter au réseau de neurones.

```
for ii = 1:size(vectinput,1)
    matinput(:,ii) = vectinput{ii,1};
end
```

Figure 7.22 : Entrées pour le réseau de neurones.

Code pour l'apprentissage avec le réseau de neurones.

```
%% Apprentissage avec un Réseaux de neurones
clear all;
clc;
close all;

load('PLY_Pattern\input_ply_12C_80space_256bits.mat');
inputs = input_ply_12C_80space_256bits; %1000000x150
load('PLY_Pattern\target_ply_12C.mat');
targets = target_ply_12C; %10x150

% Create a Pattern Recognition Network
hiddenLayerSize = [100 50 100];
net = patternnet(hiddenLayerSize);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 90/100;
net.divideParam.valRatio = 9/100;
net.divideParam.testRatio = 1/100;

% Train the Network
[net,tr] = train(net,inputs,targets);

% Test the Network
outputs = net(inputs);
errors = gsubtract(targets,outputs);
performance = perform(net,targets,outputs);

% Plots
figure, plotperform(tr)
figure, plottrainstate(tr)
figure, plotconfusion(targets,outputs)
figure, ploterrhist(errors)

% View the Network
view(net)
```

Figure 7.23 : Apprentissage avec un réseau de neurones.

Nous pouvons modifier certains paramètres du réseau de neurones comme les vecteurs d'entrées, les classes de sorties, le nombre de couches cachées et leur nombre de neurones. Nous pouvons aussi modifier les ratios d'apprentissage, de validation et de test sur l'ensemble de données. Les affichages ressortis sont le graphique de performance, le graphique des états d'entrainement, le graphique de la matrice de confusion, un histogramme et la structure du net. Une fois satisfait de l'apprentissage, il faut enregistrer le net.

Code pour la détection d'objet.

Lecture d'une scène acquise.

```
% Chargement de la scene PLY Binaire
%[Elements,varargout] = plyread('100.ply', 'false');
nom_scene = '011';
[Elements,varargout] = plyread(strcat('Scene_PLY\scenes\',num2str(nom_scene
v_scene(:,1) = Elements.vertex.x;
v_scene(:,2) = Elements.vertex.y;
v_scene(:,3) = Elements.vertex.z;
vscene(:,1) = v_scene(:,1);
vscene(:,2) = v_scene(:,2);
vscene(:,3) = v_scene(:,3);
v_RGB(:,1) = Elements.vertex.red;
v_RGB(:,2) = Elements.vertex.green;
v_RGB(:,3) = Elements.vertex.blue;

% Angle de camera
% Besoin d'une méthode automatique pour trouver l'angle....
v_scene = v_scene * Rx(-angle*pi/180);

% Extremum du volume
min_x = min( v_scene(:,1)); min_y = min( v_scene(:,2)); min_z = min( v_scen
max_x = max( v_scene(:,1)); max_y = max( v_scene(:,2)); max_z = max( v_scen

% Dimension hors
dimension_hors_x = (max_x - min_x); dimension_hors_y = (max_y - min_y); dim

% Centroide
centre_x =  min_x + dimension_hors_x/2; centre_y =  min_y + dimension_hors_

% Remettre a l'origine
v_scene(:,1) = v_scene(:,1) - centre_x;
v_scene(:,2) = v_scene(:,2) - centre_y;
v_scene(:,3) = v_scene(:,3) - centre_z;

% Transformation du nuage de points vers une matrice 3D
[ scene, sceneX, sceneY, sceneZ, m_transX, m_transY, m_transZ, centreX, cen
```

Figure 7.24 : Code de la lecture d'une scène acquise par la Kinect V2 en PLY.

Transformation du nuage de points en une matrice.

```
function [ matrice_0_1, m3DX, m3DY, m3DZ, m_transX, m_transY, m_transZ, ce
% Extremum du volume de l'objet
min_x = min( v_nuage(:,1)); min_y = min( v_nuage(:,2)); min_z = min( v_nua
max_x = max( v_nuage(:,1)); max_y = max( v_nuage(:,2)); max_z = max( v_nua

% Dimension exterieur du nuage
dim_hors_x = (max_x - min_x); dim_hors_y = (max_y - min_y); dim_hors_z = (

% Centre du nuage
centre_x =  min_x + dim_hors_x/2; centre_y =  min_y + dim_hors_y/2; centre_

n_divX = round(((max_x-min_x)/Pas)+1);
n_divY = round(((max_y-min_y)/Pas)+1);
n_divZ = round(((max_z-min_z)/Pas)+1);
matrice_0_1 = zeros(n_divX,n_divY,n_divZ);

% Centre de la matrice 3D de 0 et 1
centreX = n_divX/2; centreY = n_divY/2; centreZ = n_divZ/2;

% Pente de transformation
m_transX = (n_divX-1)/(dim_hors_x);
m_transY = (n_divY-1)/(dim_hors_y);
m_transZ = (n_divZ-1)/(dim_hors_z);
m3DX = round(m_transX*v_nuage(:,1) + centreX);
m3DY = round(m_transY*v_nuage(:,2) + centreY);
m3DZ = round(m_transZ*v_nuage(:,3) + centreZ);

for i=1:size(m3DX,1)
    matrice_0_1(m3DX(i)+1, m3DY(i)+1, m3DZ(i)+1) = 1;
end

end
```

Figure 7.25 : Code de la transformation du nuage de points en une matrice.

Algorithme de palpage par le haut pour détecter les objets.

```
%% Algorithmes de palpage
% Palpage par le haut
ligne_thy = 0;
tab_haut = 0;
for xx = 1:size(scene,1)
    for zz = 1:size(scene,3)
        ligne_thy = ligne_thy + 1;
        for yy = size(scene,2):-1:1
            if (scene(xx,yy,zz) == 1)
                tab_haut(ligne_thy,1) = xx;%%yy;
                tab_haut(ligne_thy,2) = yy;%%xx;
                tab_haut(ligne_thy,3) = zz;%%zz;
                break;
            end
            if (yy == 1)
                tab_haut(ligne_thy,1) = xx;
                tab_haut(ligne_thy,2) = yy;
                tab_haut(ligne_thy,3) = zz;
                break;
            end
        end
    end
end
% Compteur de valeurs similaires
tab_compte_y = zeros(size(scene,2),1);
for ligne = 1:size(tab_compte_y,1)
    tab_compte_y(ligne,2) = ligne;
end
for ligne_thy = 1:size(tab_haut,1)
    for hauteur_y = 1:size(scene,2)
        if (tab_haut(ligne_thy,2) == hauteur_y)
            tab_compte_y(hauteur_y,1) = tab_compte_y(hauteur_y,1) + 1;
            break;
        end
    end
end

% Deduire la hauteur de la table
[valmax, ligne] = max(tab_compte_y(2:size(tab_compte_y,1),1));
ligne = ligne +1;
hauteur_table = tab_compte_y(ligne,2) + ajustement_hauteur;

% Enlever les points en dessous de la table (tabletab_haut)
for ligne_thy = size(tab_haut,1):-1:1
    if (tab_haut(ligne_thy,2) <= hauteur_table)
        tab_haut(ligne_thy,:) = [];
    end
end
```

Figure 7.26 : Code de l'algorithme de palpage par le haut.

Algorithme d'élimination de points indésirables par gradient et comptage du nombre d'objets dans la scène.

```
%% Compte des objets dans la scene
% Avec Image de profondeur et detecteur de contour
nb_objet = 0;
max_ligne = max(tab_haut(:,3));
max_col = max(tab_haut(:,1));
map_deep = zeros(max_ligne,max_col);
for ligne = 1:size(tab_haut,1)
    map_deep(tab_haut(ligne,3),tab_haut(ligne,1)) = tab_haut(ligne,2);
end

% Remplir les trous de l'image
map_deep_extend = padarray(map_deep, [ajout_contour ajout_contour], 0);
map_deep_extend = imclose(map_deep_extend, strel('disk', 1));
map_deep_extend = imclose(map_deep_extend, strel('line', 4, 90));
% map_deep_extend = imclose(map_deep_extend, strel('disk', 1));

% Calculer le gradient de l'image
[gradX, gradY] = gradient(map_deep_extend);

% Calculer la moyenne en chaque points en utilisant les 3 points derrière
% lui et lui-même
[gradY] = conv2(gradY, [1 1 1 1 0 0 0]', 'same');
[gradY] = conv2(gradY, [1 1 1 1 0 0 0]', 'same');
[gradY] = conv2(gradY, [1 1 1 1 0 0 0]', 'same');

% Si la dérivée en Y est >= 10, c'est que ça fait partie de la trainée.
% Ensuite on utilise imdilate() et imclose() pour refermer les trous
pointsTraine = gradY >= 11;
pointsTraine = imdilate(pointsTraine, strel('disk', 1));
pointsTraine = imclose(pointsTraine, strel('line', 5, 90));
pointsTraine = ~imclose(~pointsTraine, ones(3));


% Conserver seulement la trainée
map_deep_noTraine = map_deep_extend;
map_deep_noTraine(pointsTraine) = 0;
trailClean = (map_deep_noTraine <= 0);
trailClean = ~imclose(trailClean, strel('disk', 2));
map_deep_noTraine = trailClean .* map_deep_noTraine;

% Trouver les contours de l'image
contour = edge(map_deep_noTraine,'sobel',1);
contour = imclose(contour, strel('disk', 1));
contour = imdilate(contour, ones(2));

% Trouver l'intérieur des objets
interieur = 1 - contour;
interieur = bwareaopen(interieur, 5);
eti2 = bwlabel(interieur);

% Étendre les objets pour qu'ils bouffent le contour
eti2_noBack = eti2;
eti2_noBack(eti2== eti2(1,1)) = 0;
eti2_noBack = imdilate(eti2_noBack, strel('disk', 2));
eti2_noBack(eti2== eti2(1,1)) = 0;
statsr = regionprops(eti2_noBack, 'Area','Centroid', 'BoundingBox');
nb_objet = size(statsr,1)-1;
```

Figure 7.27 : Algorithme d'élimination de points indésirable et comptage des objets.

Code pour la préhension d'objets.

Séparation des sous-matrices d'objets après la segmentation.

```
%% Séparer les sous-matrice 3D objets
% Obtenir les centroides à trois dimensions (Centre X, Y et Z)
for objet = 2:nb_objet+1
    centroide(objet-1,1) = round(statsr(objet,1).Centroid(1,1)-ajout_contou
    centroide(objet-1,3) = round(statsr(objet,1).Centroid(1,2)-ajout_contou
    for ligne = 1:size(tab_haut,1)
        if ( (tab_haut(ligne,1)==centroide(objet-1,1)) && (tab_haut(ligne,3
            centroide(objet-1,2) = round(((tab_haut(ligne,2)-hauteur_table)
        end
    end
end
% Obtenir l'espace d'un objet (Espace X, Y et Z)
for objet = 2:nb_objet+1
    espace(objet-1,1) = round(statsr(objet,1).BoundingBox(1,1)-ajout_contou
    espace(objet-1,2) = round(statsr(objet,1).BoundingBox(1,1)-6+statsr(obj
    espace(objet-1,5) = round(statsr(objet,1).BoundingBox(1,2)-ajout_contou
    espace(objet-1,6) = round(statsr(objet,1).BoundingBox(1,2)-ajout_contou
    for ligne = 1:size(tab_haut,1)
        if ( (tab_haut(ligne,1)==centroide(objet-1,1)) && (tab_haut(ligne,3
             espace(objet-1,3) = hauteur_table+1;
             espace(objet-1,4) = tab_haut(ligne,2);
        end
    end
end
% Obtenir les nuages de points de chaque objets
sceneXYZ(:,1) = sceneX;
sceneXYZ(:,2) = sceneY;
sceneXYZ(:,3) = sceneZ;
for objet = 1:nb_objet
    ligne_obj = 0;
    for ligne = 1:size(sceneXYZ,1)
        if (sceneXYZ(ligne,1)>=espace(objet,1) && sceneXYZ(ligne,1)<=espace
            sceneXYZ(ligne,2)>=espace(objet,3) && sceneXYZ(ligne,2)<=espace
            sceneXYZ(ligne,3)>=espace(objet,5) && sceneXYZ(ligne,3)<=espace
                ligne_obj = ligne_obj + 1;
                objets_nuages{objet,1}(ligne_obj,1) = sceneXYZ(ligne,1);
                objets_nuages{objet,1}(ligne_obj,2) = sceneXYZ(ligne,2);
                objets_nuages{objet,1}(ligne_obj,3) = sceneXYZ(ligne,3);
                objets_nuages{objet,1}(ligne_obj,4) = v_RGB(ligne,1);
                objets_nuages{objet,1}(ligne_obj,5) = v_RGB(ligne,2);
                objets_nuages{objet,1}(ligne_obj,6) = v_RGB(ligne,3);
        end
    end

end
 % Retrouver le meilleur centre du nuage de points
for objet = 1:nb_objet
     minonx = min(objets_nuages{objet,1}(:,1));
     maxonx = max(objets_nuages{objet,1}(:,1));
     minonz = min(objets_nuages{objet,1}(:,3));
     maxonz = max(objets_nuages{objet,1}(:,3));
     centroide(objet,1) = round(minonx + (abs(maxonx-minonx))/2);
     centroide(objet,3) = round(minonz + (abs(maxonz-minonz))/2);
end
```

Figure 7.28 : Code de séparation des sous-matrices d'objets.

Arbre de décision de l'approche et des endroits de préhension.

```
%% Obtenir les lieux d'intérêts pour prendre l'objet
for objet = 1:nb_objet
    for ligne = 1:size(objets_nuages{objet,1},1)
        dist_centre_surf{objet,1}(1,ligne) = ...
            sqrt((centroide(objet,1)-objets_nuages{objet,1}(ligne,1))^2+...
                 (centroide(objet,2)-objets_nuages{objet,1}(ligne,2))^2+...
                 (centroide(objet,3)-objets_nuages{objet,1}(ligne,3))^2);
        dist_centre_surf{objet,1}(2,ligne) =  objets_nuages{objet,1}(ligne,
        dist_centre_surf{objet,1}(3,ligne) =  objets_nuages{objet,1}(ligne,
        dist_centre_surf{objet,1}(4,ligne) =  objets_nuages{objet,1}(ligne,
    end
    % Extremum du volume de l'objet
    min_obj_x = min(objets_nuages{objet,1}(:,1)); min_obj_y = min(objets_nu
    max_obj_x = max(objets_nuages{objet,1}(:,1)); max_obj_y = max(objets_nu
    % Dimension hors de l'objet
    dimension_obj(objet,1) = (max_obj_x - min_obj_x);
    dimension_obj(objet,2) = (max_obj_y - min_obj_y);
    dimension_obj(objet,3) = (max_obj_z - min_obj_z);
end

% Conditions: Si le Y est la plus grande dimension,
% prendre une approche par le côté, sinon prendre par le haut.
for objet = 1:nb_objet
    if (dimension_obj(objet,2)+4>=dimension_obj(objet,1) && dimension_obj(o
        % Prendre par le côté
        prise{objet,1} = 'côté';
        if (dimension_obj(objet,1) > dimension_obj(objet,3))
            % Par direction Z
            pouce(objet,1) = centroide(objet,1);
            pouce(objet,2) = centroide(objet,2);
            col = 0;
            for ligne = 1:size(objets_nuages{objet,1},1)
                if (objets_nuages{objet,1}(ligne,1)== pouce(objet,1))
                    col = col + 1;
                    on_pouce_z{objet,1}(1,col)= objets_nuages{objet,1}(ligne
                end
            end
            pouce(objet,3) = max(on_pouce_z{objet,1}(1,:));
            majeur(objet,1) = centroide(objet,1);
            majeur(objet,2) = centroide(objet,2);
            majeur(objet,3) = min(on_pouce_z{objet,1}(1,:));
        else
```

```
                % Par direction X
                pouce(objet,2) = centroide(objet,2);
                pouce(objet,3) = centroide(objet,3);
                col = 0;
                for ligne = 1:size(objets_nuages{objet,1},1)
                    if (objets_nuages{objet,1}(ligne,3)== pouce(objet,3))
                        col = col + 1;
                        on_pouce_x{objet,1}(1,col)= objets_nuages{objet,1}(lig
                    end
                end
                pouce(objet,1) = min(on_pouce_x{objet,1}(1,:));
                majeur(objet,2) = centroide(objet,2);
                majeur(objet,3) = centroide(objet,3);
                majeur(objet,1) = max(on_pouce_x{objet,1}(1,:));
            end
        else
            % Prendre par le haut
            prise{objet,1} = 'haut';
            if (dimension_obj(objet,1) > dimension_obj(objet,3))
                % Par direction Z
                pouce(objet,1) = centroide(objet,1);
                pouce(objet,2) = centroide(objet,2);
                col = 0;
                for ligne = 1:size(objets_nuages{objet,1},1)
                    if (objets_nuages{objet,1}(ligne,1)== pouce(objet,1))
                        col = col + 1;
                        on_pouce_z{objet,1}(1,col)= objets_nuages{objet,1}(lig
                    end
                end
                pouce(objet,3) = max(on_pouce_z{objet,1}(1,:));
                majeur(objet,1) = centroide(objet,1);
                majeur(objet,2) = centroide(objet,2);
                majeur(objet,3) = min(on_pouce_z{objet,1}(1,:));
            else
                % Par direction X
                pouce(objet,2) = centroide(objet,2);
                pouce(objet,3) = centroide(objet,3);
                col = 0;
                for ligne = 1:size(objets_nuages{objet,1},1)
                    if (objets_nuages{objet,1}(ligne,3)== pouce(objet,3))
                        col = col + 1;
                        on_pouce_x{objet,1}(1,col)= objets_nuages{objet,1}(lig
                    end
             end
             pouce(objet,1) = min(on_pouce_x{objet,1}(1,:));
             majeur(objet,2) = centroide(objet,2);
             majeur(objet,3) = centroide(objet,3);
             majeur(objet,1) = max(on_pouce_x{objet,1}(1,:));
        end
    end
end
```

Figure 7.29 : Code de l'arbre de décision.

Code pour la reconnaissance d'objets.

Classification des objets par le réseau de neurones artificiels.

```
%% Réseaux de neurones

% Réseau choisi
load('PLY_Pattern\net_ply_95_0_H40_12C_80space.mat');

% Reconnaissance
inputs = matinput; %1000000x150
for objet = 1:nb_objet
    output = net(inputs(:,objet));
    [maximum, classe] = max(output);
    pourcentage_net{objet,1} = maximum;
    if maximum < 0.01
        classe = 0;
        nom_objet{objet,1} = ' unrecognized';
        nom_objet_fr{objet,1} = ' non reconnu';
    else
        switch classe
            case 1
                nom_objet{objet,1} = ' voley ball';
                nom_objet_fr{objet,1} = ' ballon de voley';
            case 2
                nom_objet{objet,1} = ' iron cylinder';
                nom_objet_fr{objet,1} = ' cylindre de métal';
            case 3
                nom_objet{objet,1} = ' juice box';
                nom_objet_fr{objet,1} = ' boite de jus';
            case 4
                nom_objet{objet,1} = ' expresso cafe';
                nom_objet_fr{objet,1} = ' café expresso';
            case 5
                nom_objet{objet,1} = ' tennis ball';
                nom_objet_fr{objet,1} = ' balle de tennis';
            case 6
                nom_objet{objet,1} = ' blue Mentos';
                nom_objet_fr{objet,1} = ' mentos bleu';
            case 7
                nom_objet{objet,1} = ' glasses box';
                nom objet fr{objet,1} = ' boite a lunette';

            case 8
                nom_objet{objet,1} = ' xbox box';
                nom_objet_fr{objet,1} = ' boite de xbox';
            case 9
                nom_objet{objet,1} = ' tissue box';
                nom_objet_fr{objet,1} = ' boite de mouchoir';
            case 10
                nom_objet{objet,1} = ' yop';
                nom_objet_fr{objet,1} = ' yop';
            case 11
                nom_objet{objet,1} = ' red box';
                nom_objet_fr{objet,1} = ' boite rouge';
            case 12
                nom_objet{objet,1} = ' cookie box';
                nom_objet_fr{objet,1} = ' boite de biscuit';
            otherwise
                nom_objet{objet,1} = ' unrecognized';
                nom_objet_fr{objet,1} = ' non reconnu';
        end
    end
end
```

Figure 7.30 : Code de la reconnaissance d'objets.