POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in Ingegneria dell'Automazione

POLITECNICO
MILANO 1863

A POINT-AND-COMMAND INTERFACE FOR GRASPING UNKOWN OBJECTS WITH ROBOTIC MANIPULATORS

Relatore: Prof. Andrea Maria Zanchettin

Correlatore(i): Prof. Paolo Rocco

Tesi di laurea di:

João Marcos Correia Marques, Matricola: 848958

Anno accademico 2017/2018

ACKNOWLEDGEMENTS

First, and above all, I give thanks to God, without whose mercy I would no longer be.

I wish to thank Professor Andrea Zanchettin, for his patience, guidance and help were essential in the process of developing this project. Without his counsel and his expertise, the final demonstrations would not have been possible in the limited time available.

I extend my gratitude to Professor Paolo Rocco, who made this project a reality by allowing me to start the project way earlier than most – and for allowing my thesis presentation to be performed remotely.

Special thanks are also in order to Renzo Villa and Davide Nicolis, who've helped time and time again with the laboratory's infrastructure and overall debugging of my code.

Special thanks go to Idil Su Erdenlig for helping me out with the final demonstration videos.

My gratitude also goes out to Gerardo Malavena for helping me out with the Italian Abstract! Gera, you're the best!

Lastly, to my family, whose unwavering support provided me with the safest of havens in the hardest of times.

ABSTRACT

Automata are set to become ubiquitous in modern society, fulfilling assistive, collaborative industrial and domestic roles. In fulfilling these roles, the robots will come in contact with users of varied technical backgrounds, often having no special training for operating them. It is crucial, therefore, to provide novel ways in which humans may command them, lowering the barrier of entry to provide a safer, more intuitive and collaborative interactive environment. Pointing is one of the first ways humans interact with one another and throughout their lives it remains a very important element in human communication. As such, it is a natural candidate for being used as a means to interact with robots. Therefore, properly establishing an effective point-and-command human-machine interface comes as a natural milestone in the process of universalizing the application of automata. In interacting with robots, one of the crucial tasks to be performed is picking and placing objects. In domestic or uncontrolled industrial environments, the objects involved in these operations are often previously unknown. This work, thus, set out to provide a broad view on how human-robot interfaces have been done in the past, while proposing a different pipeline for commanding the grasping of unknown objects. This system proposes the use of an RGB-D sensor – the Microsoft Kinect – as a main visual sensor and by using its embedded skeleton tracking capabilities, is able to identify which object the user wants the robot to grasp (*i.e.* the object the user is pointing at) and autonomously generate a grasping pose for said object. The proposed system offers added robustness with respect to object traits, such as varying and non-uniform colors and different lighting conditions of the scene by applying an extension of Felzenszwalb and Huttenlocher’s graph-based image segmentation algorithm instead of traditional color tresholding or background removal techniques and by implementing a different kind of Point-Cloud filtering technique that allows it to reduce the effect of false-negative identifications in the image segmentation step.

ESTRATTO IN LINGUA ITALIANA

Gran parte delle previsioni indicano che i robot diventeranno onnipresenti nella società umana, integrati in tutte gli ambiti della nostra vita - domestici, professionali e personali. Questa onnipresenza, però, presenterà una grande sfida, mai affrontata per coloro che hanno il compito di progettare i robot, cioè il cambiamento radicale del profilo degli utilizzatori e del contesto nel quale il robot è adoperato.

Con la diffusione dei sistemi automatici, la classe degli utilizzatori, che inizialmente comprendevano soltanto operai specializzati da aziende, includerà elementi meno esperti, dal momento che non sarà possibile dedicare all'intera popolazione il tempo e le risorse che inizialmente venivano destinati agli operai.

Inoltre, in particolar modo per gli *assistive robot*, l'utilizzatore potrebbe non essere capace di interagire con il robot in modo convenzionale; questo potrebbe essere il caso delle persone disabili, impossibilitate nell'utilizzo, ad esempio, persino della tastiera con la quale avviare un *dispositivo vigilante*.

Di conseguenza, sia per questioni di sicurezza che di praticità, l'interfaccia uomo-macchina deve diventare necessariamente più semplice ed intuitiva. Inoltre anche nell'ambito industriale, in cui gli utilizzatori continueranno a possedere una formazione tecnica adeguata e le difficoltà associate alle persone con disabilità sono meno rilevanti, l'area dei robot sarà interessata da importanti cambiamenti.

Con l'avvento dell'*Industria 4.0*, vi è una crescente richiesta di un ambiente industriale che sia più dinamico e più flessibile, che permetta di modificare sia il prodotto finale, sia la disposizione delle macchine e degli operati all'interno dell'azienda. Per questi motivi, l'approccio tradizionale con il quale sono programmati i robot di oggi, caratterizzati da traiettorie fisse e *hard-coded* (cioè difficili da

modificare, dal momento che alcune volte è richiesta buona esperienza in diversi linguaggi di programmazione), presto non sarà più adeguata per grande parte delle nuove aziende.

Dato questo paronama, è evidente che vi sia la necessità di introdurre nuove tecniche per programmare gli automi futuri,e ci sono già numerose proposte che prevedono un'interfaccia uomo-computer che faciliterebbe questo incarico.

Alcune soluzioni (come, ad esempio, il caso di Matlab e Simulink) mirano a questo obiettivo introducendo elementi visuali nel linguaggio di programmazione, che prende il nome di *programmazione con blocchi*; queste alternative, sebbene siano utili ai fini dell'apprendimento della programmazione, sono a volte molto limitanti.

D'altro canto, vi sono anche altre alternative che propongono di utilizzare mezzi più intuitivi.

Sin dai primi anni della sua vita, l'essere umano è infatti abituato a interagire per mezzo di gesti, i quali rappresentano per l'uomo la tecnica più diffusa di comunicazione a breve distanza.

Per questo motivo un'interfaccia uomo-macchina che sia capace di capire e processare i comandi emessi per mezzo dei gesti sarebbe molto intuitiva e semplice da usare.

Infine, una tra le attività più basilari dei robot è quella di raccogliere e posizionare oggetti; dal momento che i nuovi robot saranno usati in ambienti e contesti sconosciuti, è conveniente sviluppare una Human Machine Interface (HMI) che sia robusta rispetto a variazione ambientali e delle caratteristiche degli oggetti.

Sulla scorta del discorso che è stato presentato finora, questa tesi si dedica a studiare e proporre un metodo alternativo alle tradizionali HMI basato sul paradigma

point-and-command per operazioni di prelievo e posizionamento di pezzi per manipolatori robotici.

Dopo aver effettuato un'ampia ricerca sullo stato dell'arte, è stata identificata un'importante mancanza nelle HMI proposte, cioè che esse fossero allo stesso tempo robuste a variazioni ambientali (a causa delle tecniche di rimozione di fondo usate) e a variazioni nelle caratteristiche degli oggetti (a causa delle tecniche di identificazione degli oggetti manipolati).

In riposta a questa mancanza, è stata proposta una nuova interfaccia che usa il paradigma *point-and-command*; essa utilizza una tecnica di segmentazione di immagini basata sulla teoria dei grafi e lo *skeletal tracking* per identificare l’obiettivo del manipolatore. I risultati ottenuti in questo modo sono sia robusti rispetto all'ambientazione, sia rispetto a molte caratteristiche degli oggetti, come ad esempio colore, riflettività e struttura.

Per la sua semplicità e robustezza, la pinza utilizzata è di natura a ventosa, e la strategia di generazione delle configurazioni di grasping è basata sulle euristiche ricavate dalla letteratura; esse consistono nell'afferrare l'oggetto nelle aree più piatte della sua superficie e quanto più vicino possibile al suo centro di massa.

Il risultato finale è stato quello di ottenere una HMI funzionale, comandata unicamente tramite gesti, capace di funzionare in ambienti generici e di permettere la manipolazione di oggetti di diversa natura; essa può essere utilizzata anche affiancata ad altri moduli di controllo per fornire un'esperienza più intuitiva e sicura per gli utilizzatori non esperti.

Infine, un ulteriore e importante contributo di questa tesi è stato quello di fornire un'estensione del paradigma *point-and-command* basato sulla segmentatzione di immagini che ne migliora la versatilità, e una tecnica innovativa per filtrare le nuvole di punti migliorandone la robustezza.

FIGURE LIST

EQUATION LIST

ACRONYM LIST

Acronym	**Meaning**
PbD	*Programming by Demonstration*
HRI	*Human-Robot Interface*
HMI	*Human-Machine Interface*
DCNN	*Deep Convolutional Neural Networks*
FH Algorithm	Felzenszwalb and Huttenlocher’s (FH) algorithm
PR	*Probabilistic Rand metric*
NPR	*Normalized Probabilistic Rand metric*
VI	*Variation of Information metric*
GCE	*Global Consistency Error metric*
BDE	Boundary Displacement Error

INDEX

1. Introduction & Thesis Outline

1.1. Introduction

In its September 2016 press release, the International Federation of Robotics (IFR) highlighted that the annual supply of industrial robots in the world was expected to grow up to 60% by 2019, as seen in Figure 1. This large growth reflects the growing adoption of robotics and overall automation of industrial processes. This growth, however, is not directly linked with the traditional model of automation. As noted in [1], the current and future industrial environment differs from the past ones in many factors, including the need for increasingly more versatile manufacturing plants, increased interaction with non-specialized workers in the same work environment [2] and, consequently, increased complexity of tasks to be performed by the machines under a constantly shifting workspace. One example of this shift is the increase in research [3]–[5] and market availability of collaborative robots, such as ABB's IRB 14000 YuMi, Rethink Robotics' Baxter, Fanuc's CR-35iA, KUKA's LBR iiwa and Universal Robot's UR3 UR5 and UR10. These robots are designed to be used safely in the same environment as humans without requiring usual safety features such as cages and no-go areas, allowing for deeper interaction between workers and machines.

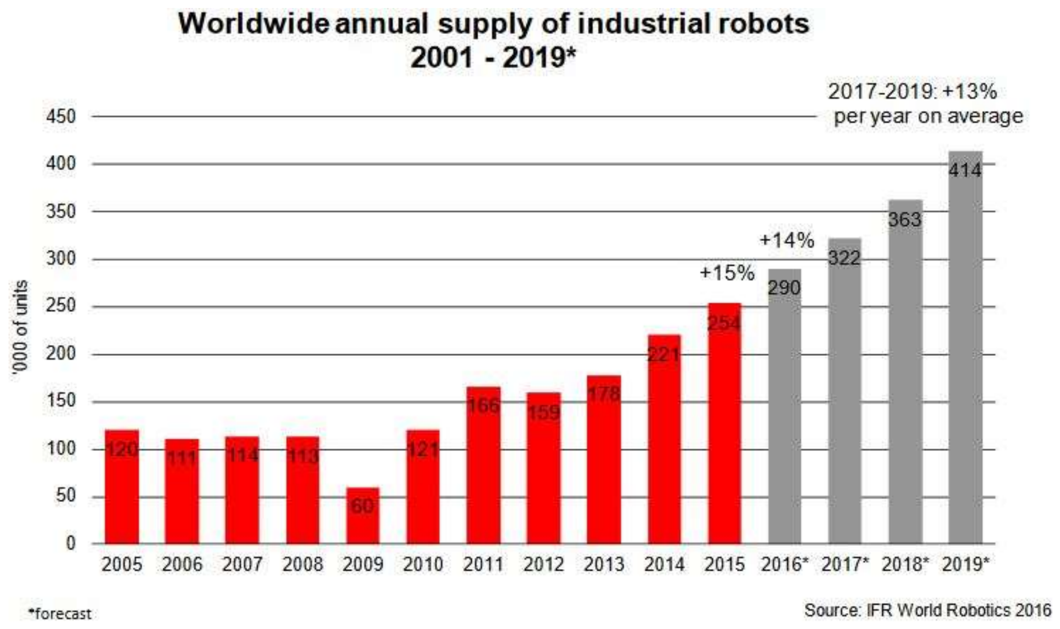


Figure 1 - Worldwide annual supply of industrial robots [1]

Parallel to the Industry 4.0 movement, another important paradigm shift has occurred, in which the focus of robotics has been shifting from strictly specific industrial settings to more consumer, home and service-oriented environments, as noted by [6] and [7]. Under this new framework, an automaton is forced to interact with a highly disorganized environment, while being operated by a wide variety of users, which are likely not to have access to specialized training on how to operate such robots or, in the case of assistive robotics, may not have physical conditions to operate them the usual way – that is – through robot-specific programming languages.

When considering these scenarios for the future of robotics and automation, a flexibilization of the way robots are programmed is due in order to both lower the barrier of entry to their usage and speed up reprogramming industrial robots in a dynamic environment, as the rigid way conventional programming methods work (by requiring the user to learn syntax and semantics of a proprietary robot programming language and exhaustively debugging the program) makes changing the robot's task slow and difficult, even for trained operators, as highlighted in [8] and [9]. Moreover, being able to freely and easily interact with the machine has been reported in literature as one of the crucial factors in determining the successful incorporation of robots into ordinary people's lives [10], [11].

Many paradigms for human-robot interaction (HRI) have been proposed over the years trying to facilitate and make it more intuitive. In their survey of Robot Programming Systems [9], the authors separate those efforts in two fronts: Those of manual and automated programming. For them, manual programming systems are those that require the user to directly specify the robot's actions in its code, while automatic programming allows users to specify the robot's actions without ever directly changing the machine's code. Relevant efforts in simplifying manual programming systems listed are generic procedural languages – such as Motion Description Language, for Java-; behavior based languages – like Functional Reactive Programming; Graphic Programming Interfaces - like that of Lego Mindstorms or the one presented in [12] as efforts in improving manual programming.

On the automatic front, learning systems like that of [13] are able to perform complex actions by joining together simply-defined ones using a conditioning protocol, whereas [14] uses *kinesthetic teach-in* to teach simple movements to the robot and use reinforcement learning to later allow the robot to perform more complex tasks by joining them together. [15] instructed a six legged robot on how to walk by performing Q-learning into each of the legs to learn optimal policies of movement given the goal to either swing or stand. Finally, [16] and [17] present an extensive survey on the use of reinforcement learning in robotics and which I shall not detail as they are out of the scope of this work.

Another approach to automated programming is programming by demonstration (PbD). In this paradigm, the user guides the robot through a teach-pendant, like the one in Figure 2 - which allows for the direct control of the robot’s joint positions or of the robot’s end-effector’s 6 degrees of freedom- to guide the robot to its desired positions, recording those point’s coordinates at each step, which are later replicated by the robot. Some robots, collaborative robots in particular, allow the robot’s joints or end-effector to be manually moved to the set-points This facilitates the programming steps, but still requires training on how to operate the teach-pendant and basic knowledge of the robot-specific language, as some semantics of the programming language still need to be applied in order to specify the kind of motion to be performed between the registered spatial points – *e.g.* linear or circular motion and specifying the speed of motion.

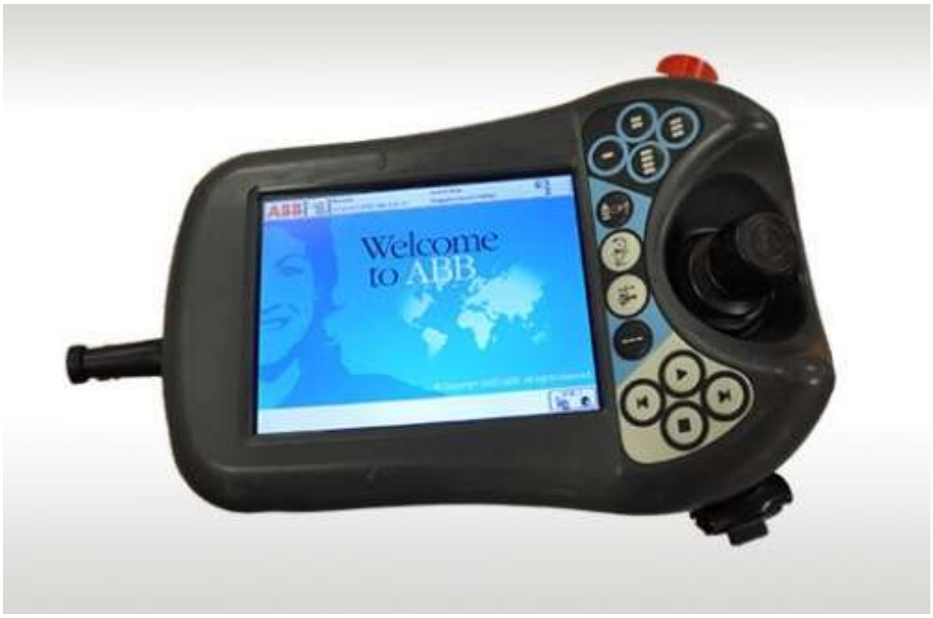


Figure 2- ABB teach-pendant [18]

In addition to those methods, some additional paradigms have been presented, extending the concept of PbD by allowing for the use of multi-modal systems in the process of teaching. Teach-by-showing, for instance, allows automata to infer motion patterns from observing human subjects performing the goal task. For instance, in [8], a robot received the force, torque and position inputs of a human demonstrating the goal activity of inserting a PC cart into a motherboard and used those to generalize subtasks and tasks and then perform those tasks by itself; [19] used a similar paradigm to teach a robot how to perform the ball-in-a-cup task and [20] used imitation learning associated with reinforcement learning to train a humanoid robot to grasp items in front of it, proving the potential of this strategy.

Finally, another programming paradigm is that of instructive systems. In this case, the robot already possesses a set of well-defined tasks and operations on which he has already been instructed – by means such as manual coding or any of the previously mentioned programming paradigms – and executes those tasks on demand based on human input, usually through gestures and voice recognition [9]. Examples of the application are found in [21] , which provided a system to allow for commanding a fetching robot using gestures and [22] provided a framework to identify complex gestural commands to be interpreted by the robot.

On a highly interactive work environment, this paradigm, which represents a very high-level kind of programming, is expected to be the most useful, as many of the industrial tasks to be performed are standard, but their targets, their order or their frequency are not well defined or fixed. Thus, allowing the users of the automata to freely assign it a specific task on demand by voice commands or gestures should provide the flexible workspace cooperation that is demanded from a modern industrial setting or the ease of use necessary for the application of assistive robotics in a domestic environment.

Considering the above-mentioned arguments, this work will focus on the study and development of a simple instructive system to command an industrial robot through a known set of gestures, captured by a Kinect V2.0, focusing on the task of identifying and grasping unknown objects within the fixed robot’s field of view, which will start with an bibliographical overview of each of the isolated components of the project.

1.2. Objectives and Chapter Structure

1.2.1. Objectives

This work's ultimate goal is to create a framework that can be applied in the creation of an instructive system that will allow users to command robots to perform grasping tasks using their bodies. In order to demonstrate this framework in action, a simple demonstration is designed, in which a simple instructive system that allows users to command a robotic manipulator that is fixed in space to pick completely unknown objects in unknown positions by using gestures captured by a Kinect V2.0 system, also fixed in space. Considering a modular approach to robot control, one may imagine it as shown in Figure 3. In this model, the user conveys his intention to the robot by whatever means (in the context of this work, it would be through gestures). The Human Robot Interface (HRI) then interprets this intention and turns it into a set of high level instructions to be followed by the robot, such as "place your end-effector at position pose P_1 coming from pose P_2". This set of high level instructions is fed into the Robot Control Module, which will compute the set of low-level instructions to be given to the robot, such as the intermediate steps between the robot's current position and the starting point P_1 and the joint positions that the robot should assume in order to reach those poses, while also checking for the feasibility of those instructions and notifying the HRI when the instruction is unfeasible (out of reach or in a singular position or would cause the robot to collide with itself or the environment).

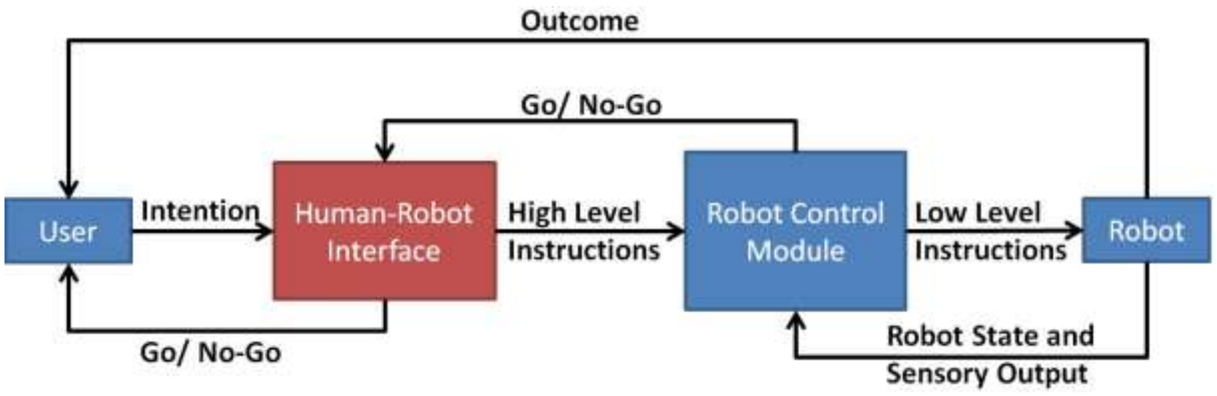


Figure 3 -Simplified Robot Control Workflow

While much research exists in the area of robot control, focusing on avoiding singularities, solving kinematic redundancies and avoiding collisions, often achieving satisfactory great run-time performance, the field of development of HRIs still needs much work. Thus, this work will focus in the red module in Figure 3, while most of the already available modules for robotic control, such as collision detection, singularity avoidance and redundancy resolution being either bypassed completely or done in the simplest way possible- via hardcoding of solutions- in order to focus on the main goal in the limited time available for the development of the project.

1.2.2. Chapter Structure

In the framework for the creation of the HRI, three key aspects must be taken into account, namely:

- Finding the most intuitive ways to interact with the robot;
- Establishing the robot’s task given the command;
- Generating the high-level description of the task to be fed into the robot controller.

In this work, each of these aspects will be dealt with in a more theoretical approach Section 2, Bibliographical Review. Section 3 will detail the proposed system regardless of implementation, while the implementation details for the demonstration will be detailed in Section 3.2. Experimental Results will be displayed in Section 4 and Section 5 will present the closing remarks on the project and suggestions on how to improve upon this work.

2. Bibliographic Review

An instructive system is an integrated software solution, which involves many software components sewn together with the aim of allowing the final user to intuitively interact with the automaton. This interaction is done through multimodal systems of interaction, which provide a natural, human-like way through which commands are issued. The current state of the art for multi-modal HRI is described in detail in this section. In addition, instructive systems also rely on the existence of high-level functional blocks, which represent autonomous routines that implement each of the robot's high-level commands. Some of these blocks, with special attention to autonomous grasping, are also presented in depth in this section.

2.1. Human Robot Interaction

In his survey on HRI, [23] defines the field as the search for ways to "*understand and shape the interactions betweenone or more humans and one or more robots*". In that context, the authors highlight five key attributes that define and affect the interactions between user and robot. These are: Autonomy; Nature of Information exchange; Structure of the Team; Adaptation, learning and training of people and the robot and Shape of the task. Each of those attributes is briefly discussed in the following section.

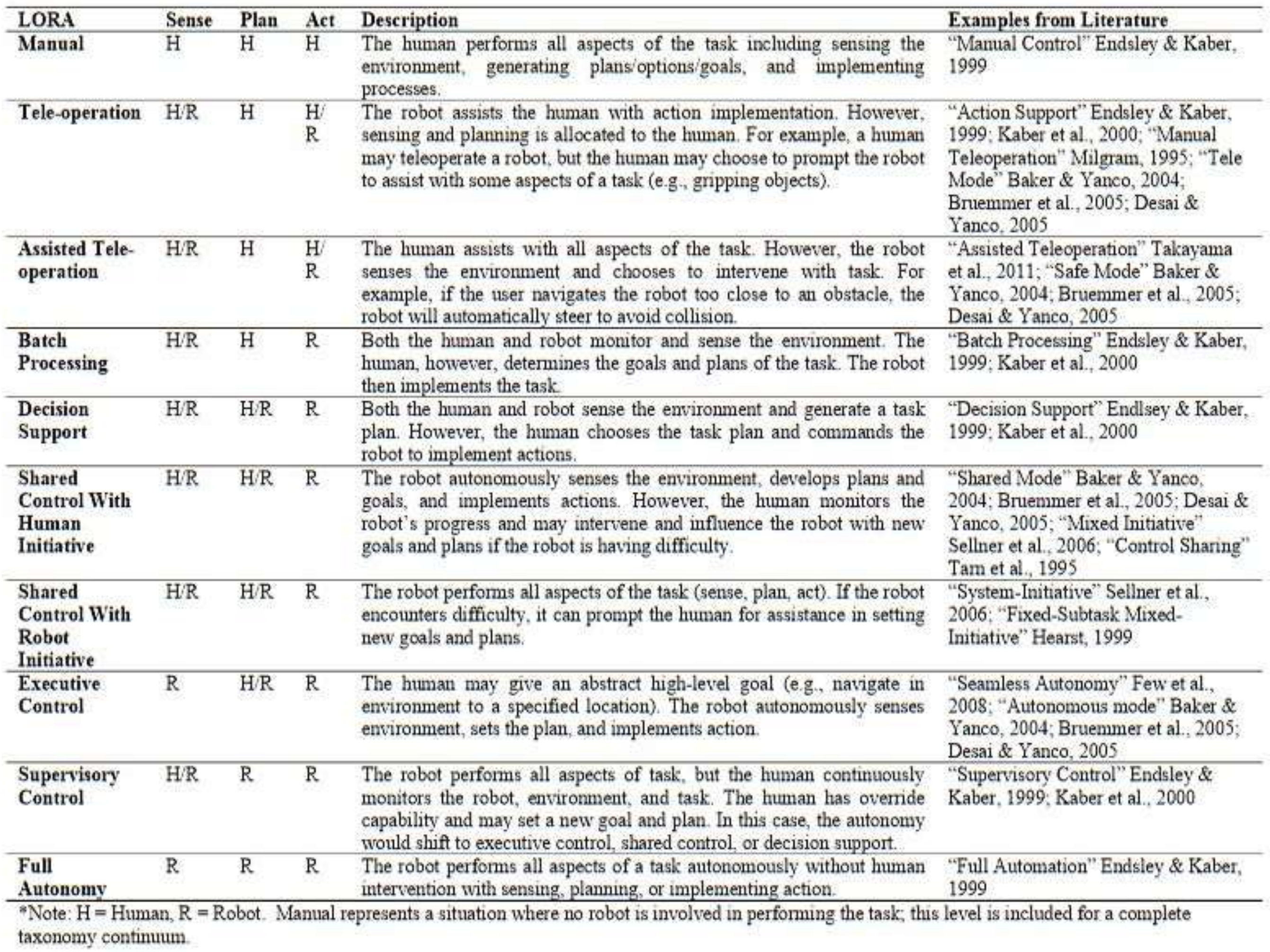
2.1.1. Autonomy

Autonomy is a still much debated concept in robotics, as literature has not yet settled on a single definition that is appropiate for all contexts. Indeed, [24] presents a summary of such definitions, reproduced in Table 1

Table 1- Definitions of Agent and Robot Autonomy [24]

Definitions of Agent and Robot Autonomy
“The robot should be able to carry out its actions and to refine or modify the task and its own behavior according to the current goal and execution context of its task.” [25]
“Autonomy refers to systems capable of operating in the real-world environment without any form of external control for extended periods of time” [26]
“An autonomous agent is a system situated within and a part of an environment that sense that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future;” “Exercises control over its own actions.” [27]
“An Unmanned System’s own ability of sensing, perceiving, analyzing, communicating, planning, decision-making, and acting, to achieve goals as assigned by its human operator(s) through designed HRI ... The condition or quality of being self-governing.” [28]
“‘Function autonomously’ indicates that the robot can operate, self-contained, under all reasonable conditions without requiring recourse to a human operator. Autonomy means that a robot can adapt to change in its environment (the lights get turned off) or itself (a part breaks) and continue to reach a goal.” [29]
“A rational agent should be autonomous—it should learn what it can to compensate for partial or incorrect prior knowledge.” [30]
“Autonomy refers to a robot’s ability to accommodate variations in its environment. Different robots exhibit different degrees of autonomy; the degree of autonomy is often measured by relating the degree at which the environment can be varied to the mean time between failures, and other factors indicative of robot performance.” [31]
“Autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal states.” [32]

Although a single definition for autonomy has yet to be achieved, [24] propose several guidelines for categorizing it, most notably the one presented in Figure 4, where the author classifies and defines 10 Levels Of Robotic Autonomy (LORA) accoding to the roles taken by the user and the robot in the normal operation of the automaton, providing as well a brief description on how each LORA could be characterized and a list of few examples of those LORAs that were available in literature at the time.



LORA	Sense	Plan	Act	Description	Examples from Literature
Manual	H	H	H	The human performs all aspects of the task including sensing the environment, generating plans/options/goals, and implementing processes.	"Manual Control" Endsley & Kaber, 1999
Tele-operation	H/R	H	H/R	The robot assists the human with action implementation. However, sensing and planning is allocated to the human. For example, a human may teleoperate a robot, but the human may choose to prompt the robot to assist with some aspects of a task (e.g., gripping objects).	"Action Support" Endsley & Kaber, 1999; Kaber et al., 2000; "Manual Teleoperation" Milgram, 1995; "Tele Mode" Baker & Yanco, 2004; Bruemmer et al., 2005; Desai & Yanco, 2005
Assisted Tele-operation	H/R	H	H/R	The human assists with all aspects of the task. However, the robot senses the environment and chooses to intervene with task. For example, if the user navigates the robot too close to an obstacle, the robot will automatically steer to avoid collision.	"Assisted Teleoperation" Takayama et al., 2011; "Safe Mode" Baker & Yanco, 2004; Bruemmer et al., 2005; Desai & Yanco, 2005
Batch Processing	H/R	H	R	Both the human and robot monitor and sense the environment. The human, however, determines the goals and plans of the task. The robot then implements the task.	"Batch Processing" Endsley & Kaber, 1999; Kaber et al., 2000
Decision Support	H/R	H/R	R	Both the human and robot sense the environment and generate a task plan. However, the human chooses the task plan and commands the robot to implement actions.	"Decision Support" Endlsey & Kaber, 1999; Kaber et al., 2000
Shared Control With Human Initiative	H/R	H/R	R	The robot autonomously senses the environment, develops plans and goals, and implements actions. However, the human monitors the robot's progress and may intervene and influence the robot with new goals and plans if the robot is having difficulty.	"Shared Mode" Baker & Yanco, 2004; Bruemmer et al., 2005; Desai & Yanco, 2005; "Mixed Initiative" Sellner et al., 2006; "Control Sharing" Tam et al., 1995
Shared Control With Robot Initiative	H/R	H/R	R	The robot performs all aspects of the task (sense, plan, act). If the robot encounters difficulty, it can prompt the human for assistance in setting new goals and plans.	"System-Initiative" Sellner et al., 2006; "Fixed-Subtask Mixed-Initiative" Hearst, 1999
Executive Control	R	H/R	R	The human may give an abstract high-level goal (e.g., navigate in environment to a specified location). The robot autonomously senses environment, sets the plan, and implements action.	"Seamless Autonomy" Few et al., 2008; "Autonomous mode" Baker & Yanco, 2004; Bruemmer et al., 2005; Desai & Yanco, 2005
Supervisory Control	H/R	R	R	The robot performs all aspects of task, but the human continuously monitors the robot, environment, and task. The human has override capability and may set a new goal and plan. In this case, the autonomy would shift to executive control, shared control, or decision support.	"Supervisory Control" Endsley & Kaber, 1999; Kaber et al., 2000
Full Autonomy	R	R	R	The robot performs all aspects of a task autonomously without human intervention with sensing, planning, or implementing action.	"Full Automation" Endsley & Kaber, 1999

*Note: H = Human, R = Robot. Manual represents a situation where no robot is involved in performing the task; this level is included for a complete taxonomy continuum.

Figure 4 - Taxonomy for robot autonomy classification[24]

These definitions account for several layers of autonomy and automatization of the process, allowing for a fine classification of robotic systems.

2.1.2. Information Exchange

As defined by [23], the way in which robots and users exchange information is also crucial in the proper analysis of a HRI. The media through which this information is conveyed between them varies from visual displays presented as a Graphical User Interface (GUI) or Augmented Reality (AR) interfaces; gestures; speech and natural language; non-speech audio; and physical interaction and haptics. Though formerly employed mostly in a separate way, these interactive media have been facing an increasing pressure to be used in conjunction to one another, with crescent interest for multimodal interfaces [22], [33]–[35] , as they are deemed more intuitive and efficient in conveying commands than traditional separate methods.

2.1.3. Team Structure

Another crucial feature in an HRI analysis is defining who will be taking part of the interactions. Indeed, teams of humans and robots have different compositions when applied to different fields and each of these compositions requires different interaction protocols. For instance, a team composed solely of robots may use the fastest communication means available, whereas teams which contain one human and several robots have to abide by a slower communication protocol to allow human oversight – and teams with more than one human are subject to even slower communication constraints, to allow for human understanding and consensus.

2.1.4. Learning and Training

As remarked by [23], one of the objectives of a good HRI design is to minimize the training time for both users and robots without compromising the system's operability. This may be done through increasing the robot's learning capabilities or by making the interfacing intuitive to humans.

2.1.5. Task-Shaping

Another crucial factor in the design of HRIs is the attention to how the introduction of the robot to the task will affect how the human who is interacting with him will change the way in which the task is done. Indeed, when designing an HRI the designer should ponder whether he may improve the process he is working on, rather than just replacing to some degree the human effort that was previously needed. Robots often do not have the same restrictions that humans do (such as temperature, orientation, positioning) and- as such- create new possible ways to perform a task, often much more efficiently.

That being said, we may conclude that the design of an HRI consists in finding a way for humans and robots to collaborate, while minding their environment, their interaction media, the shaping of the task at hand and the composition of the teams that are working on it. This process of collaboration relies heavily on the media through which we interact with those robots and thus, if we are to improve it, we need to provide the users with intuitive interactive interfaces. As noted by [36], humans naturally interact with the world using multiple resources simultaneously. It should be, therefore, simpler for them to interact with machines in a similar manner. Hence, the next session will detail the many scientific initiatives in providing multimodal HRIs.

2.2. Multimodal Human-Robot Interaction

Vision, hearing, touch, olfaction and gustation have long been considered the five basic senses, the gateways used by the human brain to communicate with the environment around it. Even though this paradigm has been facing scientific scrutiny over the past few years, as proprioception and balance are now often included in the list of senses, these 5 remain the main media of conscious human interaction with its surroundings.

Taste is mostly neglected as valid means of human-machine interaction due to its intrinsic contact-based chemical nature. Olfaction has seen some innovation with growing development of electronic-noses, applied in several fields , such as chemical plants monitoring [37] and even object recognition [38], with varying degrees of success. Its use in the field of robotics is, however, still fairly restricted and thus left out of this survey.

With recent advances in haptics, touch is seeing an increase in usage as a form to enrich Human-Computer-Interaction (HCI), both in tele-operated robots, such as [39] and in virtual reality environments [40]. Though this dimension of multimodality shows promise in enriching user experiences in simulation systems, improving accuracy in tele-operated tasks, improving safety in human-robot collaborative spaces (with the implementation of tactile kill switches in industrial robots, often implemented in collaborative robots) and improving teaching by demonstration robot programming [41], it is not further detailed in this thesis for the sake of brevity, with the reader being referred to [42] for a broad survey of advances in the field, their methods and technologies, though a short detailing of force-feedback robotic programming is presented in the following section, as it was necessary for the implementation of suction-based grasping on a collaborative robot.

Considering the factors exposed above, most of the research pertaining to the field of multimodal HRI concerns the use of visual or auditory information to mediate the exchange of information between users and systems, being used in concurrency to improve the effectiveness of the process according to Wicken’s multiple resources theory [43]. An overview of the forms of unimodal audiovisual HRI paradigms will thus be provided first, alongside a brief overview of how they are being combined in the current research landscape.

2.2.1. Vision-Based Human Robot Interaction

By adding a visual input to a robotic system, many ways for interacting with them arise. In particular, visual systems that possess embedded depth sensing, such as the Microsoft Kinect® allow for better tracking of human intention. All forms of visual-based HRIs can be seen as special cases of object detection and tracking. In their survey on object tracking, [44] provide an ample analysis of the field which is, alongside [45] the basis for this section. Visual-Based HRIs can be based in four main techniques, all subfields of object tracking: Blob Detection, Background Removal, Skeletal Tracking and Gaze Tracking, explained in further detail below.

2.2.1.1. Blob Detection

As defined by [45], a blob is :

"…a region inside which the pixels are considered to be similar to each other, meanwhile be different from the surrounding neighborhoods."

As such, they remark that blob detection methods rely mostly in identifying interest points and interest regions, with interest points being classified as extrema in scale-location spaces denoting regular regions, whereas interest regions being classified as irregular segmented regions that are considered "constant" in a given metric.

Classic interest point detection algorithms, such as Laplacian of Gaussian (LoG), Difference of Gaussian (DoG) and Hessian-Laplacian, rely on the construction of Gaussian pyramids in order to detect points of interest. LoG has a high computation cost due to the computation of second derivatives involved in its application and has in DoG an approximation of lower computational cost.. In both cases, the extrema of both pyramids are recorded as being LoG ad DoG points of interest, respectively. Another alternative method to locate points of interest is based on the Determinant of Hessian, in which the scale-normalized determinant of the Hessian matrix,$\sigma^4 \det(H)$, is used to detect interest points.

The interest points indicated through these metrics can then be applied in many algorithms to detect key points in an image, such as Scale Invariant Feature Transform (SIFT, used in object recognition, visual tracking and baseline matching[45], [46]). Some methods, such as Speeded-Up Robust Feature (SURF, which uses box filters approximations to generate efficient Hessian interest point detection [45], [47]), try to generate these points in a more computationally efficient way, while more modern methods, such as KAZE features [48] avoid Gaussian smoothing-induced blurring of the images by making use of nonlinear diffusion filtering techniques, preserving natural image boundaries though at a higher computational cost [45].

Another important component of blob detection is interest region detection. An interest region can be defined as a “region segmented from neighboring areas by exploiting the constancy of image properties” [45], like pixel intensities. One of the algorithms used to obtain interest regions is Maximally Stable Extremal Region (MSER) [49]. MSER is based on thresholding pixel intensities with several different thresholds and finding regions that remain stable over the largest number of different thresholds. Though not free from questioning [50], MSER is still used in many fields due to its simplicity. Other commonly used algorithms include Edge-Based Region (EBR) and Intensity Extrema-based Region (IBR) [51].

Regardless of the algorithms used, blob detection is usually used as an initial step in more complex image processing algorithms or for applications that do not require finely detailed knowledge of the objects surrounding the actor in the environment, such as obstacle avoidance [52].

2.2.1.2. Background Removal

Commonly used in the pre-processing steps of many complex computer vision algorithms, background removal can take many forms. If both the robot and the camera are fixed in space – and most of the robot’s workspace remains unaltered during its operation, one may resort to the technique applied in [53], in which the authors took an image of the workspace before the introduction of the robot and the human, which they labelled as reference image, which they edge-filtered using Laplacian Filters. Then, when it came time to detect the relevant objects in the scene, the authors took another picture, now with all the relevant elements within the camera’s field of view. After also edge-filtering the new image, they proceeded to take the difference between the two images and, through thresholding, determine whether an interesting object is located in a given position within the frame, *i.e.,* the authors used the difference between images to determine what changed between frames and, knowing the reference frame was of an empty room, conclude that whatever changed must be an object. The authors later perform many post-processing steps to identify which objects were modelled- like the robot- and which were invading its workspace, like the human, and later used the labelled images to calculate the distance between them, enabling the implementation of a safety stopping monitoring mechanism. Though the implementation details are beyond the scope of this work, their processing pipeline is exposed in Figure 5.

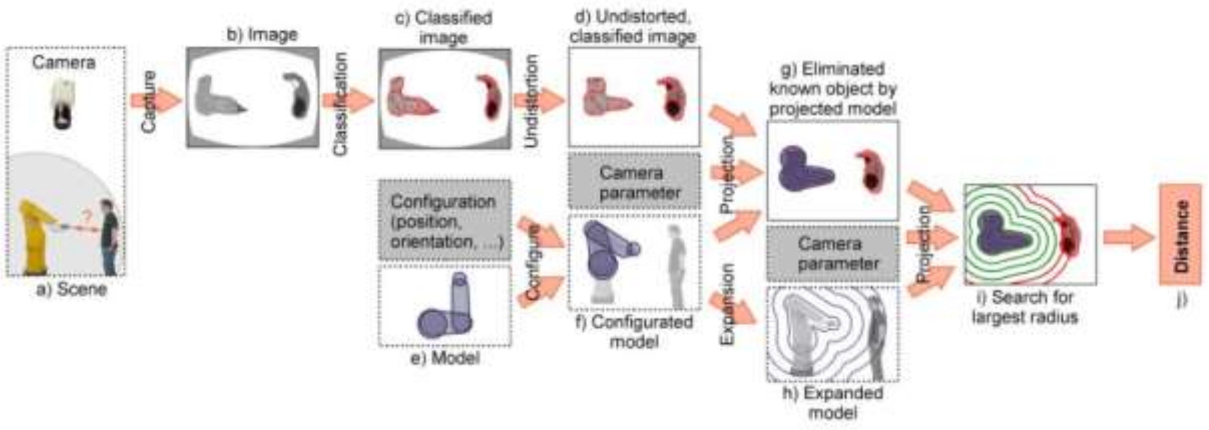


Figure 5 - [53]'s pipeline for vision-based distance calculation

Though a rather complete survey of background removal techniques may be found in [54], [55], two other techniques are of interest within the scope of this project. One of them is presented in [56], presented here for the novelty of using traditional clustering technique K-means as a pre-processing background removal technique. The other one is presented in [57], which made use of Microsoft Kinect®'s depth camera to segment the image into foreground and background by means of auto thresholding, which is based on finding the valley following the first large peak in an image's depth histogram.

Both these techniques allow for fast and slightly more detailed object identification, even though several more pre-processing steps are needed to render this new information useful in most settings.

2.2.1.3. Skeletal Tracking

In many interactive settings, the most intuitive way of interacting with robots rely on pointing [21] or at least identifying the pose of the humans around the automata. In order to do so, skeletal tracking is often used. Skeletal tracking is a tracking technique in which the human body is represented by a number of 3D points, called joints, each associated with a part of the human body. Initial models that applied this technique had low reliability due to the fact that many of them relied on specific models of human anatomy not being robust, therefore, to changes in height, weight and orientation. However, with the advent of the Microsoft Kinect®, an effective and robust skeletal tracking device was now available and its use boosted the research on HRIs that relied in this technology [58].

In the Kinect, the skeletal tracking is done by treating the segmentation of the Kinect’s depth image as a pixel-per-pixel classification task. In it, a deep randomized decision forest classifier is trained using a large motion-capture database and run on a GPU to accelerate execution of the classifier. Once the depth image has been classified by body parts, a global centroid of probability mass is found through mean shift. Finally, the hypothesized joints are mapped to their corresponding skeletal joints and adjustments are made to the skeletal model to fit the observed proportions, taking into account temporal continuity of the depth image [58].

Indeed, many projects make use of the Kinect the main visual robotic interface. Some apply it to recognizing fine human gestures [57], [59], whereas others apply it as a means to identify pointing gestures [21], [60], and others even apply it to detect falls in the context of geriatric care [61]. There is still, however, active research in finding alternatives with better performance than the Kinect, notably in hand tracking [62].

2.2.1.4. Gaze Tracking

Though skeletal tracking is a very intuitive way to detect human intention, there are settings in which the large scale of skeletal movements are prohibitive (like when driving a car [63]) and some others where the camera’s field of view is too small to capture the entire body, and, most importantly, those where the user is disabled and, thus, unable to perform large scale motion, like in cases where the user suffers from tetraplegia or advanced Amyotrophic Lateral Sclerosis (ALS). In those instances, gaze tracking is one of the go-to methods to identify human attention and intention. Gaze is defined as the direction to which the eyes are pointing in space, and is widely regarded as one of the prime ways of indicating human focus [33].

There exist three main approaches to gaze tracking: Analytical vision-based, Machine-learning vision-based and intrusive methods [64], [65]. Intrusive methods have fallen out of use in recent years due to their cumbersomeness [66] and will not be detailed any further.

In the realm of analytical gaze tracking, three main algorithms are presented: One-Point Calibration (OPC) [67], Longest Line Scanning (LLS) and Occluded Circular Edge Matching (OCEM) [68].

OPC relies on a geometric eyeball model by the same author to perform the tracking, while using the single calibration point to determine the differences between the eye being observed and the model, with the distance between the eye and the camera being determined by the camera's auto-focus and the distance between two infrared lights arranged near the camera [67]

LLS relies on the facts that the projection of the iris unto a picture is an ellipse and that the center of an ellipse lies on the center of the longest horizontal line within the ellipse. Thus, the algorithm looks to the midpoint of said line to determine the center of the iris. This method by itself, however, is not sufficient for measuring eye-gaze precisely due to intra-iris noise, rough iris edges in the image and occlusion of longest line by eyelids [68].

In order to address these weaknesses in LLS, OCEM was developed. In general lines, due to the low eccentricity of the ellipse and the small angle of rotation of the eyeball, the iris may be approximated with a circle, which then has it's center and radius inferred based on a pixel-matching score, with more details being available in [68].

As for machine-learning based methods, a wide survey of these techniques for gaze tracking is presented in [65], with the reader being directed to their work for more details on the subject, as there are many approaches with varying degrees of success, with notable success in deep learning based approaches [69].

Regardless of the method used for tracking the gaze, many notable projects make use of this technology in order to improve user experience when interacting with robots, be it by providing a driver monitoring system [63], providing easy access to computers for disabled people [70] or even improving user experience in glasses-free 3D devices [71].

2.2.2. Voice Based Human Robot Interaction

As is the case with vision-based HRIs, voice-based ones figure as a very intuitive way to issue commands, since it is one of the primary means of interactions with fellow humans in daily life. This approach, however, presents many challenges, including translating the voice sample to text, so the robot can analyze the command.

In the realm of speech-to-text transcription, the state of the art involves Context-Dependent Deep Neural Network Hidden Markov Models (CD-DNN-HMMs), whose complexity is beyond the scope of this project, with the reader being directed to the seminal paper for further details. [72]. Nowadays, however, many off-the-shelf speech-to-text transcriptors exist, such as Google’s speech API, Amazon’s Alexa and the Kinect’s Speech Recognition modules, making it easier for robotic systems to integrate speech into their projects without the concern of developing the transcriptor.

Once the text has been obtained from speech, one must then establish the syntax which shall be used with the robot. Simpler applications, such as [73] allow for the designer to create a simple, rule-based syntax, comprised of a small set of commands which are characterized by a specific sequence of words appearing only in a set order. For more complex systems, such as chatbots, Deep Neural Networks and Recurrent Neural Networks tend to be used in order to determine the robot's response, allowing it to have a broader range of dialogue options and to understand a wider range of commands [74], [75].

Speech is often used to complement visual inputs to the system, as is the case in [21], as it allows for deeper interaction with the robotic systems.

2.3. Autonomous Grasping

In this thesis, a two-fold problem is addressed: that of providing a vision-based HRI for interacting with a robot- *i.e.* – commanding the robot to pick up unknown objects- and providing the robot with a routine that will allow it to autonomously grasp unknown objects in unknown environments. The state of the art in HRIs was presented in chapter 2.2, while the state of the art for the second task is discussed in the current chapter.

Autonomous grasp synthesis can be divided into four main categories: Model Based, Recognition based, On-line based grasp synthesis [76] and machine learning based [77], detailed in the following sections.

2.3.1. Model Based Grasp Synthesis

If all the objects that are to be manipulated possess a detailed 3D CAD model, precise geometric and dynamic analysis may be performed in those models in order to find the optimal grasp pose for each of the objects. One popular tool used in the process of finding the optimal grasping pose is "GraspIt!" [78] , a simulator specialized in grasp synthesis and analysis. In order to generate optimal grasping poses, "GraspIt!" performs a decomposition of the object into its basic, primitive shapes in order to easily calculate the approach directions [76]. Then, with the model of the gripper being positioned along those approach directions, collision and contact detection are performed in order to determine the optimal grasping pose [76]. Refinements to this method have been proposed, especially with regards to the feasibility of the proposed poses with respect to the environment around the robot, such as with the introduction of global accessibility [79], which considers the robot's kinematic restrictions, as well as environmental constraints, such as obstacles, in the evaluation of any given grasp solution and [80], whose decomposition trees help reduce the searching space for optimal grasping points, speeding up the selection process.

As [76] highlights, this approach is often used in an industrial setting which used to be- prior to the Industry 4.0 push – highly controlled environments, with repeatable tasks and identical manipulated objects. This approach, however, is not suited to be applied in a modern industrial – or even domestic- setting, as the variety of objects and the constantly shifting environments make it impractical to keep a log of all the objects in the robot's environment.

2.3.2. Recognition Based Grasp Synthesis

Another approach consists of building a database composed of a list of known objects – and their respective 3D models- and their optimal grasping poses computed in advance using any of the model-based grasp synthesis approaches. Then, at run-time, this approach is responsible only for identifying the object to be manipulated at the time with one contained with the database and adapting the grasping pose to the orientation and position of the identified object.

In their implementation of this approach, [81] built a database of object models and compiled their optimal grasping poses using "GraspIt!". Then, at run-time, the authors proceeded to identify the objects to be manipulated at the moment through the use of Scale Invariant Feature Transform algorithm, as described in [82]. The authors also allowed their system to be manually updated with additional information, such as preferred grasping positions or no-go zones. In another example, [83] implemented a similar system, in which the object was identified through the Nearest Neighbor Algorithm. Their systems defining feature, however, was that it updated its database after each grasping attempt, keeping a log of which grasping poses were the most stable for which object.

As [76] points out, recognition based techniques have found wide use in academia and industry, as their implementation often requires low effort and computational power and there are a fair number of open-source object model databases and object recognition frameworks available. One may not ignore, however, the biggest downfalls of these analytical techniques, as they fail to generalize grasping poses for objects outside their knowledge base and could potentially perform wrong object recognition, leading to potentially disastrous failures at execution time. This approach, as with the model-based ones, is not suited for the application described in this work.

2.3.3. Machine Learning-based Grasp Synthesis

With advances in computing power and machine learning techniques, their use in a wide variety of fields has seen a sharp increase in recent years, with grasp synthesis being no exception to it. Of most notoriety at present is U.C. Berkley's Dex-Net [77], [84]. Dex-Net implements a cloud-computing machine learning based grasp synthesis method. In a sense, [84] still draws from the ideas of a recognition based approach, but does not rely on the objects being the same as those in the database. Instead, using Multi-View Convolutional Neural Networks [85], the authors constructed a similarity metric between different objects by embedding their 3D properties into a latent space and calculating the Euclidean distance between them as a metric of their similarity. Once a list of most similar objects has been produced, their grasps are obtained from the database and then, using a Bayesian Multi-Armed Bandit (MAB) model with correlated rewards [86]–[88] in order to estimate which of them has the highest chance of success – and optimize the Gaussian process to find the grasping pose with the highest chance of success.

While this approach has seen impressive results, of up to 99% accuracy in grasping known objects [77] either with a suction cup or with a parallel jaws gripper, while being robust to sensing noise and uncertainty, it requires a special infrastructure to be applied, and thus was deemed not suitable for the specific application presented in this work, as it would require either constant internet connectivity or an unrealistic amount of embedded computational power and memory.

2.3.4. On-Line Based Grasp Synthesis

In stark contrast to the other methods presented thus far, on-line techniques do not demand any prior knowledge of the objects to be manipulated or any databases of 3D models and grasping positions. Indeed, in this method, object models are reconstructed on-the-fly based on the sensory input from the robot at the time of the grasping task and the grasp synthesis is performed using this model. As noted by [76], the reconstruction of object models is critical phase in this methodology, since sensors are subject noise, uncertainty and partial information and any grasp estimation's quality is directly related to the quality of the model it receives.

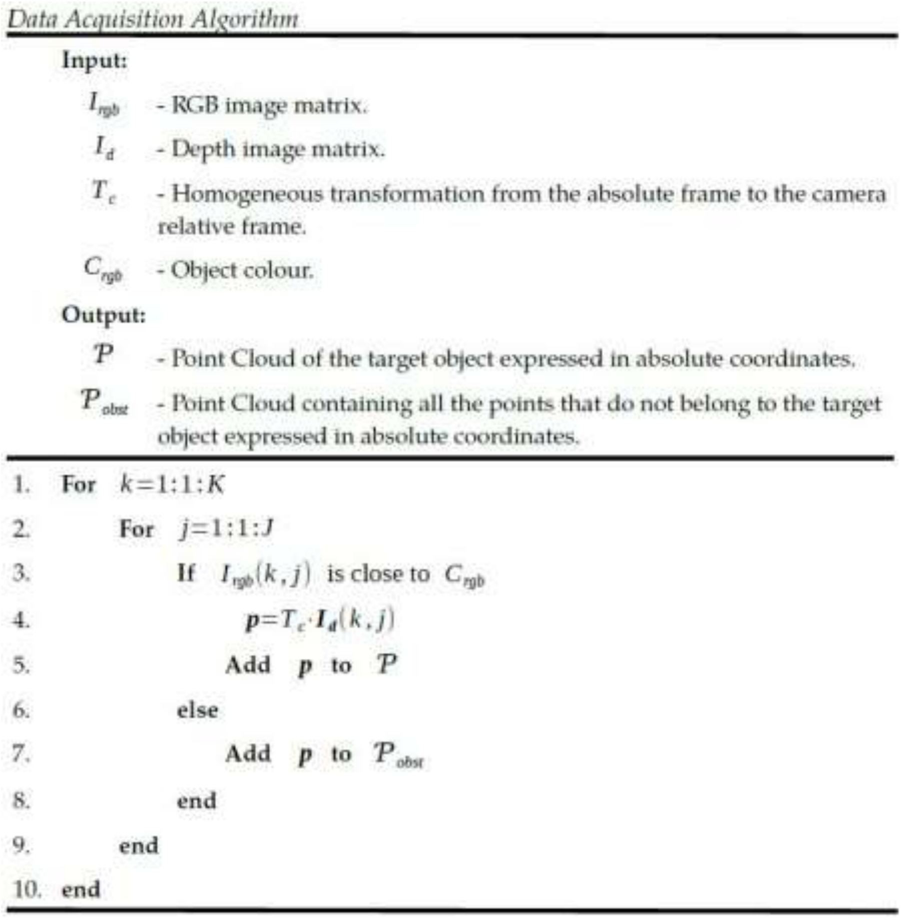
One example of application of this paradigm can be found in [89], where Structure from motion is used in conjunction with a voxeling technique to reconstruct surfaces and volumes of the objects. Those surfaces are then used to calculate their surface normals, which are then used as candidate gripping poses, later evaluated according to the size of contact area between object and gripper, momentum balance, manipulability and robot motions. In [90], grasp synthesis is performed simultaneously with the motion planning, with the configurations obtained during the execution of Rapidly-exploring Random Trees being verified for compatibility with the geometry of the object that must be grasped. In [80] the authors reconstruct the models with superquadrics approximation, while using “GraspIt!” for pose generation and validation.

However, the framework that seemed most fitting for application in this work was the one presented in [76]. Ancona proposes a four-step process to generate the desired grasping pose, comprised of: Point Cloud Acquisition, Data Pre-Processing, Grasp Generation and Grasp Selection. A brief description of each step is provided in the following section, followed by the alterations that were provided in this work in order to adequate his approach to the context of multi-modal robot programming.

In addition, as pointed in [77], suction grasping presents clear advantages over parallel-jaw and multi-finger grasping due to its ability to reach narrow spaces and pick objects from just a single point of contact. In addition, when considering a domestic environment, a parallel jaw gripper (as multi-finger grippers tend to be prohibitively expensive) would present a serious restriction in size of objects that could be grasped due to the maximum aperture of its jaws. Indeed, this superiority in performance and versatility were made evident in the Amazon Picking Challenge [91], where teams armed with suction grippers tended to outperform other groups. Therefore, though Ancona’s four-step methodology is followed, his method has been adapted for a suction-gripper based grasping problem.

2.3.4.1. *Ancona's 4-step method: Point Cloud Acquisition*

The first step of the method consists of finding a 3D description of the object to be grasped in the form of a 3D point cloud. Though many commercially available products exist that can provide the full colored 3D point cloud of the scene in front of a camera, it is still necessary to segment the scene into its several components – *i.e.* objects. In his particular application, [76] considered a simple color-based segmentation algorithm. In this method, a certain color threshold in the red, green and blue channels of the colored image is established, and any points whose colors fell within those limits were considered as part of the object. This algorithm is computationally efficient, running at $\mathcal{O}$(k), with k being the number of pixels in the color image and can be simply implemented, as shown in Figure 6.



Data Acquisition Algorithm

Input:

I_{rgb}	- RGB image matrix.
I_d	- Depth image matrix.
T_c	- Homogeneous transformation from the absolute frame to the camera relative frame.
C_{rgb}	- Object colour.

Output:

$\mathcal{P}$	- Point Cloud of the target object expressed in absolute coordinates.
$\mathcal{P}_{obst}$	- Point Cloud containing all the points that do not belong to the target object expressed in absolute coordinates.

```
1.  For $k=1:1:K$
2.      For $j=1:1:J$
3.          If $I_{rgb}(k,j)$ is close to $C_{rgb}$
4.              $\boldsymbol{p}=T_c \cdot \boldsymbol{I_d}(k,j)$
5.              Add $\boldsymbol{p}$ to $\mathcal{P}$
6.          else
7.              Add $\boldsymbol{p}$ to $\mathcal{P}_{obst}$
8.          end
9.      end
10. end
```

Figure 6 - Ancona's Image Segmentation Algorithm in Pseudo-code [76]

While this segmentation strategy sufficed for the purposes of that work, this simplistic approach is not applicable to a domestic or modern industrial setting. In domestic settings, the colors of objects are not set and objects often have many colors associated with them, like in Figure 7. In addition, many elements of the visual elements of a domestic scene may be composed of similar colors, as is common in harmonized and well-designed homes. This poses a threat to the house and to the humans around it, as the robot may be inclined to try and manipulate items it was not originally supposed to manipulate – or even worse – mistake the human's clothes for the object to be manipulated. This same threat is present in modern industrial settings, with a dynamic, ever changing environment around the automata – and with even greater risks for the humans, should they be mistaken for an object. The simplicity of this algorithm hasn't stopped it from being used in other contexts, however, as can be seen in [21].

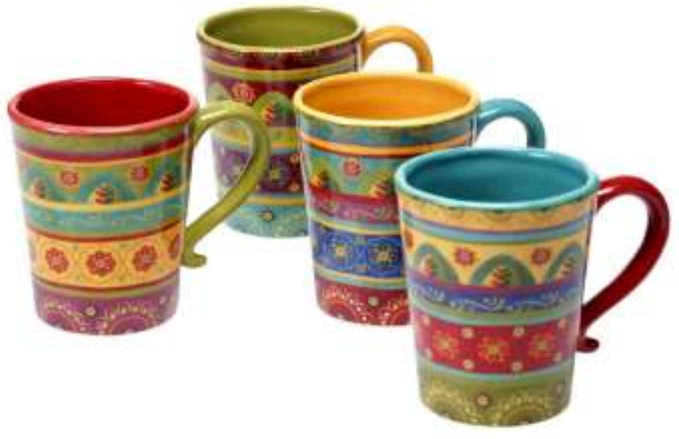


Figure 7 - Domestic objects that would provoke failure in threshold-based segmentation techniques [92]

Having considered these caveats to Ancona's algorithm and the additional information provided by the HMI, a small section of this work will be dedicated to analyzing image segmentation algorithms to be applied in this step.

2.3.4.2. Ancona's 4-step method: Data pre-processing

After applying any method of image segmentation to the colored 3D point cloud, the resulting model will still be raw and populated with false-positive and false-negative identifications. These detection imperfections would cause the grasping algorithm to underperform and, thus, need to be filtered out through some pre-processing steps, namely Density Filtering, Uniformity Filtering and Surface Normals Smoothing.

Density Filtering

Due to imperfections in the image segmentation algorithm and uncertainties in the depth sensor, the identified model may contain several false-positive identifications. These are usually present in the form of low-density regions in the observed point-cloud, and may be filtered out using a simple density-based filtering algorithm designed to remove outliers [93]. In [76], this filter was implemented in the following way: Let $\boldsymbol{p_i}$ be the ***i****-th* point of the point cloud $\boldsymbol{\mathcal{P}}$ and $\boldsymbol{N(K)}$ the *neighbor set,* defined as the set of $\boldsymbol{K}$ points closest to $\boldsymbol{p_i}$. Then the local density of point $\boldsymbol{p_i}$, $\mathbf{d_i(K)}$, can be calculated by Equation 1, where $||V||$ stands for the L2 norm of a vector.

Equation 1 -Point Cloud Density in a neighborhood K of point i

$$d_i(K) = \frac{1}{K} \sum_{p_j \in N_i(K)} \|p_i - p_j\|$$

The mean point cloud local density and the standard deviation of said mean**,** $\boldsymbol{\mu(d_i(K))}$ and $\boldsymbol{\sigma(d_i(K))}$ can then be calculated. The filter can then be defined as in Equation 2, with n being a flexible threshold for variance in density, set as 3 in [76].

Equation 2 - Density Filter Definition

$$F_d(P,K) = \{p_i \in P, d_i(K) > \mu(d_i(K)) - n\sigma(d_i(K))\}$$

The effects of this filter can be seen in Figure 8 , with the first image to the left being the raw point cloud, the middle image being the resulting filtered cloud and the rightmost image being the points that were filtered out, which clearly shows the removal of outliers.

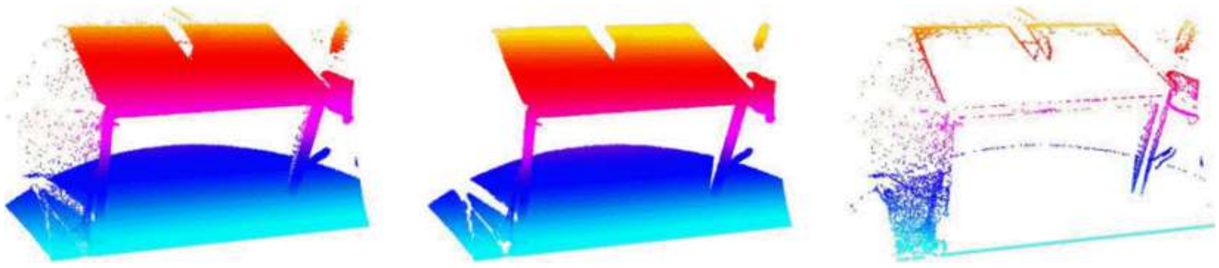


Figure 8 - Outlier removal through density filtering [93]

Uniformity Filtering

As an additional filtering step, [76] suggested removing points within the point cloud that were too close from one another, in order to improve the stability of the inference of surface normals and the performance of grasp selection algorithms, as it reduces the amount of candidate grasping positions to be analyzed. This uniformity filter is defined as in Equation 3, where d_{min} stands for the minimum allowed distance between points in the point-cloud.

Equation 3 - Definition of uniformity filter [76]

$$F_u(P, d_{min}) = \{p_i \in P, \nexists p_i \in P, \|p_j - p_i\| < d_{min}\}$$

This method, though computationally efficient, is not very adequate, as it simply removes points that are close together, simply discarding their information without making any use of it. A better method for smoothing out point clouds is presented in [94], where the Weighted Locally Optimal Projection (WLOP) is introduced. Given an unorganized point cloud P, WLOP defined a set of projected points X by a fixed point iteration where, given the current iterate X^k , $k \in \mathbb{N}$, the next iterate X^{k+1} is defined by Equation 4, where $\boldsymbol{v_j}$ is defined in Equation 5 , w_i^k is defined in Equation 6 , $\xi_{ij}^k = x_i^k - p_i$, $\delta_{ii'}^k = x_i^k - x_{i'}^k$, $\theta(r)$ defined in Equation 7,being a rapidly decreasing smooth weight function that penalizes point x_i that get too close to other points in X, $\alpha_{ij}^k = \frac{\theta(\|\xi_{ij}^k\|)}{\|\xi_{ij}^k\|}$, $\beta_{ii'}^k = \frac{\theta(\|\delta_{ii'}^k\|)|\eta(\|\delta_{ii'}^k\|)|}{\|\beta_{ii'}^k\|}$ and η being a repulsion function, in this case being defined as $\eta(r) = -r$. This algorithm is an improvement over the Locally Optimal Operator (LOP), as it allows for the attraction of point clusters in a given set P to be relaxed by the local density *v* in the first term, and the repulsion force from points in dense areas to be strengthened by the ω term. This process produces a point cloud that still adheres to the original shape, but has its points more evenly spread out, with a more constant density, as can be illustrated in Figure 9. As can be seen, this method produces an evenly distributed sparse point cloud, with superior quality than that obtained through the usual LOP method.

Unfortunately, even though WLOP provides excellent filtering and normalizing features in point cloud density, its iterative nature and heavy computational costs make it prohibitively expensive to be applied in a real-time-based HRI, though the suggestion of its implementation in systems in which time is not of essence remains noted.

Equation 4 - Definition of the iteration of WLOP

$$x_i^{k+1} = \sum_{j\in J} p_j \frac{\frac{\alpha_{ij}^k}{v_j}}{\sum_{j\in J}\left(\frac{\alpha_{ij}^k}{v_j}\right)} + \mu \sum_{i\prime\in I\backslash\{i\}} \delta_{ii\prime}^k \frac{\omega_{i\prime}^k \beta_{ii\prime}^k}{\sum_{i\prime\in I\backslash\{i\}}(\omega_{i\prime}^k \beta_{ii\prime}^k)}$$

Equation 5- definition of v_j

$$v_j = 1 + \sum_{j\prime\in J\{j\}} \theta(\|p_j - p_{j\prime}\|)$$

Equation 6- definition of w_i^k

$$w_i^k = 1 + \sum_{i\prime\in I\backslash\{i\}} \theta(\|\delta_{ii\prime}^k\|)$$

Equation 7 - Rapidly decreasing weighting function

$$\theta(r) = e^{-\frac{r^2}{(h/4)^2}}$$

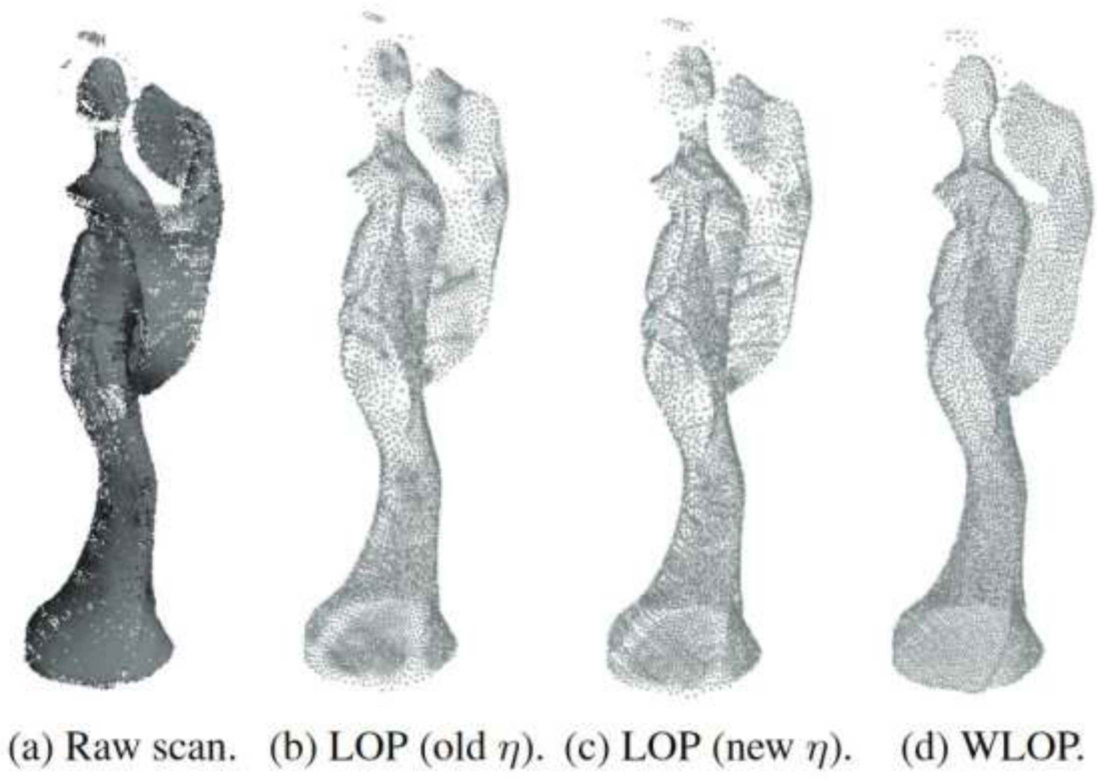


Figure 9 - Performance comparison of WLOP on a scan of a Japanese lady statue[94]

Surface normals smoothing

After having performed these filtering operations, Ancona then estimates the surface normals of the filtered point cloud. In order to do that, for each point p_i in the point cloud **P** he defines the neighbor set $N_i(K)$ of K nearest points. Within this set, he performs the regression of the plane equation ($z_{pk} = \beta_0 + \beta_1 x_{pk} + \beta_2 y_{pk}$ by applying the Least Squares algorithm, with the resulting normal vector to point $\mathbf{p_i}$, $\mathbf{n_i}$, being described by Equation 8.

Equation 8 - Surface normal n_i

$$n_i = \frac{[\beta_1, \beta_2, 1]}{\sqrt{1 + \beta_1^2 + \beta_2^2}}$$

These calculated surface normals may be smoothed out even further by defining the smoothed surface normal, $\tilde{n}_i(K)$ as the average surface normal within a vicinity K of point *i*, as shown in Equation 9

Equation 9 - Smoothed surface normal $\tilde{n}_i$

$$\tilde{n}_i(K) = \frac{\sum_{k=1}^{K} n_k}{\|\sum_{k=1}^{K} n_k\|}$$

This smoothing operation improves the estimation of surface normals, in particular in corners, in which the estimation of surface normals through the Least Squares method presented above fails, due to there being no plane defined by the nearest neighbors of the point [76]. The results of this operation can be seen in Figure 10.

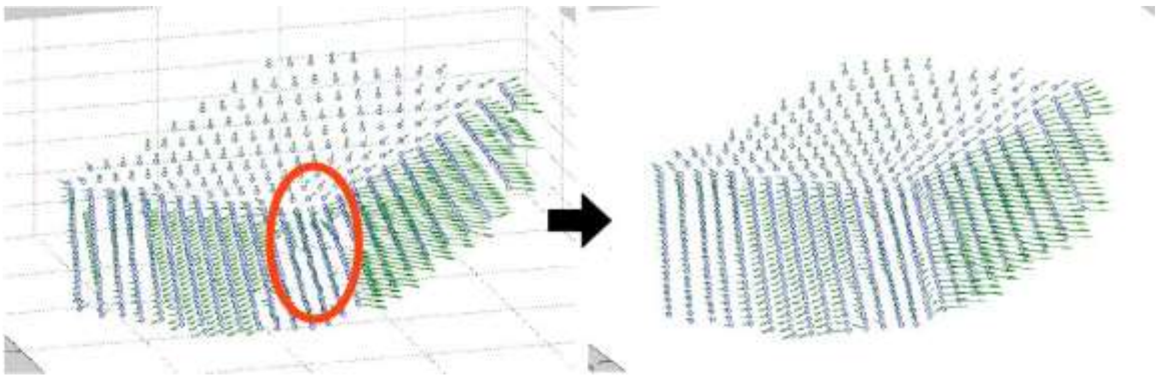


Figure 10 - Example of surface normal smoothing [76]

2.3.4.3. Ancona's 4-step method: Grasp Generation

In his work, Ancona detailed two analytical methods to generate valid grasping poses, one called Locally Approximated Algorithm and another one he called Exact Algorithm. In the locally approximated algorithm, he approximates the gripper by three planes, disposed in space as shown in Figure 11 and looks for candidate gripping points that keep the parallelism between its fingers and the faces of the objects, while avoiding collisions with it in the neighborhood of the gripper structure, limiting his search to positions where the object's surface normals would either be aligned with the gripper's fingers or with the gripper's base. [76]

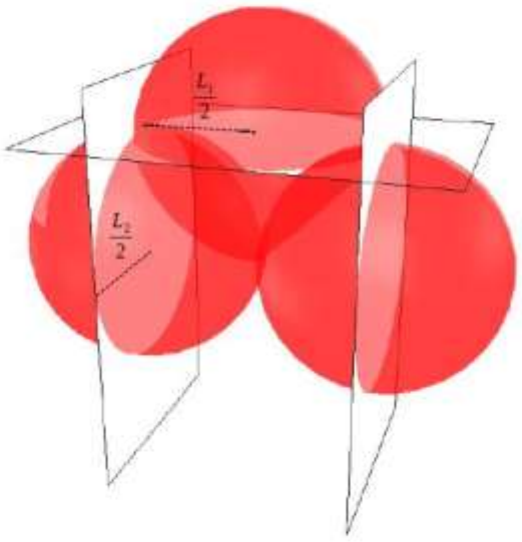


Figure 11 - Gripper planar approximation [76]

In the exact algorithm, Ancona approximates the gripper as a junction of 3D cuboids and performs the same checks he performed in the approximated algorithm, but for the 3D structure. As his gripping pose generation strategy is focused in generating grasping poses for parallel-jawed grippers, this section is not detailed any further in this work, being replaced by section 2.5. In general lines, however, this step is responsible for generating a number of candidate grasping poses, to be selected in the following step according to feasibility and quality of grasp criteria.

2.3.4.4. Anconas 4-step method: Grasp Selection

Once a list of candidate poses has been produced by the previous step, one must now evaluate them according to some metric in order to select the best possible among them. Ancona chose to apply those presented [81], [95], from which he selected three main criteria: *Feasibility, Stability* and *Robot Motion*. In the context of his work, feasibility scores are used to rule out grasping poses which result in collisions with the object or its surroundings or grasp configurations that result in placing the robotic manipulator in a singularity. Stability scores are supposed to grade grasping poses with respect to the “firmness” of each grasping pose, being defined as robustness to external disturbances and wrenches that act on the grasped object. Finally, the Robot motion score tends to evaluate which poses require the least motion of the manipulator's joints. Ancona then synthetizes them into a single global score, used for grasp evaluation. Each of the scores is detailed in the following section for clarity.

Feasibility

First, the manipulator's kinematics is used to determine if the suggested pose is reachable. As a second step, given a reachable pose, a model of the robot must be provided, possibly by approximating the manipulator's links as cuboids, to check if by positioning the robot in that pose results in having any of the points detected from the scene colliding with the manipulator or the gripper. This is a Binary eliminatory index, not really providing a score rather than "feasible" or "unfeasible".

Stability

As pointed out by [76], a more precise estimation of the stability of grasp poses with the traditional force closure method is impossible without a complete dynamic model of the object, including its masses and inertias, which is not available from its Point Cloud. Therefore, he proposes two heuristic methods to approximate this scoring, both based in the distance between the grasping point and the estimated center of mass of the object. In one method, the center of mass is estimated as the centroid of the object's point cloud, with $\boldsymbol{P_{COM}} = \frac{1}{|P|}\sum_{i=1}^{|P|} p_i$. In this case, the stability of a grasping pose G is **$\mathbf{S_{COM}(G)}$**, defined in Equation 10.

Equation 10 - Stabilit score with centroid

$$S_{COM}(G) = \frac{1}{\|P_G - P_{COM}\|}$$

Another possible heuristic, designed to compensate for the incompleteness of the model and the varying densities of points in the perceived point cloud would be to estimate the center of mass as P_{ECOM}, defined in Equation 11, with its stability score $S_{ECOM}(G)$ being defined in similar fashion as $S_{COM}(G)$, in Equation 12.

Equation 11 - Heuristic estimate of the center of mass of a point cloud

$$P_{ECOM} = \left[\frac{x_{max} + x_{min}}{2}, \frac{y_{max} - y_{min}}{2}, \frac{z_{max} - z_{min}}{2}\right]^T$$

Equation 12 - Stability score using P_{ECOM}

$$S_{ECOM}(G) = \frac{1}{\|P_G - P_{ECOM}\|}$$

One final heuristic evaluation of stability of grasp posture is proposed in [76], which implements a way of estimating the extension of the object that is in full contact with the gripper by extending the gripper model of a small amount ε and counting the amount of points of the object's point cloud that are now contained within this extended model. Formally, this score is $S_{CONTACT}(G,P) = |P_{CONTACT}(G,P)|$.

Robot motion score

This score arises from the desire to minimize the movement of the robot, improving the energy efficiency of the algorithm and reducing the execution time of the grasping task [76]. Let a robot configuration required to execute a grasping pose G be defined by the state of its joints $\mathbf{q}_G$, obtained through inverse kinematics of the manipulator. Let $\mathbf{q}_{now}$ denote the current state of the manipulator. The robot motion score, $S_{RM}(G)$ can then be defined as in

Equation 13 - Robot Motion Score $S_{RM}(G)$

$$S_{RM}(G) = \frac{1}{\|q_{now} - q_G\|}$$

Global Performance Index

Having defined the individual performance indices, a global performance index was devised, S(G,P), being defined in Equation 14, where W(G) is a weighting factor used by Ancona to prioritize grasping poses with vertical configurations as they synergized better with the redundancies of his mobile manipulator.

Equation 14 - Global performance index S(G,P)

$$S(G,P) = W(G)S_{contact}(G,P)S_{ECOM}(G)$$

2.4. Image Segmentation Algorithms

As discussed in section 2.3.4.1, Ancona's approach of segmentation the image by a simple color thresholding algorithm is not suitable for the application of a versatile HMI. Given that through skeletal tracking and pointing the robot is provided with a sense of selection, one of the natural ways for dealing with the issue of finding out which object must be manipulated lies in simply segmenting the image and manipulating which object has been indicated, much like a *point-and-command* paradigm would advise. In that context, one must then evaluate image segmentation algorithms that could serve as an alternative to color thresholding, while still being computationally efficient enough to be used on-line without jeopardizing execution times. As pointed in [96], image segmentation remains an active area of research in computer vision, since it is an inherently ill-posed problem.

Though recent advances in artificial neural networks have afforded a great increase in usage of the technique in the field of computer vision, they often require powerful CPUs and GPUs to train and heavy hardware to be run on, not being, yet, suitable for use in embedded hardware. These techniques will not be exposed in great detail in this section, but briefly mentioned, as their successes are staggering and their results quite promising. Notable examples of the application of artificial neural networks in image segmentation can be found in [97], which uses a simple multi-layer perceptron to reconstruct cells and neurons and [98], which makes use of Deep Convolutional Neural Networks (DCNNs) associated with fully-connected Conditional Random Fields (CRF) to match – and in some cases outperform- state-of-the-art algorithms in image segmentation standard datasets, like Cityscapes [99], as can be seen in Figure 12, where G.T. stands for Ground Truth.

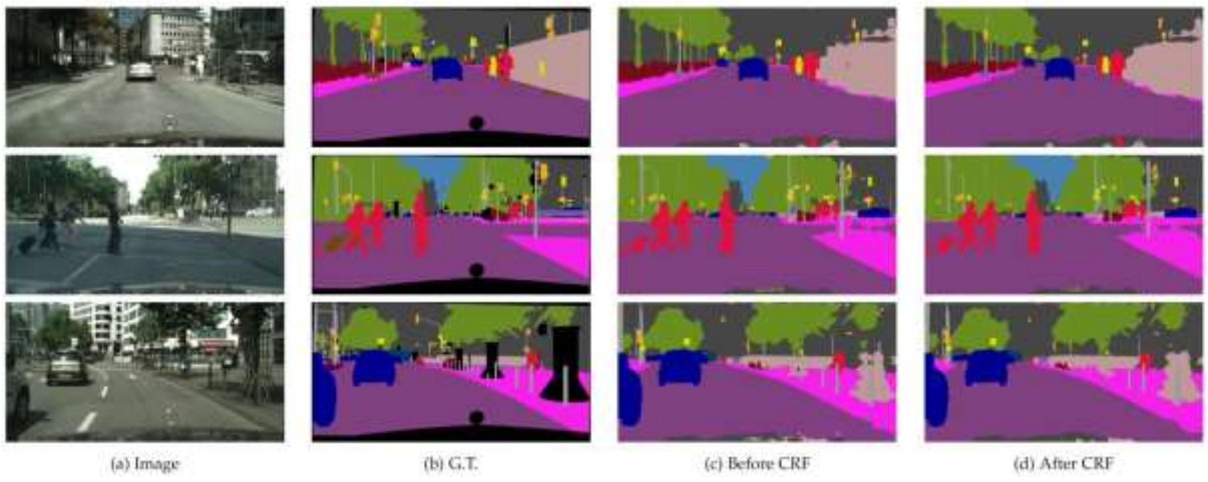


Figure 12 - Performance of DeepLab in Cityscapes dataset [98]

This section will thus focus on detailing analytical methods for image segmentation. Analytical image segmentation algorithms can be grouped into five main categories: threshold-based, edge-based, region-based, watershed-based and graph-based approaches [100], detailed in the following sections.

2.4.1. Threshold-based segmentation

This segmentation strategy is based on the idea that different parts of the image, like foreground and background, follow a distinct set of values and that, by finding the boundary between those values, one may segment the image by putting each pixel in the appropriate bin. Threshold based segmentation is held as one of the easiest, fastest and most intuitive segmentation techniques [100], being used in [21], [76], in which the authors established thresholds in the 3 color channels of an image and segmented the image according to them.

In more general segmentation challenges, such as separating foreground and background from any given image, Otsu's algorithm [101] may be used in order to automatically find the appropriate threshold, by maximizing pixel variance between the regions and minimizing pixel variance within them. Some improvements over the algorithm have been made in recent years by proving its analogy to k-means clustering techniques [102], yet its peak performance lies in segmenting images in foreground and background, which is, unfortunately, not enough for the proposed application in this work.

2.4.2. Edge-based segmentation

Edge-based segmentation methods rely on the fact that the boundaries between different elements in images are often marked by sudden changes in pixel properties and that, by finding these sudden spikes in pixel property, one could find the outlines of different visual elements within a scene and, based on these outlines find the separate elements in the scene, which will be surrounded by their edges. These edges are often found by applying filters that approximate pixel intensity derivatives, like Canny [103], and Sobel filters or by other techniques, such as Prewitt or Roberts methods [100]. These techniques, however, are not very robust to noise in images and tend to underperform in natural, more nuanced photos, as the gradients of color tend to be smoother and boundaries blurred. These methods also tend to be plagued by false edge detections [100], like it can be seen in Figure 13, where it fails to detect edges between teeth and tongue and either ignores boundaries between the dog and the car or finds edges where there are none. These methods, thus, are also not suited for this works application, as it aims to be applied in domestic organic environments.

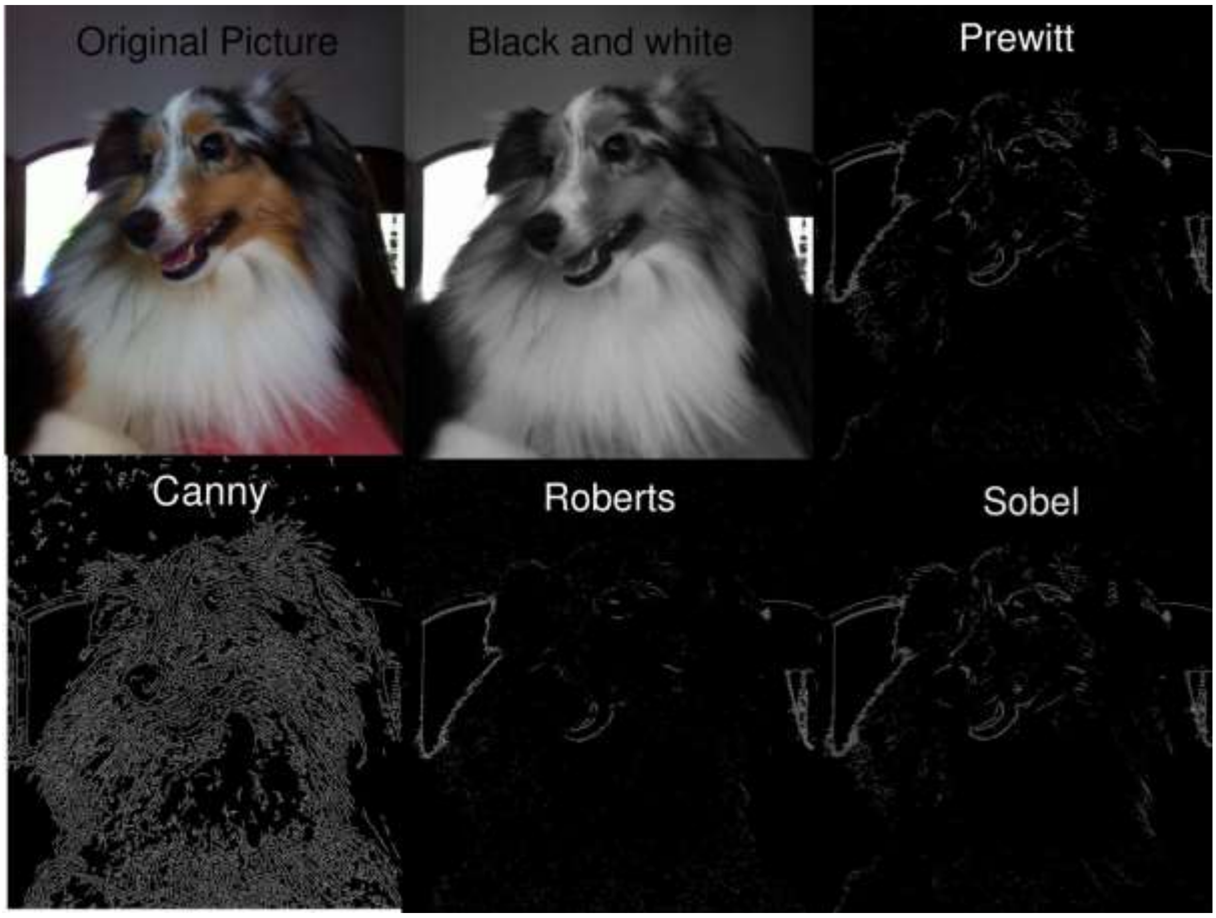


Figure 13 - Lasagna, my dog, under several edge detection schemes

2.4.3. Region-based segmentation

Region-based algorithms can be divided into two categories: region growing and region merging algorithms. In region growing algorithms, pixels are grouped into sub-regions according to predefined criterion, with the number of regions being defined by the number of initial “region seeds” provided to the algorithm. The performance of this algorithm is heavily dependent on the choice of seeds. Region splitting and merging algorithms, on the other hand, start by segmenting the image into random disjoint small clusters and proceeding to merge neighboring clusters according to some predefined criteria, often minimizing an energy function. These algorithms, too, are heavily reliant on the quality of the initial super-segmentation [100], [104], [105]. Due to their heavy reliance on seeds and priors, these techniques are also not suited for the application discussed in this work.

2.4.4. Watershed-based segmentation

Watershed-based segmentation can be explained by a simple physical analogy. Let the image be seen as a topographical map, with the pixel intensity of the black-and-white image being seen as the height. If this topography starts being filled with water it springing from the local minima in height. As the water level rises, those valleys should become isolated lakes. As the water level continues to rise, though, some of these isolated lakes will tend to merge. At these merging points, called watersheds, we erect infinitely tall dams, which will be the borders between separate regions in our image. Once the water level has risen atop all the topography, the dams should now define clear borders between isolated regions in the map.

In a similar fashion, watershed algorithms transform images into these depth maps (usually by looking at pixel gradient measures) and iteratively find those borders between the regions [106]. Their performance, though, is also heavily reliant on the initial seeds (the water sources) and their resulting images tend to be over-segmented due to irregularities of the pixel gradients and to noise [100]. This method, then, is also not suited for this application.

2.4.5. Graph-based segmentation

The final family of image segmentation algorithms relies on graph theory to section the image. This is done by transforming the image’s elements into nodes in a graph- and finding a set of disjoint subgraph to this graph that establishes the most uniformity within subgraphs and the most variance between subgraphs. This transformation of the image into a graph is useful because there are many efficient algorithms established in the field of graph theory that can be used to find suitable segmentations. Indeed, so strong is this connection that [96] classifies these algorithms based on the original graph theoretical problem they arose from, namely: minimal spanning tree based, graph cut based, shortest path based etc.

In their survey of these algorithms, [96] proposed a framework to analyze the efficiency of segmentation algorithms, based on 5 indices: Probabilistic Rand (PR), Normalized Probabilistic Rand (NPR), Variation of Information (VI), Global Consistency Error (GCE) and Boundary Displacement Error (BDE), briefly explained below.

PR provides a statistical measure of segmentation correctness. If we describe image segmentation in a binary representation $(l_i^{S_k} = l_j^{S_k})$ on each pair of pixels (x_i,x_j), then these numbers follow a Bernoulli distribution with a random variable whose expected value is $\mathbf{p}_{ij}$. If we are to let a series of humans perform manual segmentation on the test images to produce the set of possible ground truths $\mathbf{\{S_K\}}$, then the PR score can then be defined as in Equation 15, where **N** is the number of pixels and $\mathbf{p_{ij}}$ is the ground truth probability that the labels of $\mathbf{(x_i,x_j)}$ are the same [96]. The PR score is a number between 0 and 1, indicating how much the algorithm agrees with the human segmented images.

Equation 15 - Definition of PR

$$PR(S_{test},\{S_k\}) = \frac{1}{\binom{N}{2}}\sum_{i<j}^{i,j}\left[I\left(l_i^{S_{test}} = l_j^{S_{test}}\right)p_{ij} + I\left(l_i^{S_{test}} \neq l_j^{S_{test}}\right)(1-p_{ij})\right]$$

NPR is a refinement of PR, because it provides a way to compare scores across algorithms and images by providing an “expected” value for PR, used to normalize its value across evaluations. NPR can be similarly defined as in Equation 16, where **Φ** is the number of different images in the dataset, K_Φ is the number of ground truth segmentations of a given image **φ** and $\boldsymbol{p'_{ij}}$ is can be estimated by Equation 17 [96].

Equation 16 - Expected value of NPR

$$E(NPR(S_{test},\{S_k\})) = \frac{1}{\binom{N}{2}}\sum_{i<j}^{i,j}\left[p'_{ij}p_{ij} + (1-p'_{ij})(1-p_{ij})\right]$$

Equation 17 - Estimation of p'$_{ij}$

$$p'_{ij} = \frac{1}{\Phi}\sum_{\Phi}\frac{1}{K_{\Phi}}\sum_{k=1}^{K_{\Phi}} I(l_i^{S_k^{\Phi}} = l_j^{S_k^{\Phi}})$$

VI is an information-theory based metric. It can be interpreted as the average conditional entropy of one segmentation given another, as defined by Equation 18, where **I()** stands for the mutual information between segmentation $\mathbf{S_{test}}$ and the ground truth, $\mathbf{S_K}$, while **H()** stands for the entropy of the given segmentation [96].

Equation 18 - Definition of VI

$$VI(S_{test},S_K) = H(S_{test}) + H(S_K) - 2I(S_{test},S_K)$$

GCE, in turn, is a metric designed to quantify the segmentation quality in different granularities. Its definition is as follows: Let R(S,p$_i$) be the set of pixels in segmentation S that contains pixel p$_i$, then the local refinement error is defined in Equation 19 and the GCE index is defined by Equation 20, where S$_1$ and S$_2$ are two segmentations at different granularities, "\" stands for difference between sets and **n** is the number of pixels within the image [96].

Equation 19 - Local Refinement Error

$$E(S1,S2,p_i) = \frac{|R(S_1,p_i)\setminus R(S_2,p_i)|}{|R(S_1,p_i)|}$$

Equation 20 - Global Consistency Error

$$GCE(S_1,S_2) = \frac{1}{n}min\left\{\sum_i E(S_1,S_2,p_i), \sum_i E(S_2,S_1,p_i)\right\}$$

Finally, BDE takes as a base for the metric the boundaries between segmentations, defining the error of one boundary pixel as the distance between it and its closest pixel in the other boundary image. Let B_1 stand for a boundary point in segmentation S_1. Then, the BDE for point B_1 is the minimum distance between it all the boundary pixels in the ground truth [96].

Making use of these metrics, the authors in [96] designed an experiment where they measured these 5 performance indices for a set of algorithms on the Berkley dataset [107]. The algorithms that were chosen for this experiment were the best three of the best performing and fastest easily available graph-based segmentation algorithms, namely Felzenszwalb and Huttenlocher's (FH) [108], normalized cut (Ncut) [109] and ratio cut (Rcut) [110], as well as the traditional mean shift algorithm [111], being used as a baseline. The results of the experiment may be seen in Figure 14. The reader is kindly reminded that an ideal segmentation algorithm should have high PR and NPR scores and low GCE, VI and BDE scores. The authors note that all four algorithms present good performance in PR and NPR, though mean-shift is clearly superior to the other ones in those metrics. The authors also note, however, that no single algorithm dominates all metrics, with FH being the best in BDE and VI.

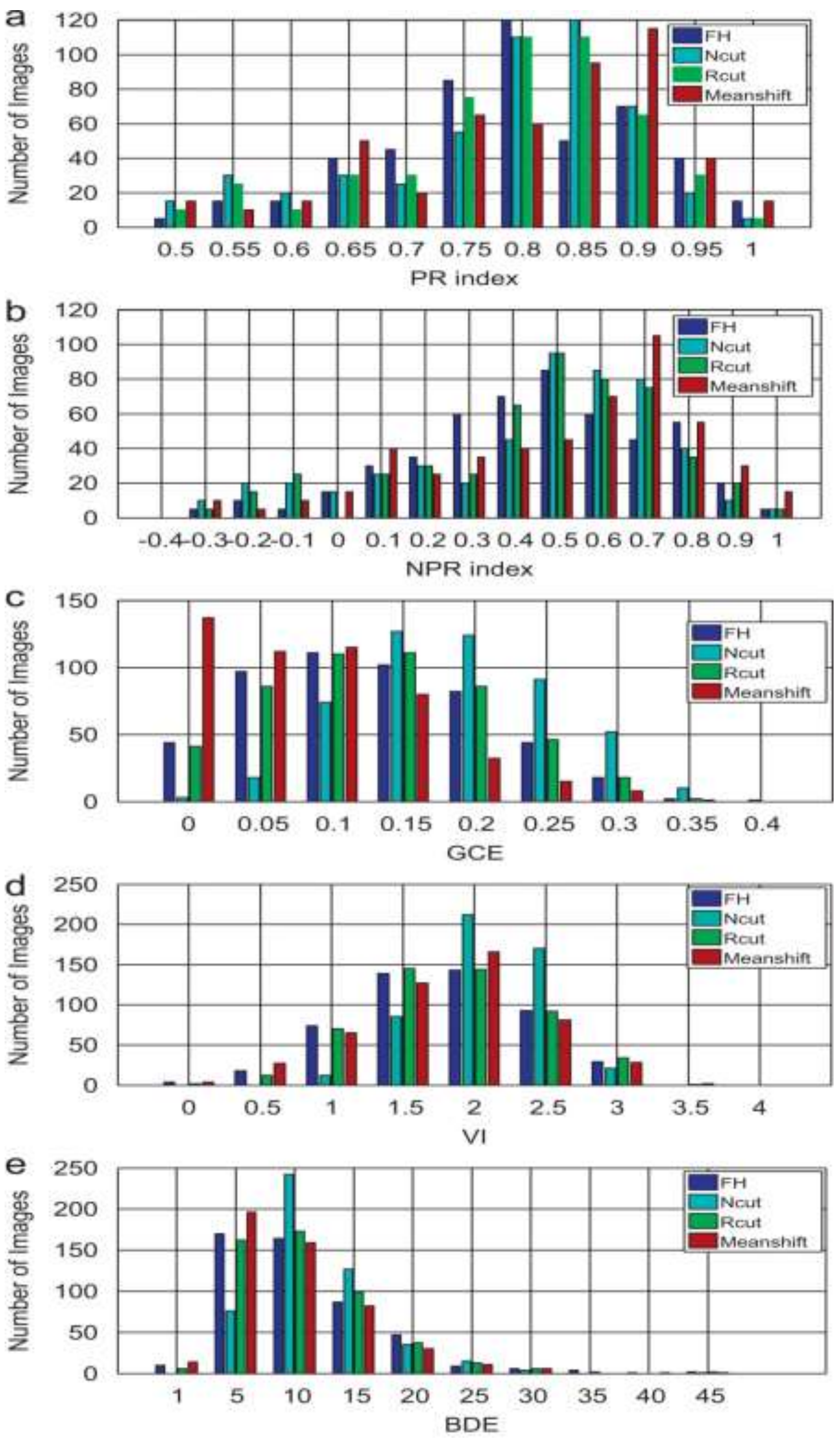


Figure 14 Image segmentation algorithms evaluation [96]

In a subsequent evaluation step, [96] performed another round of experiments, this time aiming to determine the stability of the performance of the three graph-based algorithms (FH, Ncut,Rcut) with respect to their hyperparameters, in which they concluded that FH tends to extract the main structure of objects, particularly when the number of segments was high. The authors also noted that, though Rcut had better stability of performance with respect to its hyperparameters, FH had better average segmentation quality, measured by NPR, than the other two [96].

Having considered these performance evaluations, one may be inclined to consider that the ideal segmentation algorithm would be mean-shift – as it outperformed the graph-based algorithms in NPR and PR scores. However, as this segmentation algorithm is supposed to be applied on-line, the use of an iterative algorithm could impact the HRI's run-time performance (since the time complexity of the mean-shift algorithm is $\mathcal{O}(Tn^2)$, where n is the number of pixels and T is the number of iterations).

Therefore, one of the graph-based algorithms had to be selected instead. As FH presented a good performance on average in all scores, had a time complexity of $\mathcal{O}(m\log m)$ (where m stands for the number of edges in the graph) and had a robust performance with respect to hyperparameters, it was chosen for this project's application. As an added bonus, there had been publications which extended FH method to an application where a full RGB-D image was available in the context of grasping unknown objects [112], which further inspired me to use it in the project. As such, the FH algorithm is presented in detail in section 3.2.2.2, Software implementation.

As a final remark, recent literature suggests that the use of an initial super-segmentation (also named superpixels) often generated by the simple linear iterative clustering (SLIC) algorithm could improve the performance of image segmentation algorithms, as it reduces the number of nodes in the graph and this, theoretically, would improve performance at run time [118]. Unfortunately, during experimental trials using standard libraries, my experiments showed that this did not positively impact the performance of the segmentation algorithm, neither in accuracy, nor in run time. The accuracy loss comes from the fact that superpixelation causes a loss of detail and forces homogeneity within pixels and the loss of run time performance is probably due to the fact that superpixelation eliminates the inherent definition of neighbors in an image – and the calculation of superpixel distances, associated with the time to run superpixelation does not compensate the run-time improvement caused by the reduction of dimensionality of the segmentation algorithm's input in the resolutions used in these experiments.

2.5. Grasp generation for suction-cup grippers

As remarked by [77], most of the research done in the area of suction-cup gripping has been guided by two basic heuristic principles being applied directly on point clouds: Grasping near the inferred object centroid [116] and just targeting planar surfaces [117]. The intuition behind these strategies is that by trying to grasp the object from its most planar surface, the suction gripper will have better chances of maintaining a vacuum – and thus, holding on tightly to the object it is trying to grasp and by gripping an object as close as possible from its center of gravity, the gripper is maximizing its stability with respect to external perturbations (note, for instance, how the stability score is calculated in Ancona's work, reflecting this notion.

In an effort to improve upon these heuristics, a mixed approach was proposed, which tries to implement both of these heuristics at once and is described in further detail in section 3.2.2.4.

3. Proposed Method & Implementation

3.1. Proposed Method

As mentioned in Section 1.2.1, this project aimed at developing a novel HRI that would allow humans to command robots' grasping tasks for unknown objects. After reviewing the possible levels of autonomy, this project could be classified with a LORA of Batch Processing (the reader being referred to Figure 4 for further reference), as the human is responsible for indicating which object to the picked and where to place it, while the robot is responsible for processing all the tasks and motions required to achieve said goal. In a brief review of Multimodal HRIs, it was concluded that Audio-visual HRIs tend to be the most intuitive for human operators and- for simplicity, a visual-based HRI was chosen for the project. As for the grasping task planning, the pipeline proposed in [76] was used. Having these things in mind, the workflow presented in Figure 15 is proposed.

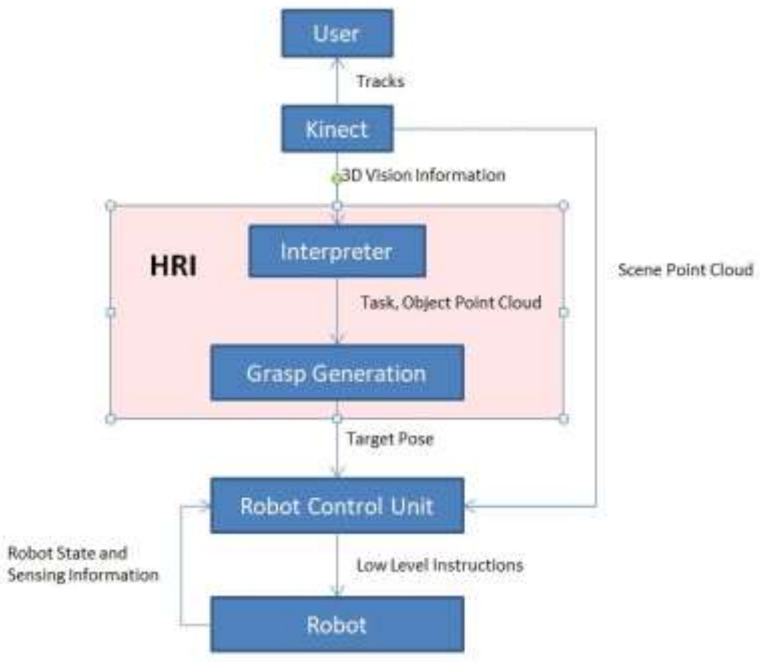


Figure 15 – High-Level Proposed System Workflow

In this workflow, we see that the Kinect is responsible for tracking the human and providing the HRI with the visual information it needs. This is done through three embedded Kinect V2.0 functions: skeletal tracking, hand state tracking and colored point cloud generation. Skeletal tracking provides a 25-point model of the human body, as can be seen in Figure 16, with their respective corresponding tracked points and hand tracking is a rudimentary gesture recognition module, which provides us with three possible states for each hand: Open, Closed in a fist and “Lasso”, equivalent to pointing with the middle and index fingers held together [119].These 3 components are fed as visual information to the HRI. The HRI is composed of two main components: the interpreter, which interprets the user’s commands, and the grasp generation module, which generates the grasping pose for the robot’s end-effector that is fed into the Robot Control Unit, which will then process the low level commands for the robot. Each module is discussed in detail below.

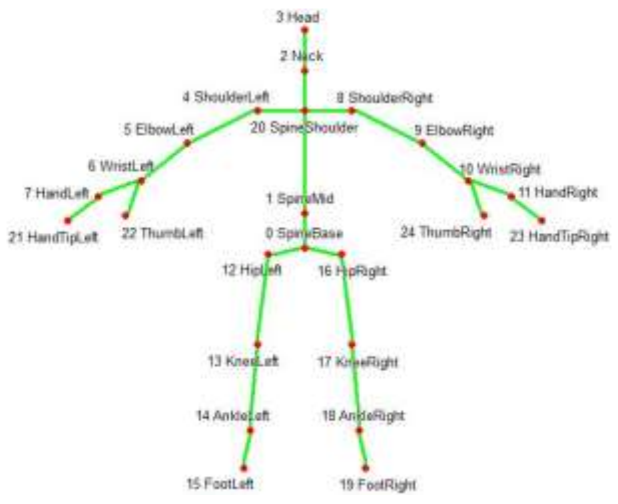


Figure 16 - Kinect Skeletal Model [120]

3.1.1. Interpreter

In order for an HRI to be effective, one must establish a clear communication protocol between the user and the robot. A usual syntax for this protocol consists of a Subject Verb Object syntax. As this HRI is designed (presently) to control a single robot, the Subject is already defined. We must then establish the action- which fills the role of the Verb and the Object, which in this case is either the object to be manipulated or the point where to place it and the Object, which is the object to be manipulated or the position to place it. These tasks are performed by the Interpreter module, whose workflow is shown in Figure 17

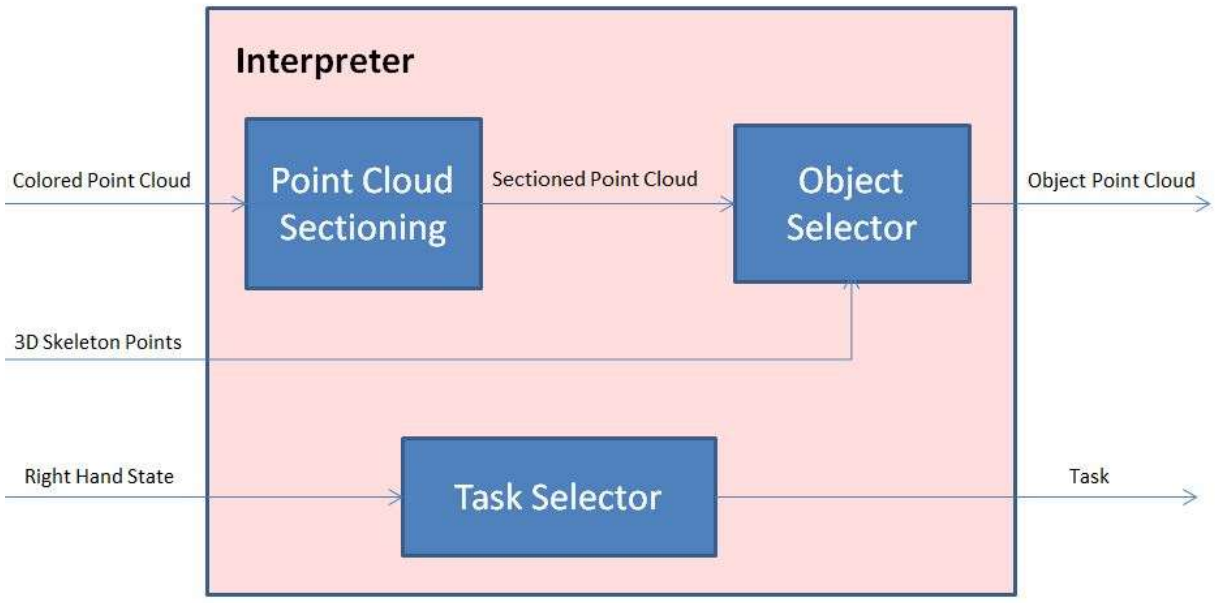


Figure 17 - Interpreter Workflow

The selection of the action is performed by the Task Selector. In this particular application, since there are two possible commands and three possible right hand states, one may map each hand state to a given action – Pick, Place or wait for command, as a simple dictionary. Once the human command has been given by the right hand, another problem still stands: How to identify the object pointed at by the human.

In order to do so, one must section the scene observed by the Kinect. In this particular work, the altered Felzenszwalb and Huttenlocher's (FH) algorithm, proposed in section 0 is used. Then, given the sectioned point cloud and the estimated skeleton, we infer the object that was being pointed at by projecting a line from the user's left arm (defined by the user's elbow and wrist) and considering as the object to be manipulated all the points in the point cloud that belong to the same section as the point intersected by the user's extended left arm line. In pseudo-code, the process is defined as in Figure 18 – :

```
Algorithm 1 Original Target Identification Algorithm
    Input: $\mathbf{p_{elbow}}$,$\mathbf{p_{wrist}}$,$\mathbf{P_{finger}}$, 3D position of the finger, elbow and wrist.
           $\mathbf{P}$: Scene Point cloud
           action : either "Pick" or "place"
           $\mathbf{Im_{sectioned}}$ : Sectioned depth image.
    Output: $\mathbf{P_{raw}}$: Target's point cloud
1: procedure GETTARGET($\mathbf{p_{elbow}}$,$\mathbf{p_{wrist}}$,$\mathbf{P_{finger}}$,$\mathbf{P}$,action, $\mathbf{Im_{sectioned}}$)    ▷ Finding the user's target
2:     Define the line
               $\mathbf{L} = \mathbf{P_{finger}} + (\mathbf{P_{wrist}} - \mathbf{P_{elbow}})\lambda$
3:     Find the set of points $\mathbf{P_{inter}} \in \mathbf{P}$ such that their distance to line $\mathbf{L}$ is beneath a given threshold (say, 5cm, to account for pointing imprecision and for point cloud sampling)
4:     Find the point $\mathbf{p_{closest}}$ whose Euclidean distance to point $\mathbf{p_{finger}}$ (tip of the left hand) is the smallest. Find its section, $\mathbf{S_{closest}}$, by projecting it into $\mathbf{Im_{sectioned}}$.
5:     if action is "Pick" then
6:         return return all the points in the section $\mathbf{S_{closest}}$
7:     else
8:         return the point of intersection, $\mathbf{p_{closest}}$
```

Figure 18 – Original Target Identification Algorithm

The action to be performed and the object's point cloud are then fed into the Grasp Generation Module.

3.1.2. Grasp Generation Module

This module is composed of two simple parts. If the action it receives from the interpreter is "Pick", the Grasp Generation algorithm described is executed. If, on the other hand, the action received is "place", the algorithm simply translates the current robot position to be directly above the point it received as a placing point, while keeping the orientation of the end-effector (as it supposes the object will be placed on top of the same surface from which it was picked, for simplicity).

Either way, as this system is supposed to be implementation-independent, it feeds the final pose into a Robot Control Unit, responsible for implementing all the interfacing with the robot and performing all the usual robot motion planning checks – Collision detection, singularity avoidance and redundancy resolution. An example implementation of this method was then executed as a proof of concept, which is detailed in the next section.

3.2. Implementation

3.2.1. Hardware setup

Since the main focus of the thesis was in developing the HMI, little focus will be given to the hardware and robot-specific details, as these should ideally be adapted to whatever robotic setup is available at the moment. A very broad overview of the project's infrastructure is, however, provided for the sake of completeness.

In this demonstration, an ASUS G75 personal laptop was used as high-level control unit, a Kinect V2.0 sensor was used as a 3D camera device, the controlled robot was ABB IRB 1400 prototype, FRIDA, present at the Mechatronics and Robotics Laboratory for Inovation (MeRLIn) at Politecnico di Milano, alongside its accompanying infrastructure, *i.e.* the Rosetta computers and connected to them via the OPCOM system. A simplified infrastructural diagram is presented in Figure 19. As it can be seen, the laptop interfaces with the Kinect V2.0 through a USB connection, while the laptop interfaces with the robot through an Ethernet connection, via TCP/IP protocol. Meanwhile, FRIDA interfaces with the Rosetta computers through the OPCOM system, which integrates the robots and the computers in the laboratory. It is also though OPCOM that the Rosetta computers can control the vacuum valves that are used in the manipulator's end-effector. This complicated infrastructure is partly due to the fact that FRIDA is a prototype robot and thus still lacks some features that would be native to its final version, forcing the lab to circumvent the absence of these features, such as embedded vacuum pumps.

In this experimental setup, the Kinect sensor is placed immediately above FRIDA, being tilted by about 42 degrees in order to be able to see the table and the human in front of it at the same time, as can be seen in Figure 20, where Frida is indicated by the blue arrow and the Kinect V2.0 is indicated by the red arrow.

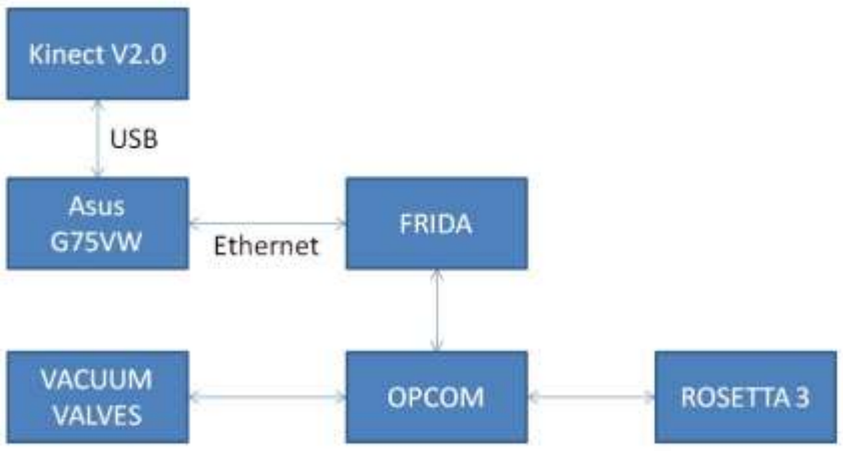


Figure 19- Simplified Infrastructural Diagram

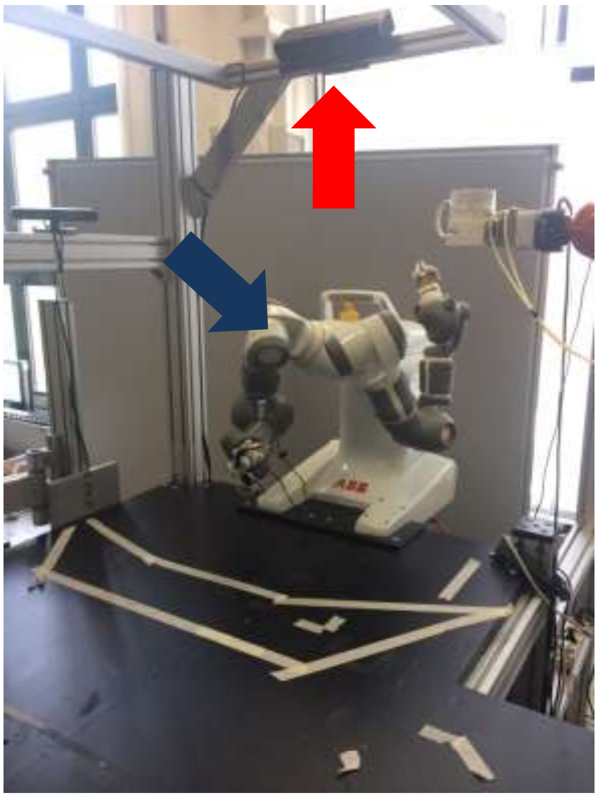


Figure 20 - Experimental Setup with FRIDA and the Kinect

3.2.2. Software Implementation

As mentioned in section 1.2.1, this work aims to develop a simple instructive system that allows users to command a robotic manipulator fixed its position by using gestures captured by the Kinect V2.0 system as a demonstration of the proposed system. This demonstration was coded mostly using MATLAB, with most of the code being compiled into MEX functions through the MATLAB coder add-on [121] in order to improve run-time performance.

Due to time constraints and to the Kinect V2.0 instability in detecting the state of the hands, the system had to be simplified to obey a single task – picking, while the placing task was hard coded to place the picked object into a bin fixed in space. In order to follow the workflow proposed in Figure 15 the following concrete steps must be achieved:

- Interfacing between the Kinect and the computer;
- Developing a system that allows for the interpretation of the user's pointing direction and desired command;
- Developing and applying an efficient image segmentation algorithm to allow for the differentiation of the target object from the background;
- Development of a point-cloud extraction and filtering procedure.
- Developing and implementing a grasp selection algorithm;
- Interfacing with the robot in order to allow it to be controlled by the computer
- Implementing force-feedback on the robot's end-effector to allow for effective suction-cup coupling and usage.

First and foremost, the use-case diagram of the project, though extremely simple, was elaborated in order to properly define the scope of the project. As can be seen in Figure 21, for my current system, there is only one possible use case, that is, the one in which the user points at an object and gives the robot the command to pick it up.

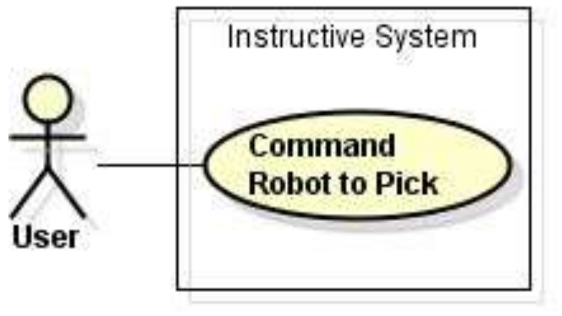


Figure 21- Use case diagram of the instructive system

Then, the workflow of the project was determined. Following the structure proposed in [122] and in [76], the component diagram was created, and is displayed in Figure 22. The system is composed mostly of 6 components: one main component, acting as a centralized controlling unit, one Kinect component, responsible for interfacing the Kinect and the computer, providing it with the color, depth, skeletal and scene point cloud information, each with its own method. The robot component, likewise, is responsible for interfacing the robot to be controlled and the computer. The “Image Sectioner” component is responsible for dividing the color image obtained from the Kinect into different sections, to allow for the identification of the robot’s target. The “Point Cloud Manager”, on the other hand, is responsible for extracting the object’s point cloud when given the full scene point cloud and the user’s 3D skeleton model, filtering this cloud and, finally, performing the coordinate transformations operations necessary to display the point cloud in the same coordinate frame as the robot so that the grasping pose may be generated, each with their own method. The “Grasp Selector” is responsible for generating a grasping pose and for generating the low level instructions for the robot. The “Robot” component, finally, is responsible for communicating the low level instructions to the robot. Each of these components is detailed in the following sections.

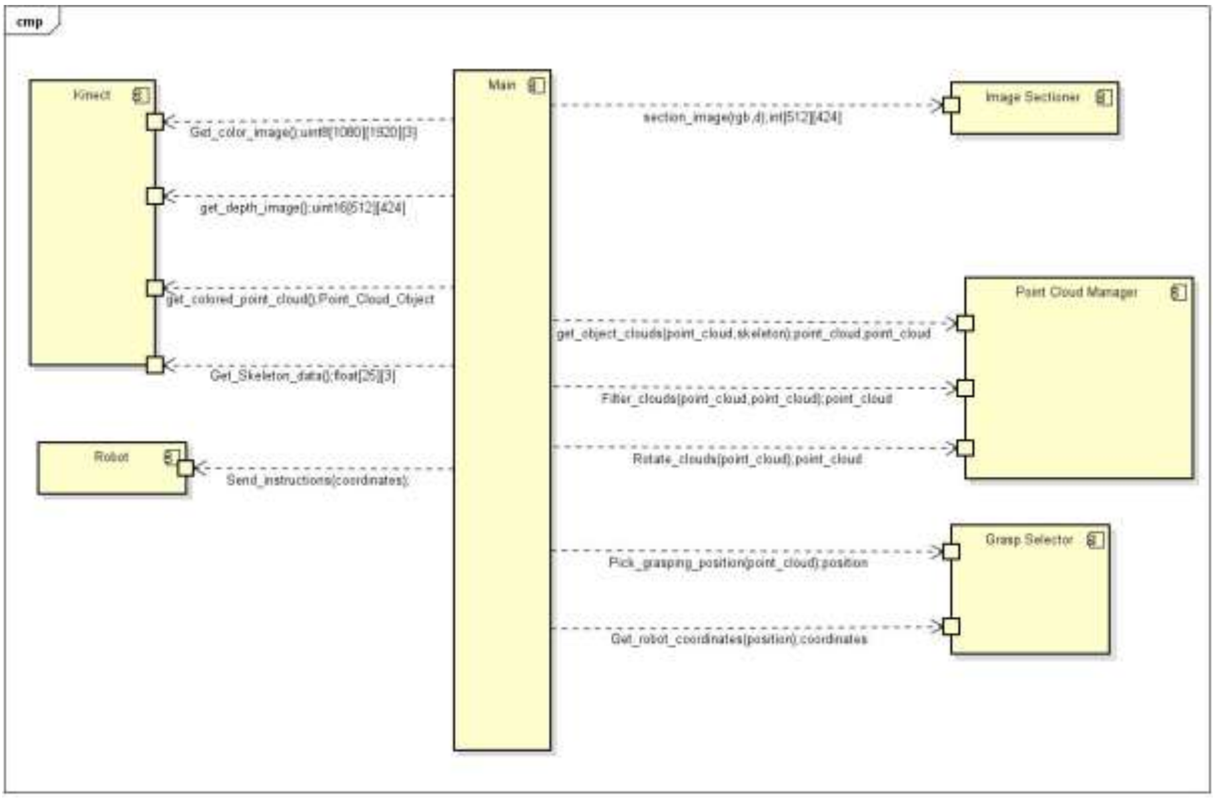

cmp
Kinect
Main
Image Sectioner
Point Cloud Manager
Grasp Selector
Robot
Get_color_image():uint8[1080][1920][3]
get_depth_image():uint16[512][424]
get_colored_point_cloud():Point_Cloud_Object
Get_Skeleton_data():float[25][3]
Send_instructions(coordinates):
section_image(rgb,d):int[512][424]
get_object_clouds(point_cloud,skeleton):point_cloud,point_cloud
Filter_clouds(point_cloud,point_cloud):point_cloud
Rotate_clouds(point_cloud):point_cloud
Pick_grasping_position(point_cloud):position
Get_robot_coordinates(position):coordinates

Figure 22 - Component Diagram for the proposed instructive system

3.2.2.1. Kinect

This component was a simple wrapper for the library Kinect 2 Interface for Matlab, by Juan R Terven and Diana M. Cordova [123], which, in turn, is a MATLAB wrapper for the Kinect V2.0 Development Kit standard implementations in C++. As specified in the library's manual [124], once the k2 object has been instantiated through the command k2 = Kin2('color','depth','body'), a series of other useful methods may be called, which is listed below.

- ***K2.updateData***: Fills an internal buffer with the most recent data from the Kinect and should be called before any other methods to ensure that the data obtained is "fresh"

- ***K2.getDepth***: Obtains the depth image as a 512x424 matrix of floating point numbers, where each point is the distance, in millimeters, from that point to the Kinect's depth sensor
- ***K2.getColor***: Obtains the color image as a 1920x1080x3 matrix of unsigned integers from 0 to 255, each indicating the pixel intensity in the Red, Green and Blue specters.
- ***K2.getPointCloud***: Obtains the point cloud of the scene observed by the Kinect, returning it as a 217088x3 matrix (with depth pixels being listed row-wise, note that 217088 = 512*424), containing one line for each pixel in the depth image and one columns for each coordinate of said point in 3D space (x,y,z) in the coordinate frame of the Kinect, which is defined in Figure 23.
- ***K2.mapColorPoints2Depth(ptColor)***: Takes as input a series of points in the color space (1920x1080) and maps them into the Depth space (512x424).
- ***K2.mapCameraPoints2Depth(ptCam)***: Takes as in input a series of points in 3D and maps them to the camera space.

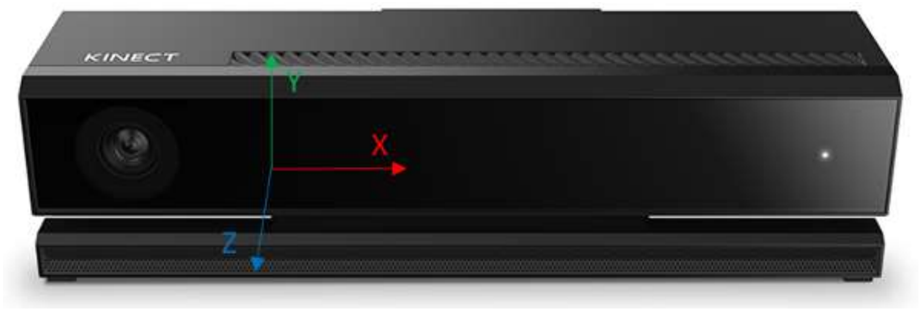


Figure 23 - Kinect V2 Reference Frame [125]

In initial implementations of this component, it used to output purely the color image, the depth image and the point cloud as they were output from the library.

However, in order to implement the image segmentation algorithm, the pixels needed to be all in the same space. In order to do that, a conversion matrix was established between the color and depth spaces. Unfortunately, this conversion is not perfect as the color camera and depth camera have different viewing angles, different positions and different resolutions. Consequently, when the color image is converted to a depth space, some holes and lines appear in the image, as can be seen in Figure 25a), which is a conversion of Figure 24 to depth space. In order to remedy that a bit, a two-dimensional median filtering is applied to the converted image in all of the color channels with a neighborhood of 4x4 pixels. The resulting image can be seen in Figure 25b), where it is clear that the scan lines are gone, at the expense of some fine details in the image. The greater problem remains that there are still points in the converted image that could not have any color associated to them. No solution, however, was found in time for this issue and- as the main area where the objects would be placed remained mostly unaffected by this phenomenon, the author decided to overlook this issue. The same filtering process is applied to the depth image, in order to remove dead pixels, smooth out the image and filter out noise. The median filtering in the depth image is performed, however, with a 3x3 neighborhood definition. The results of this filtering can be seen in Figure 26.

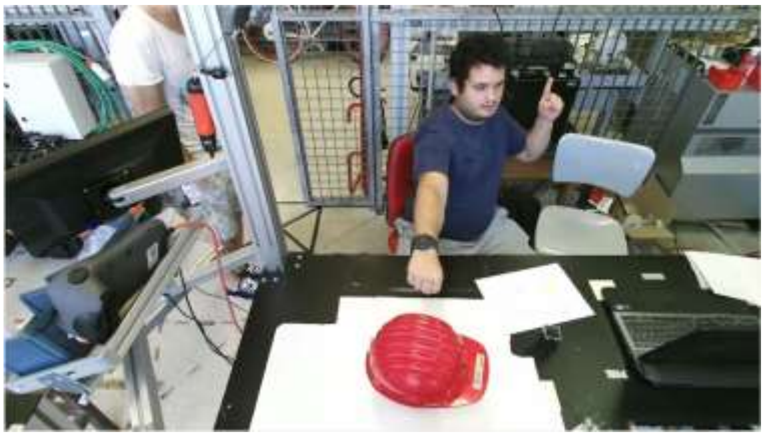


Figure 24 - Original Color Image

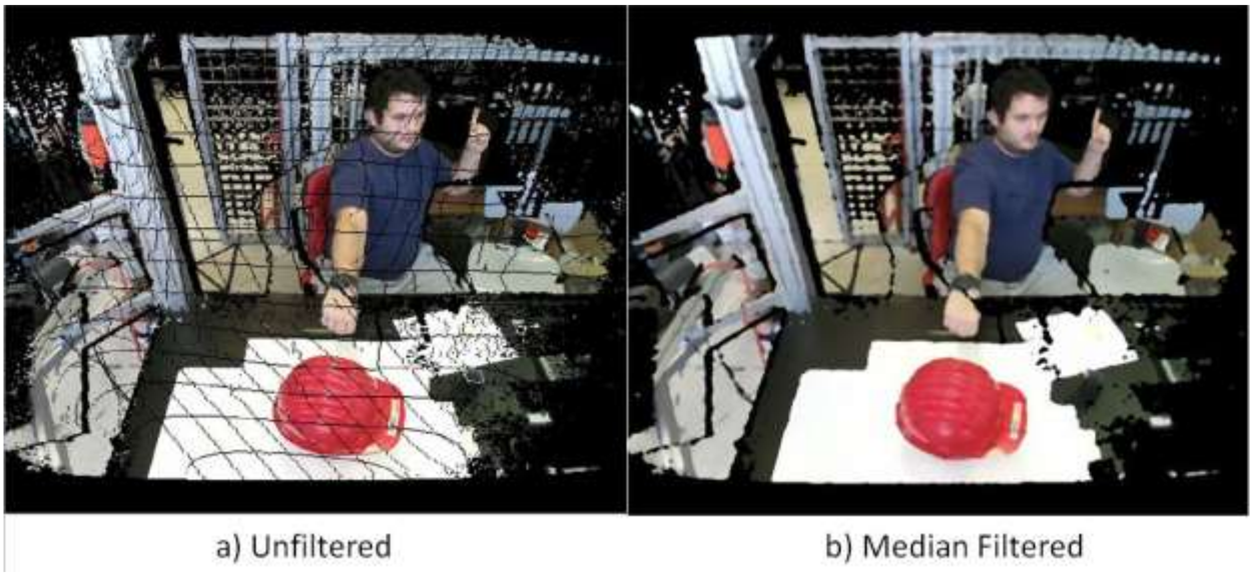


Figure 25 - Color Images Converted to Depth Space

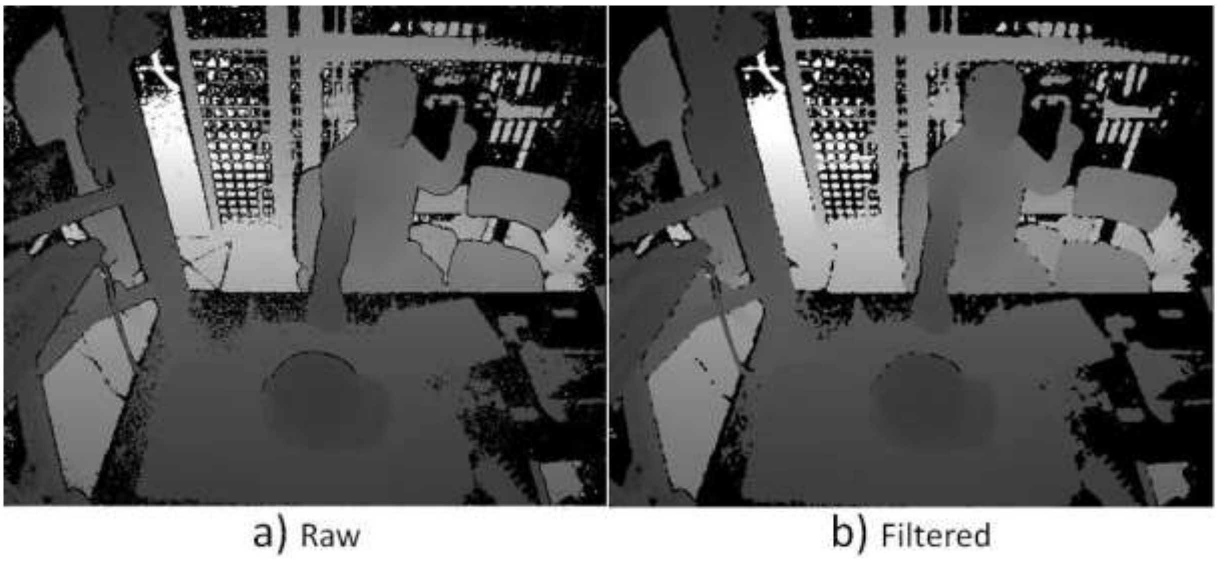


Figure 26 -Depth Image a) before and b) after filtering

3.2.2.2. Image Sectioner

As detailed in section 3, in order to properly identify the object to be grasped the HMI must be able to properly distinguish between the visual elements present in the scene. As discussed in section 2.4, image segmentation is still an active area of research, yet, for the reasons exposed in the same chapter, the altered FH algorithm [108], [112] was chosen and is presented in detail below.

Felzenszwalb and Huttenlocher's (FH) algorithm [108]

Let each pixel in an image I be considered as a vertex $\boldsymbol{v} \in \mathbf{V}$, set of all the vertices of a graph **G**. Connect each vertex to its nearest neighbors by a weighted edge $\mathbf{e} \in \mathbf{E}$, set of all edges of the graph **G**. The weight w_e of an edge e between verticies v_i and v_j can be defined in a variety of ways, but [108] and [112]'s are of particular interest in this context. Let a pixel be represented by its RGB-D values as an array of form [R,G,B,D]. Then, the weight w_e is defined generically in Equation 21, where **k1**, **k2**, **k3** and **k4** are weighting terms responsible for measuring the importance of each of the information channels with respect to one another. In their implementation, [112] found that **k1** = **k2** = **k3** = 0.3 and **k4** = 0.7 resulted in the best performance for the algorithm, and, as such, these parameters are the ones used in this work.

Equation 21- Generic Edge Weight Definition

$$w_e = \left\| [k1, k2, k3, k4] * \left[R_i - R_j, G_i - G_j, B_i - B_j, D_i - D_j\right]^T \right\|$$

We must then define a few key concepts. A component **C** is a subset of **V**. The internal difference of a component **C** is defined as being the largest weight in the minimum spanning tree of the component**, MST(C,E)**, as shown in Equation 22. Meanwhile, the difference between two disjoint components **C_1**,**C_2**, **Dif(C_1,C_2)** is defined as being the weight of the minimum weighted edge connecting the two components, formally defined in Equation 23, where **w(v_i,v_j)** stands for the weight of the edge connecting vertices i and j. Finally, the minimum internal difference between two components **C_1**,**C_2**, **Mint(C_1,C_2)** is defined in Equation 24, where $\tau(C) = \frac{k}{|C|}$, in which k is an arbitrary constant. The term $\boldsymbol{\tau(C)}$ is a threshold function that controls the degree to which the difference between two components must be greater than their internal differences in order for there to be evidence of a boundary between them and is necessary because for small components the internal difference **Int(C)** is not a good estimate of the local characteristics of the data, making it necessary that the slack term τ be introduced. The normalization of the threshold by **|C|**, however, guarantees that as the components grow larger, this slack term becomes insignificant, since the **Int(C)** starts being a proper estimate of the data's local characteristics.[108].

Equation 22 - Definition of internal difference

$$Int(C) = \max_{e \in MST(C,E)} w_e$$

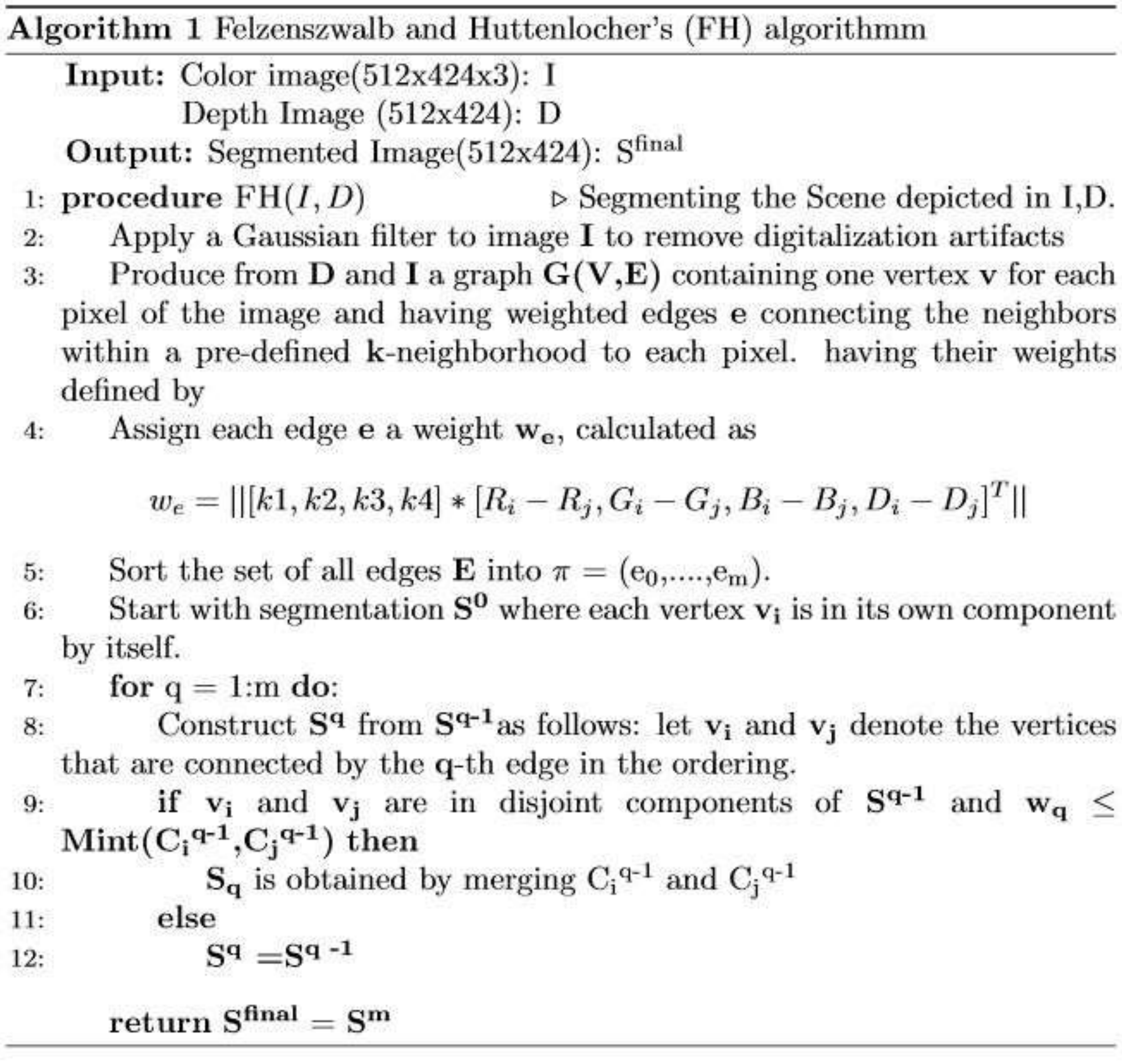
Equation 23- Definition of difference between Components

$$Dif(C_1, C_2) = \begin{cases} \min\limits_{v_i \in C_1, v_j \in C_2, (v_i, v_j) \in E} w(v_i, v_j), & \exists\, (v_i, v_j) \mid v_i \in C_1, v_j \in C_2, (v_i, v_j) \in E \\ +\infty, & else \end{cases}$$

Equation 24 - Definition of the minimum internal difference MInt

$$MInt(C_1, C_2) = \min((Int(C_1) + \tau(C_1), (Int(C_2) + \tau(C_2))$$

The segmentation algorithm may then be described in pseudo-code in Figure 27 , as in [108] :



Algorithm 1 Felzenszwalb and Huttenlocher's (FH) algorithmm

```
Input: Color image(512x424x3): I
       Depth Image (512x424): D
Output: Segmented Image(512x424): $S^{final}$
1: procedure FH($I, D$)                ▷ Segmenting the Scene depicted in I,D.
2:     Apply a Gaussian filter to image I to remove digitalization artifacts
3:     Produce from D and I a graph G(V,E) containing one vertex v for each
   pixel of the image and having weighted edges e connecting the neighbors
   within a pre-defined k-neighborhood to each pixel.  having their weights
   defined by
4:     Assign each edge e a weight $w_e$, calculated as

       $w_e = ||[k1, k2, k3, k4] * [R_i - R_j, G_i - G_j, B_i - B_j, D_i - D_j]^T||$

5:     Sort the set of all edges E into $\pi = (e_0,....,e_m)$.
6:     Start with segmentation $S^0$ where each vertex $v_i$ is in its own component
   by itself.
7:     for q = 1:m do:
8:         Construct $S^q$ from $S^{q-1}$as follows: let $v_i$ and $v_j$ denote the vertices
   that are connected by the q-th edge in the ordering.
9:         if $v_i$ and $v_j$ are in disjoint components of $S^{q-1}$ and $w_q \leq$
   $Mint(C_i^{q-1},C_j^{q-1})$ then
10:            $S_q$ is obtained by merging $C_i^{q-1}$ and $C_j^{q-1}$
11:        else
12:            $S^q = S^{q-1}$
       return $S^{final} = S^m$
```

Figure 27 - Pseudo-code for the FH algorithm

An brief intuition of this algorithm can be provided in layman’s terms: This algorithm merges groups of pixels that present a connection between them that has a weight lower than the maximum weight already present in either of the groups- *i.e.*- if the groups were to be merged, the maximum weight between its vertices would remain inaltered. The algorithm, however, provides a relaxation term at the beginning, since at that point all pixels are isolated and, thus, no component has any weights and, therefore, in order to start joining pixels together, one must establish a minimum starting criterion.

This algorithm is rather versatile, as it possesses both robustness to noise and to smooth variations in tone and depth [108]. In the seminal paper, the authors suggest values of k between 150 and 300. This hard-coded hyperparameter was faced as an initial challenge by me and I empirically determined that setting the factor k as 10 times the mean value of all the edges in the graph **(k = 10μ(w(E)))** yielded the best results, regardless of setting (both in training datasets, such as the NYU depth dataset [113] and the OSD dataset [114] and during the lab experiments). Though this evaluation was mostly qualitative, the author would like to recommend further inquiry into this problem, possibly posing it as a Gaussian process of hyperparameter optimization by using the 5 metrics mentioned in the beginning of this section [115], as finding a threshold **k** that varies automatically according to the complexity of the scene could be of great value to the field of computer vision.

Implementation details of the FH algorithm

The image sectioned module is a simple implementation of the algorithm described above, with a few minor alterations. In this application, the neighborhood of a pixel, *i.e.*, the pixels to which the current pixel is connected by edges was defined as any pixel within a 5*5 grid of the pixel, as can be seen in Figure 28. This definition of neighborhood was chosen in order to avoid that small gaps in pixels due to faulty resolution conversion (as seen in Figure 25) or due to reflective surfaces in the depth images would disrupt the stability of the segmentation algorithm. When calculating the edge weights, the biasing vector suggested in [112] was used, [0.3,0.3,0.3,0.7], for the red, green, blue and depth channels respectively.

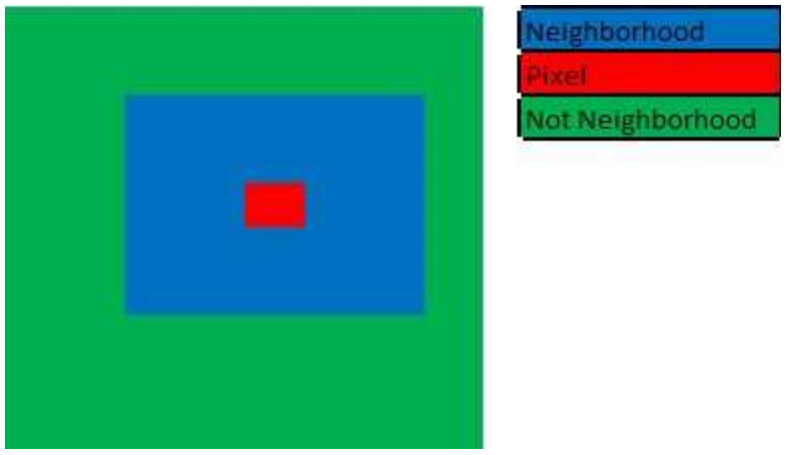


Figure 28 - Pixel neighborhood definition

In addition, all pixels belonging to segments that are considered too small (this was a manually set threshold of 30 pixels) were considered as belonging to the same segment – segment 0, the ignored segment, as these small segments are likely a product of the artifacts cause by color image conversion to depth space. Additionally, the parameter k was set to 10 times the average weight of the edges in the image graph for the reasons exposed in section 0.

This module receives as input the color image converted to depth space and the depth image and outputs two things: a matrix with the same resolution as the depth image (512x424) containing the segment to which each pixel belongs (result of the segmentation) and a colored image that is a visual representation of these segmentations, as the one shown in Figure 29. It can be seen that the segmentation quality is rather good near the center of the image and decreases in quality as it approaches the borders. That is a direct consequence of the lack of color information in the borders that is a result from the conversion of the color image to the depth space. Yet, in the crucial area- that is- the area within the robot's reach- the segmentation is satisfactory for our purposes. As a final implementation detail, both images received in the output are then scaled by a factor of 0.5 in order to optimize run time performance while still maintaining accuracy.

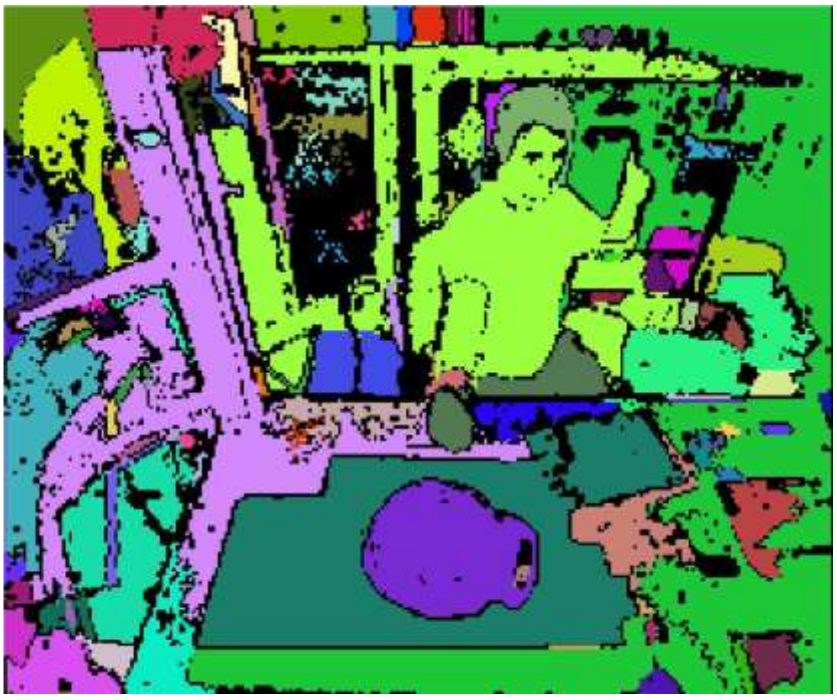


Figure 29 - Test Segmentation of the scene in Figure 24

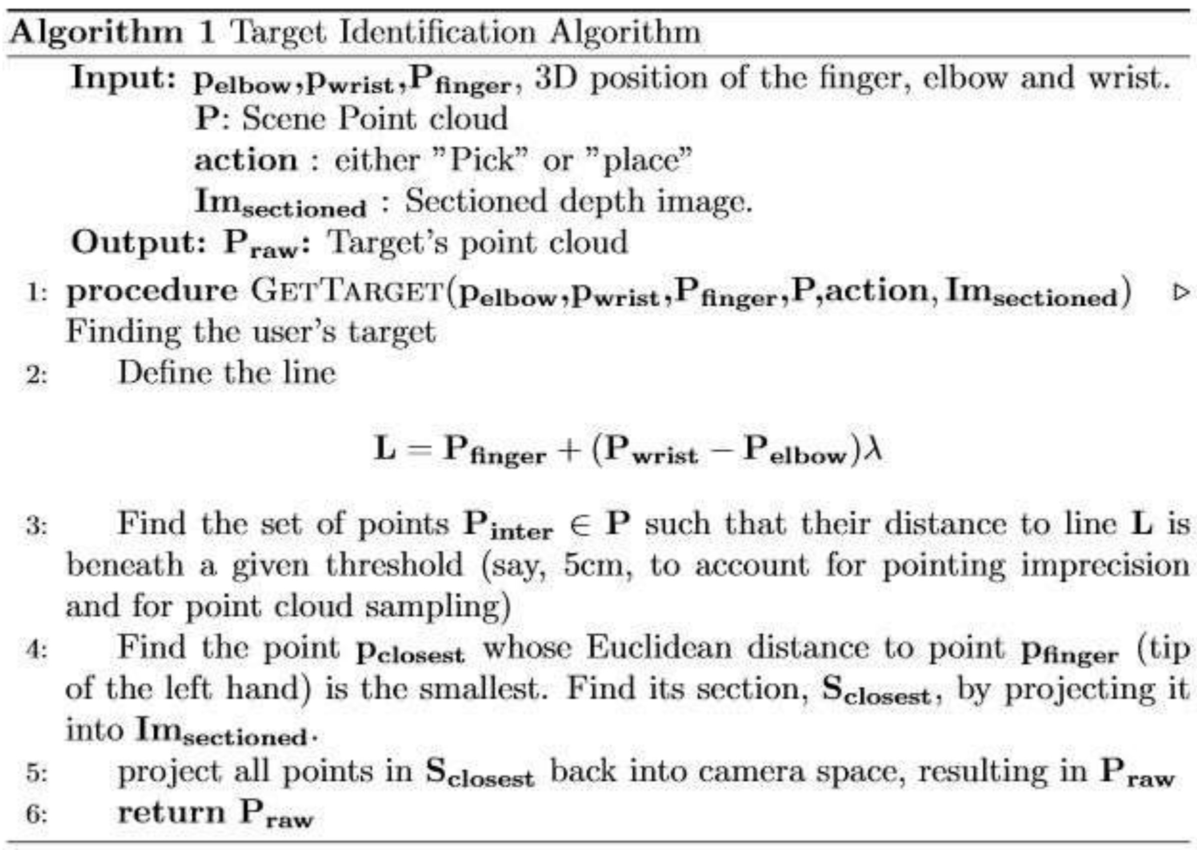
3.2.2.3. Point Cloud Manager

This object is responsible for extracting, filtering and rotating the object's point cloud so that it is in the same reference frame as the robot, each of these roles being fulfilled by a different software component, detailed below.

get_object_clouds

This function is responsible for extracting two point clouds from the scene – The point cloud corresponding to the section of the object being indicated and an extended point cloud, which is basically a bounding box around this section. It receives as input the entire point cloud, the segmented image and the skeleton 3D model.

The first step in this algorithm lies in identifying the intersection between the line inferred by the user's left forearm and the scene's total point cloud, being a simple implementation of the algorithm described in section 3.1.1, with slight modifications, being reported in pseudo-code in Figure 30.



```
Algorithm 1 Target Identification Algorithm
    Input: $\mathbf{p_{elbow}}$,$\mathbf{p_{wrist}}$,$\mathbf{P_{finger}}$, 3D position of the finger, elbow and wrist.
           $\mathbf{P}$: Scene Point cloud
           action : either "Pick" or "place"
           $\mathbf{Im_{sectioned}}$ : Sectioned depth image.
    Output: $\mathbf{P_{raw}}$: Target's point cloud
1: procedure GetTarget($\mathbf{p_{elbow}}$,$\mathbf{p_{wrist}}$,$\mathbf{P_{finger}}$,$\mathbf{P}$,action, $\mathbf{Im_{sectioned}}$)    ▷ Finding the user's target
2:     Define the line
           $\mathbf{L} = \mathbf{P_{finger}} + (\mathbf{P_{wrist}} - \mathbf{P_{elbow}})\lambda$
3:     Find the set of points $\mathbf{P_{inter}} \in \mathbf{P}$ such that their distance to line $\mathbf{L}$ is beneath a given threshold (say, 5cm, to account for pointing imprecision and for point cloud sampling)
4:     Find the point $\mathbf{p_{closest}}$ whose Euclidean distance to point $\mathbf{p_{finger}}$ (tip of the left hand) is the smallest. Find its section, $\mathbf{S_{closest}}$, by projecting it into $\mathbf{Im_{sectioned}}$.
5:     project all points in $\mathbf{S_{closest}}$ back into camera space, resulting in $\mathbf{P_{raw}}$
6:     return $\mathbf{P_{raw}}$
```

Figure 30 – Target Identification Algorithm

An example of the raw point cloud returned by applying this algorithm to the scene in Figure 24 is seen in Figure 31 (note that the axes have a weird orientation due to the pitch angle of the sensor with respect to the table). One must note that for the most part the structure of the helmet has been captured – besides the already expected misdetections due to image space conversions and noise in the depth sensors. One must also note, however, that due to the abrupt color transition in a region of smooth depth gradient, the white sticker in the cap brim was put in a different section than that of the cap – and this generates a non-existing hole in the brim. One could be tempted to suggest solving this issue by means of a flood-fill algorithm. However, this segmentation hole could be due to an actual hole in the cap – and the flood fill operation in the segmented image would actually deform the true object's point cloud. Thus, a different solution was proposed.

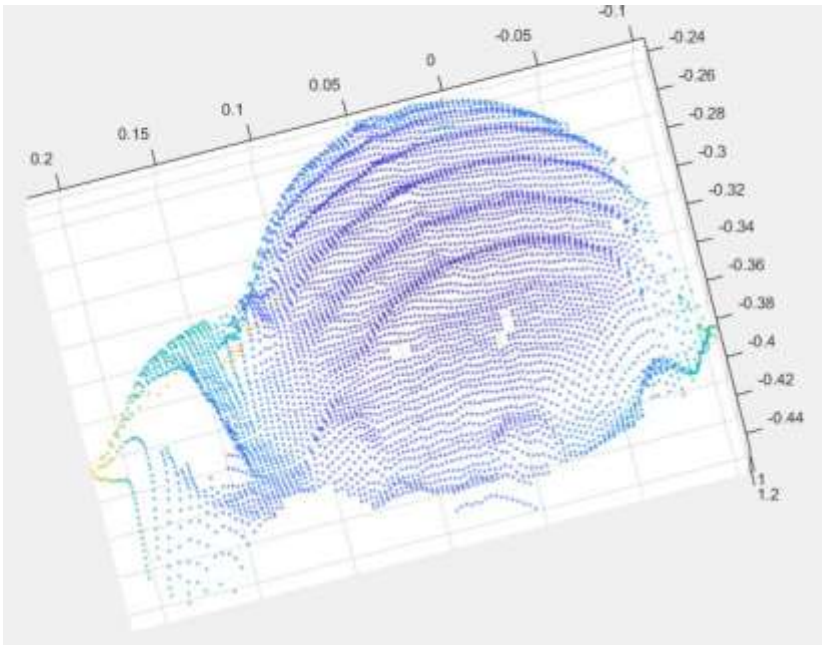


Figure 31- Raw red helmet extracted point cloud

In order to solve this issue, an extra step was added to the point cloud extraction algorithm: Now, instead of just returning the points that belong to the same segment as the point being indicated, we now take the maximum and minimum indices of the pixels in this section and create an extended bounding box for them. That is, let (x_{max},y_{max},y_{min},x_{min}) denote the extrema of the coordinates of all the pixels that belong to segment $S_{closest}$. The bounding box can then be formally defined as $\mathbf{p}(\mathbf{x},\mathbf{y})|\ \mathbf{x_{min}} \leq \mathbf{x} \leq \mathbf{x_{max}}\ ; \mathbf{y_{min}} \leq \mathbf{y} \leq \mathbf{y_{max}}$. The pixels contained within this bounding box can then be projected unto a 3D point cloud, named $\boldsymbol{P_{extended}}$. We then return both the pure set of segmented pixels, hereby named $\boldsymbol{P_{raw}}$, as seen in Figure 31 and $\boldsymbol{P_{extended}}$, shown in Figure 32. Note that the extended point cloud has many points that clearly do not belong to the hat, as was expected, yet restores some points that had been erroneously removed from it in the raw point cloud. That is why these two point clouds will be used in the filtering steps in order to extract the best possible 3D point cloud of the object to be picked, with the filtering steps to be performed being detailed in the next section.

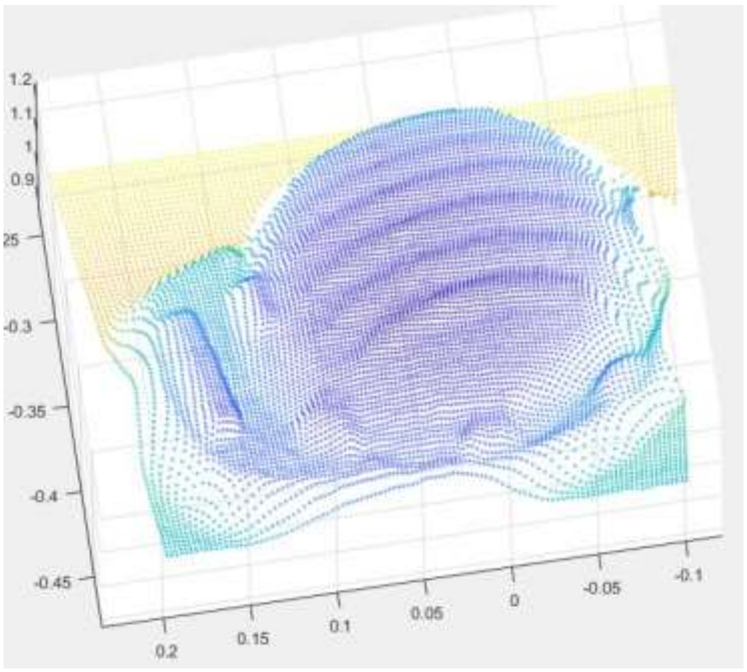
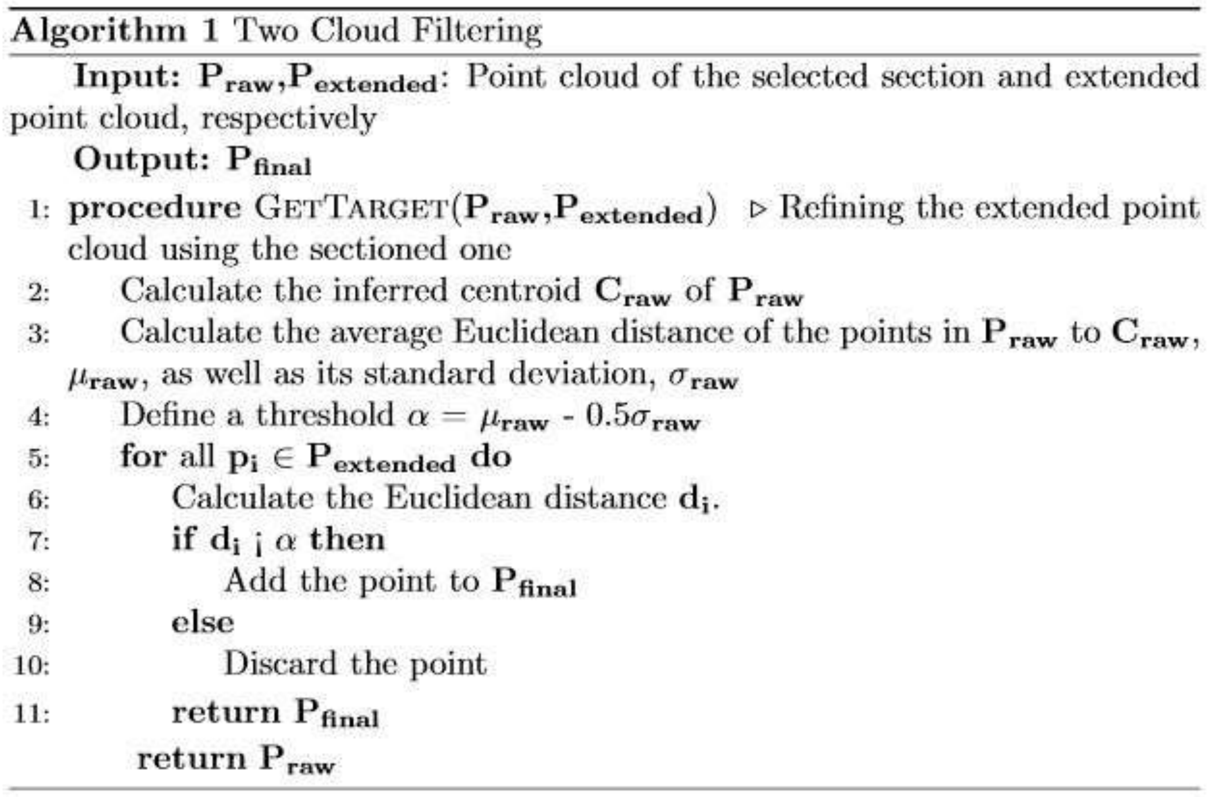


Figure 32 - Extended point cloud

filter_clouds

This method is responsible for clearing out the misdetections and odd points from the detected point clouds, being equivalent to the filtering step proposed in [76] and described in section 2.3.4.2, with one added step. The filtering step that was added is now described.

As noted in the previous section, due to segmentation errors, some parts of the object tend to be put in different sections than that of the object to which they belong. In order to remedy that, a new filter is proposed. This filtering technique is based on the fact that the raw point cloud contains a series of false negatives, but most points contained within it are true positive identifications, while the extended point cloud has many false positives, but probably contains the entire object. Additionally, we know that if a point belongs to the object, it obeys some sort of spatial continuity. Thus, one may follow the following heuristic to filter the point clouds: Calculate the inferred Centroid of the raw point cloud. Then, for each point in the extended point cloud, calculate its distance to the inferred centroid of the raw point cloud. It this distance is within the mean distance of points in the raw point cloud to their respective centroid, keep this point in the final cloud. Otherwise, discard it. The algorithm can then be more formally define in pseudo-code in Figure 33.



```
Algorithm 1 Two Cloud Filtering
    Input: $\mathbf{P_{raw}}$,$\mathbf{P_{extended}}$: Point cloud of the selected section and extended point cloud, respectively
    Output: $\mathbf{P_{final}}$
procedure GETTARGET($\mathbf{P_{raw}}$,$\mathbf{P_{extended}}$)    ▷ Refining the extended point cloud using the sectioned one
    Calculate the inferred centroid $\mathbf{C_{raw}}$ of $\mathbf{P_{raw}}$
    Calculate the average Euclidean distance of the points in $\mathbf{P_{raw}}$ to $\mathbf{C_{raw}}$, $\mu_{\mathbf{raw}}$, as well as its standard deviation, $\sigma_{\mathbf{raw}}$
    Define a threshold $\alpha = \mu_{\mathbf{raw}}$ - $0.5\sigma_{\mathbf{raw}}$
    for all $\mathbf{p_i} \in \mathbf{P_{extended}}$ do
        Calculate the Euclidean distance $\mathbf{d_i}$.
        if $\mathbf{d_i}$ ¡ $\alpha$ then
            Add the point to $\mathbf{P_{final}}$
        else
            Discard the point
        return $\mathbf{P_{final}}$
      return $\mathbf{P_{raw}}$
```

Figure 33 - Point-cloud filtering algorithm using two point-clouds

The results of this filtering step are shown in Figure 34. The first picture on the left being ***$P_{extended}$***, the middle one being ***$P_{filtered}$*** and the one on the right being the resulting point cloud after performing the additional filters suggested by Ancona (Density and Uniformity) and rotating and translating the point cloud so that it would be in robot’s coordinate frame. One should note that the resulting point cloud is not free from noise, but is of much better quality than that of Figure 31.

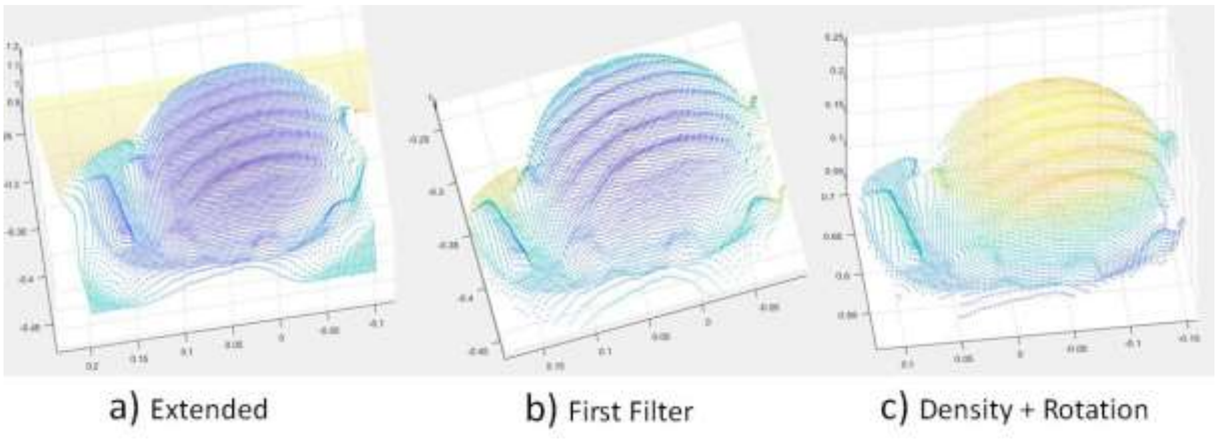


Figure 34 - Point cloud progression

Rotate_clouds

This method is a simple implementation of traditional reference frame rotation and translation from robotics literature. That is, let V^R denote a vector in the robot's frame and let V^K denote a vector in the Kinect's frame. Then, this module simply implements the Homogeneous Transformation $V^R = T_K^R V^K$ for all points in the points in the point cloud. In this particular setting, the transformation was comprised of 3 sequential rotations: One of -42 degrees around the Kinect's x axis, one rotation of -90 degrees around y', one rotation around x'' of -90 degrees and one translation of $[0.09\ 0.0603\ 1.0591]^T$in the robot's frame, resulting in this final transformation matrix expressed in Equation 25.

Equation 25 - Definition of the homogeneous transformation

$$R_K^R = \begin{pmatrix} 0 & 0.6691 & 0.7431 \\ -1 & 0 & 0 \\ 0 & .7431 & -0.6691 \end{pmatrix} ; O_K^R = \begin{bmatrix} 0.09 \\ 0.0603 \\ 1.0591 \end{bmatrix} ; T_k^R = \begin{pmatrix} R_K^R & O_K^R \\ 0 & 1 \end{pmatrix}$$

3.2.2.4. Grasp Selector

This module is responsible for selecting the optimal grasping position – and providing the appropriate low level instructions for the robot, each implemented in their own function, reported in detail below.

Pick_grasping_position

This module implements a grasp generation algorithm that uses the two common strategies used in grasp syntheses simultaneously: it tries to grab the object at the most planar surface, while also trying to grasp it near its center of mass. The algorithm is described in detail below.

Grasp Generation Algorithm

After acquiring the object's point cloud, filtering and smoothing it, one must then calculate the estimated surface normals of the object, possibly by means of the Least Squares algorithm, as described by [76]. Once the surface normals have been calculated, let $\boldsymbol{N_i^R}$ denote the set of points within a neighborhood of a given point $\boldsymbol{p_i}$ of radius ***R***. Let ***R*** be slightly bigger than the radius of the gripper, $\boldsymbol{R_{Gripper}}$, as $\boldsymbol{R = 1.5 * R_{Gripper}}$ and let $\boldsymbol{n_j}$ denote the estimated surface normal of a point $\boldsymbol{p_j}$ within $\boldsymbol{N_i^R}$. We can then calculate $\overline{\boldsymbol{T_i^2}}$, the average squared cosine of the angle between the surface normals within $\boldsymbol{N_i^R}$ according to Equation 26, if one remembers that the surface normals are unit vectors.

Equation 26 - Definition of $\overline{\boldsymbol{T_i^2}}$

$$\overline{T_i^2} = \sum_{j \in N_i^R} (n_i . n_j)^2$$

Then, one may pick the k most uniform points – *i.e.* – the k points I whose $\overline{\boldsymbol{T_i^2}}$ is closest to 1 (since $\cos(0)^2 = 1$) and pick the point $\boldsymbol{p_j}$ among them whose distance to the inferred centroid of the object $\boldsymbol{C_{inf}}$ is the smallest. The grasping pose, thus is one centered at this point, with the suction cup parallel to the average normal vector within a vicinity of $\boldsymbol{R_{Gripper}}$ of it. This algorithm can then be succinctly described in pseudo-code in Figure 35:

```
Algorithm 1 Grasp Generation for Suction Cup Grippers
    Input: Point Cloud: $\mathbf{P}$
           Radius of gripper: $\mathbf{R_{gripper}}$
    Output: Grasping Position (x,y,z): $\mathbf{p_{best}}$
            Grasping Normal Vector (x,y,z) : $\mathbf{V_{grasping}}$
 1: procedure GENERATEGRASPINGPOSE($P, R_{gripper}$)    ▷ Picking the best point and orientation to grasp the object
 2:     Apply uniformity and density filters, obtaining $\mathbf{P_{filtered}}$
 3:     Calculate the inferred centroid $\mathbf{C_{inf}}$ of $\mathbf{P_{filtered}}$
 4:     for all $\mathbf{p_i}$ in $\mathbf{P_{filtered}}$ do
 5:         Calculate $\mathbf{N_i}^{\mathbf{R}}$, set of neighbors of point i within a neighborhood $\mathbf{R}$
 6:         if $\mathbf{N_i}^{\mathbf{R}}$ ¡ $\mu(|\mathbf{N_i}^{\mathbf{R}}|)$ - $2\sigma(|\mathbf{N_i}^{\mathbf{R}}|) = \eta$ then
 7:             Set $T^{\mathbf{2}}{}_{\mathbf{i}} = 0$ , to eliminate points too close to the borders.
 8:         else
 9:             Calculate $T^{\mathbf{2}}{}_{\mathbf{i}}$ as
                    $$T^{\mathbf{2}}{}_{\mathbf{i}} = \sum_{n \in \mathbf{N_i}^{\mathbf{R}}} (\mathbf{n_i}.\mathbf{n_j})^2$$
10:     Order points according to their $T^{\mathbf{2}}{}_{\mathbf{i}}$ score and pick the k points with the highest score, possibly applying a minimum score threshold to avoid spots that are too uneven ( if your object has very few points), composing $\mathbf{K}$
11:     for all points $\mathbf{p_m} \in \mathbf{K}$ do
12:         calculate $\mathbf{d_m}$ = ——
13: textbfp$_m$ - $\mathbf{C_{inf}}$——
14:     pick the point $\mathbf{p_{best}}$ whose $\mathbf{d_{best}}$ is the minimum distance.
15:     Define $\mathbf{N_{best}}^{\mathbf{R_{gripper}}}$, set of neighbors of point i within a neighborhood $\mathbf{R_{gripper}}$
16:     calculate
                    $$\mathbf{V_{grasping}} = \sum_{i \in \mathbf{N_{best}}^{\mathbf{R_{gripper}}}} n_i$$
17:     Normalize $\mathbf{V_{grasping}}$ as
                    $$\mathbf{V_{grasping}} = \frac{\mathbf{V_{grasping}}}{||\mathbf{V_{grasping}}||}$$
        return $\mathbf{V_{grasping}}$ and $\mathbf{p_{best}}$
```

Figure 35 - Suction Gripper Grasp Generation Algorithm

The gripping pose to be used, thus, is defined by the point p_{best} and the direction parallel to the average normal vector.

The ***pick_grasping_position*** is simply an implementation of the algorithm presented in section 2.5 with the radius ***R*** being set to 0.01, the minimum number of neighbors ***η*** being determined by the average number of neighbors within a vicinity of radius ***R*** minus one half of the standard deviation of the number of neighbors within the radius and the number of smoothest points ***K*** being set to 25. The algorithm is otherwise unaltered. After running this algorithm on the filtered point cloud obtained in the previous steps in the scene presented in Figure 24, the algorithm selects the grasping point represented by the red arrow in Figure 36. It can be seen that the algorithm has selected one of the smoothest points to grasp the hat, with the orientation being normal to the surface.

One may criticize the chosen point due to it being far from the actual center of gravity of the hat. That is true and is likely to be caused by the biased view the algorithm has of the object (there are more points towards the front and in the superior part, and thus, the estimated centroid is skewed in those directions). One way to solve this issue would be to implement the centroid estimate proposed by [76] , which uses the extrema of the point cloud to estimate the centroid. This was not implemented due to a programming oversight. However, during the tests this oversight did not seem to significantly impact the robustness of the grasp, as will be seen in section 4. This is probably due to the fact that the selected points are still fairly close to the actual center of gravity of the objects and to the fact that the suction cup was reasonably overpowered for the objects it was being used to handle.

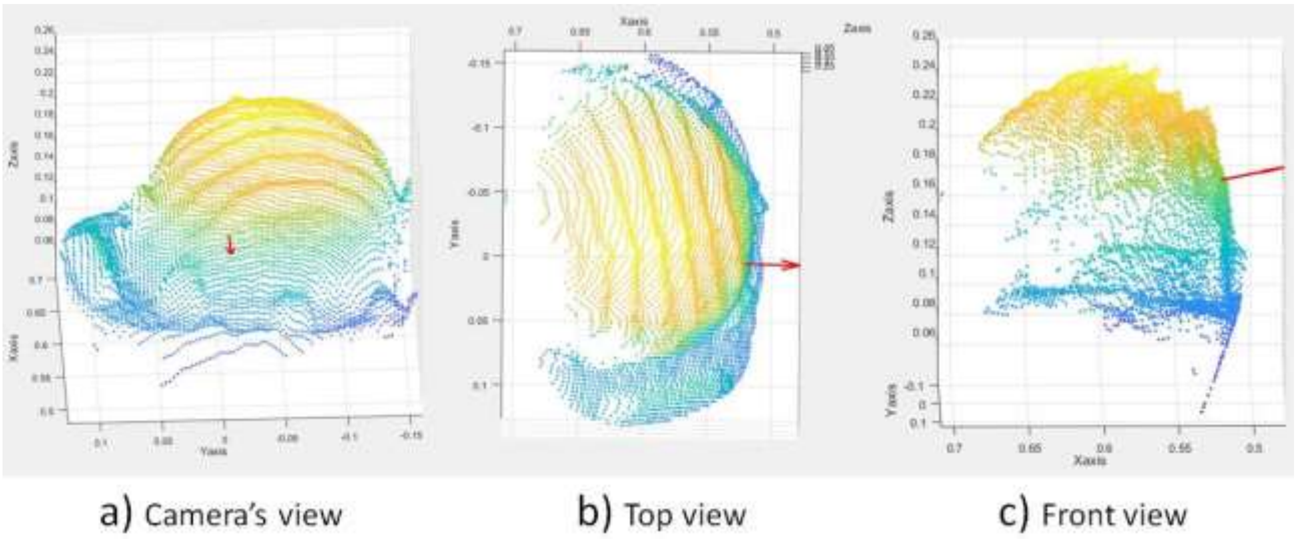


Figure 36 - Several views of the grasping pose

Get_robot_coordinates

The output of the ***pick_grasping_position*** method is the 3D coordinates of the grasping point and the estimated normal vector of the object's surface at that point. However, FRIDA's programming language, ABB RAPID, requires quaternions to fully define the orientation of a pose, which, in turn, require a full coordinate system to be defined in order to be obtained. Luckily, the suction cup gripper is axisymmetric and, thus, once we define the approach axis (Z) as the surface's normal vector at the approaching stance, we are free to choose any arbitrary X and Y axis for the end effector. For this specific project, the arbitrary X and Y axis were chosen so that the γ component of the Euler angles of the end-effector's frame with respect to the robot would be zero. With that assumption, the remaining Euler angles were calculated (α,β) according to Equation 27, where ***(X*** (X_n, Y_n, Z_n) is the normal vector expressed in the robot's coordinate frame.

Equation 27 -Euler angles, given γ is zero

$$\alpha = \arctan\left(\frac{Y_n}{X_n}\right), b = \arccos(Z_n)$$

From these angles, the rotation matrix for the goal end-effector's frame as calculated as shown in Equation 28 and the quaternions for this frame can be calculated as shown in Equation 29.

Equation 28- Rotation matrix of the End Effector

$$R^{R}_{E_{eff}} = \begin{pmatrix} \cos(\alpha)\cos(\beta) & -\sin(\alpha) & cos(\alpha)\sin(\beta) \\ \sin(\alpha)\cos(\beta) & \cos(\alpha) & \sin(\alpha)\sin(\beta) \\ -\sin(\beta) & 0 & \cos(\beta) \end{pmatrix} = \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{pmatrix}$$

Equation 29- Quaternions from rotation matrix

$$\begin{cases} q_1 = \dfrac{\sqrt{x_1 + y_2 + z_3 + 1}}{2} \\ q_2 = \dfrac{sign(y_3 - z_2)\sqrt{x_1 - y_2 - z_3 + 1}}{2} \\ q_3 = \dfrac{sign(z_1 - x_3)\sqrt{y_2 - x_1 - z_3 + 1}}{2} \\ q_4 = \dfrac{sign(x_2 - y_1)\sqrt{z_3 - x_1 - y_2 + 1}}{2} \end{cases}$$

The contact point and the quaternions that defined the end-effector frame were then returned.

3.2.2.5. Robot

On the robot software side, there were three main components running, though only one of them directly interfaced with the laptop. These were, in order: The communicator, the robot's main program and the force estimator, explained in further detail below.

The Communicator

This module is a part of the main program running on the laptop and is responsible for establishing a TCP/IP connection with FRIDA using an Ethernet cable. This module simply sends the 3 coordinates of the point slightly above the grasping point and the 4 quaternions defining the orientation of the end-effector. It is implemented using the tcpclient [126] function from MATLAB.

Robot’s Main

A brief description of the ABB RAPID code running on FRIDA during the whole experiment in pseudo-code is given. The main idea of this code is that it waits until a TCP/IP connection is established, then starts a loop where it receives the coordinates of the approximation point for an object to be manipulated, the moves to that position, starts lowering the end effector in the given orientation until it detects contact with the object (as a signal from the force estimator), point in which it activates the suction cup to grasp the object. In this demonstration, due to time constraints, the robot was then hard-coded to move to a position immediately above a bin, located to the right-hand side of the robot and drop the gripped object. The robot should then return to its waiting position and wait for new instructions to arrive via the TCP/IP connection. One must note that, as mentioned previously, due to time constraints, no obstacle avoidance, fine motion planning, singularity avoidance or complex redundancy resolution were implemented – being either ignored (such as motion planning and obstacle avoidance), while redundancy resolution and singularity avoidance were solved by hard-coding a solution (by fixing the configuration of the joints and making sure FRIDA would not enter a singular position during operation.

Force Estimator

Since the suction gripper requires the seal to be closed, there has to be some level of contact between the cup and the object before one opens the vacuum valves. One naïve way to solve this problem would be to position the suction cup a few millimeters “inside” the point cloud and then activate the vacuum gripper. However, there is one main issue with this approach: First of all, the 3D point cloud obtained from the object has a high degree of uncertainty and, thus, establishing the minimum displacement necessary to effectively touch the object would be a complex and uncertain procedure. A graver issue, though, is presented when this distance is overestimated. In this critical failure, the robot will try to push into the object too far, possible damaging both the robot and the object. Thankfully, FRIDA is a collaborative robot and will thus enter into a protective stop should it detect a big and sudden external torque, such as one provoked by a collision. This, however, presents another issue: Even if the offset is correctly identified, the force resulting from touching the object could send FRIDA into a protective stop. This solution is, therefore, not viable.

An alternative solution, slightly more complex, would be place a force sensor at the end-effector of the robot. Though this option was preferable (as it is more accurate and simpler to implement) this option was not available. Thankfully the lab possessed a module that used a state observer to estimate external torques acting over the robot. This module's general idea is presented in the following paragraphs.

Let $\boldsymbol{q}$ denote the states of FRIDA's joints (which also happen to be our generalized coordinates), the dynamic model of the robot can then be derived from the Lagrangian energy equations [127] and be described by Equation 30, where $\boldsymbol{F_v}$ is a diagonal matrix of the viscous friction coefficients, $\boldsymbol{f_S}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is a function that models the static friction at all the joints, $\boldsymbol{\tau}$ is the actuation torques acting on each of the joints and $\boldsymbol{g(q)}$ represents the gravitational forces acting on the manipulator along the generalized coordinates q**.** The definitions for matrices B and C can be found in Equation 31. In the definition of the inertia matrix, $\boldsymbol{B(q)}$, $\boldsymbol{m_i}$ is the mass of the $\boldsymbol{i}$-th link, $\boldsymbol{n}$ is the number of links in the kinematic chain, $\boldsymbol{J_P^{I_i}}$ is the Jacobian of the prismatic joints and $\boldsymbol{J_O^{(I_i)}}$is the Jacobian for the rotational joints, $\boldsymbol{I_i^i}$ is the Inertia tensor of the segment as seen from a frame fixed to itself, $\boldsymbol{R_i}$ is the rotation matrix such that $\boldsymbol{\omega_i^i} = \boldsymbol{R_i^T}\boldsymbol{\omega_i}$, where $\boldsymbol{\omega_i^i}$ is the angular velocity of the link as seen from a frame solidary to its motion and $\boldsymbol{\omega_i}$ is the angular velocity of the link expressed in the global coordinate frame. Meanwhile, in the definition of matrix **C**$(\boldsymbol{q}, \dot{\boldsymbol{q}})$, ***b***

Equation 30 - Dynamic Model of a Generic Manipulator

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + F_v\dot{q} + f_s(q,\dot{q}) + g(q) = \tau_{tot} = \tau_{ext} + \tau_{joints}$$

Equation 31- Definitions of B(q) and $C(q,\dot{q})$

$$\begin{cases} B(q) = \sum_{i=1}^{n} \left(m_i J_p^{(I_i)T} J_P^{I_i} + J_O^{(I_i)T} R_i I_i^i R_i^T J_O^{(I_i)} \right) \\ C \mid c_{ij} = \sum_{k=1}^{n} \frac{1}{2} \left(\frac{\partial b_{ij}}{\partial q_k} + \frac{\partial b_{ik}}{\partial q_j} - \frac{\partial b_{jk}}{\partial q_i} \right) \dot{q_k} \end{cases}$$

Therefore, if one knows the matrices, the joint torques, the joint positions and joint velocities one may then apply the method proposed in [128], which detects collisions based on the robot's momentum. Let $\boldsymbol{p} = \boldsymbol{B}(\boldsymbol{q})\dot{\boldsymbol{q}}$ denote the robot's momentum. One can then calculate a residual **r** through Equation 32, setting ***r(0) = 0*** and where ***K_I > 0*** is a diagonal matrix. The decoupled dynamics of ***r*** can then be calculated by Equation 33, wherein, under perfect conditions (*i.e.* no noise or uncertainties: ***K_I***→ ∞), we have that ***r*** ≈ $\boldsymbol{\tau_{ext}}$, obtaining an estimate of a vector of external torques.

Equation 32 - Residual (external) estimation

$$r(t) = K_I \left[p(t) - \int_0^t \left(\tau_{joints} + C^T(q,\dot{q})\dot{q} - g(q) - F_v\dot{q} - f_s(q,\dot{q}) + r \right) - p(0) \right]$$

Equation 33 Residual's decoupled dynamics

$$\dot{r} = -K_I r + K_I \tau_{ext}$$

Ideally, a detailed dynamical model of the end-effector should also be provided in order for the external torque estimation to be precise. However, since this observer was being used purely to detect when contact had been made with the object (and since the end-effector approaches the object very slowly, the inertial torques are negligible) a generic model was used to describe the end effector, and a high threshold was set to indicate when contact was made. This module would then continuously evaluate the external forces and, once they had reached the specified threshold that was indicative of contact being made between the suction cup and the object, it would set a flag within the robot’s main program that would signal it to open the vacuum valves.

This estimation was performed not by FRIDA, but by the Rosetta Computers. However, since these implementation details are too specific to this project due to FRIDA being a prototype, no further details are provided on this infrastructure.

4. Experimental Results

4.1. Description of Experiments

In order to evaluate the feasibility and effectiveness of this HMI – and the associated grasping generation algorithm – a series of tests were performed. In these tests, different objects were selected from the lab environment and placed within the region delineated with tape in Figure 37. The reason why the objects had to be placed exclusively in that region is because it represents the approximate intersection between the Kinect's depth camera field of view and FRIDA's workspace. Should the targets be placed too close to the robot, the KINECT would not be able to see it and should they be placed too far from it, they would be correctly identified by the HMI, but completely out of reach.

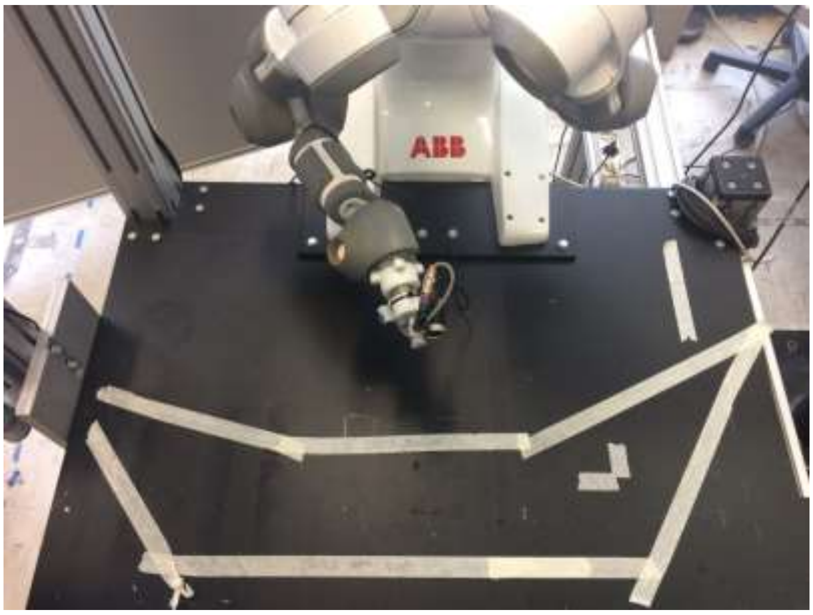


Figure 37 - Top view of Frida's workspace

Each of the tests was designed to test the robustness of the proposed algorithms to the characteristics of the manipulated elements. The first one consisted in grasping a simple rather shiny black mouse placed in a matte white background, displayed in Figure 38, which served as a baseline for the basic functionality of the code, as the scene was easily segmented and the colors were reflective – since reflectiveness is one source of noise in depth cameras.

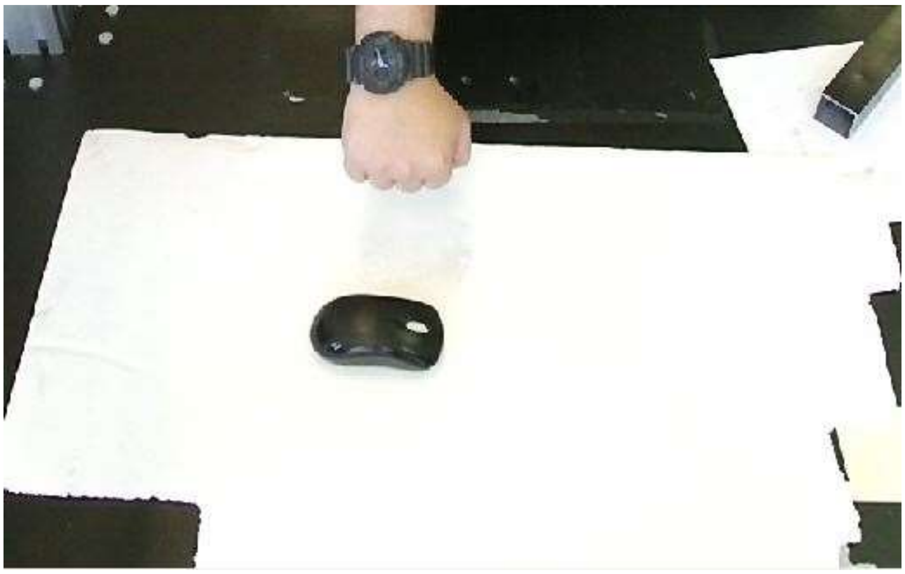


Figure 38 - First test: Matte black mouse on white matte surface

In the second experiment, four nearly identical matte white mice were positioned in a line in front of the robot with a black slightly reflective table surface as background. The robot was then ordered to pick each one of them. This setup was designed to test two aspects of the algorithm: The robustness of the algorithm with respect to objects with the same color being placed close together and the negative impact of a black reflective surface in the background. In this example, both the algorithm proposed in [76] and the HMI proposed by [21] would fail, as they would not be able to isolate one single mouse from the scene in order to pick it up. Additionally, the curved surface of the mice makes them rather challenging to grasp with suction cups, as it makes it harder for the vacuum seal to form.

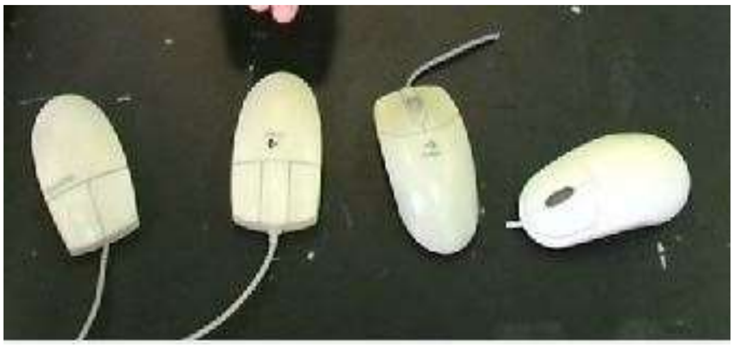


Figure 39- Four nearly identical mice in black table surface

The third and final scenario involved placing four different objects in the grasping area with different shapes, colors and textures, shown in Figure 40. Once again, each object has been selected in order to challenge the robustness of the system in different ways. First of all, each object was picked with a different color, to test whether the HMI is robust with respect to the color of the object. The mouse and the box on the right share the same color, to verify if the algorithm is robust to colors with slightly different shades. The white box was picked because it had a lid that, although provided a simple planar surface for grasping, could detach or break should the force exerted by the manipulator be too great, acting as a test for the effectiveness of the force-estimator software switch. The red box was chosen for a similar reason, with the added complexity of having a bright reflective surface, which both made color segmentation harder and jeopardized the depth sensor's performance. Finally, the banana was chosen as a final challenge for the HMI, since its color is not uniform (being instead a smooth color gradient from yellow to green), its shape is organic and uneven and its structure is rather fragile, being easily crushed – thus testing all three robustnesses at once.

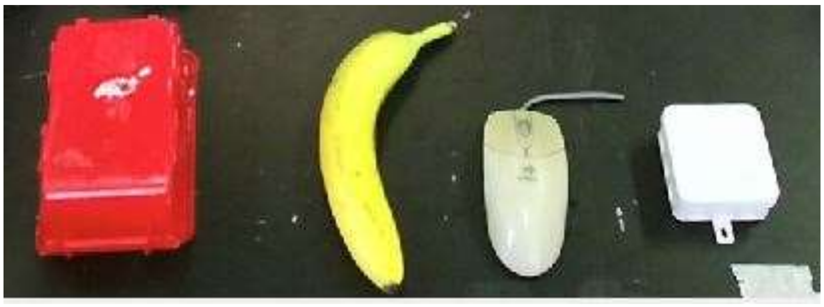


Figure 40 - Four final objects

4.2. Result Analysis

The results of the first test were satisfactory, as can be seen in Figure 41. These images are, from the top left in clockwise order: The color image transformed into the depth image space, the sectioned image, a binary highlight of the selected section and the extracted point cloud, with the grasping point and direction. It can be clearly seen that the image segmentation, though jeopardized by the transformation between color and depth spaces is still efficient in separating relevant objects in the robot’s workspace. One should also note that the resulting point cloud is considerably noisy. This stems from two reasons: One is that the mouse is rather small and, thus, we see it at a much closer scale than the previous example (the red hat, shown in Figure 31). Additionally, one should also notice that even though the resulting point cloud is quite noisy, the grasping pose that was chosen is placed at a rather central location of the mouse and at an orientation that is approximately normal to the surface of the mouse. We may conclude, therefore, that the approximation proposed in section 2.5 of using the average surface normal vector within the gripper radius adds robustness to the algorithm with respect to noisy point clouds.

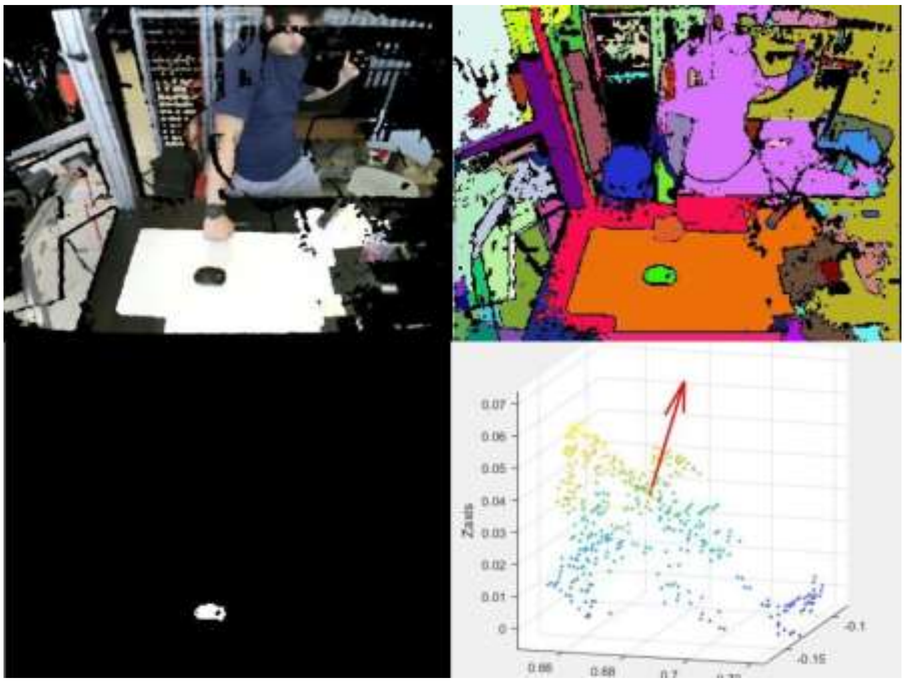


Figure 41 - Summary of the results of the first experiment

The results of the second experiment can be seen in Figure 42, Figure 43, Figure 44 and Figure 45, which have the same order of pictures as Figure 41 and depict, respectively, the choosing of the mice from left to right. From these experiments, a few conclusions can be taken. First, one should note that due to the mice being used in this experiment being matte, their extracted point clouds were considerably smoother and accurate to the shape of the mice than the ones obtained in experiment 1. Second, one must also notice that the segmentation of the table on which the mice are standing is considerably less consistent. The reason for this phenomenon is because the reflectiveness of the black surface of the table introduces considerable noise and imprecision to the measurements of the depth sensor, clearly shown in Figure 46. One must also note that the grasping poses for all four mice were similar- as it would be expected- since the mice were oriented in about the same direction and were topologically very similar. Finally, one should note that the proposed HMI was able to correctly identify and pick a grasping pose for virtually identical objects placed close to one another, demonstrating the robustness of this method with respect to object color.

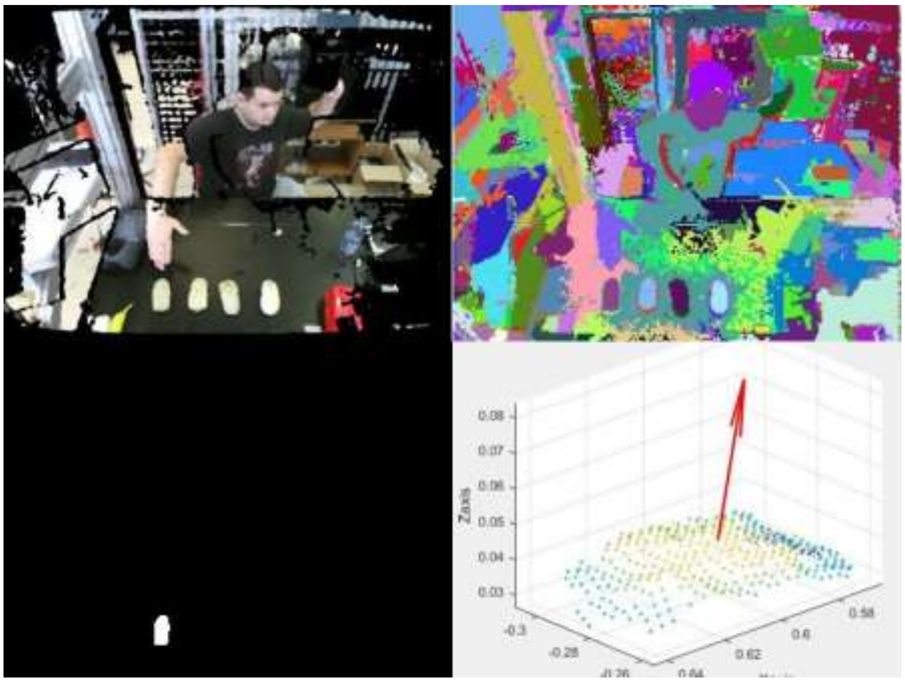


Figure 42 - Second Experiment: Leftmost mouse

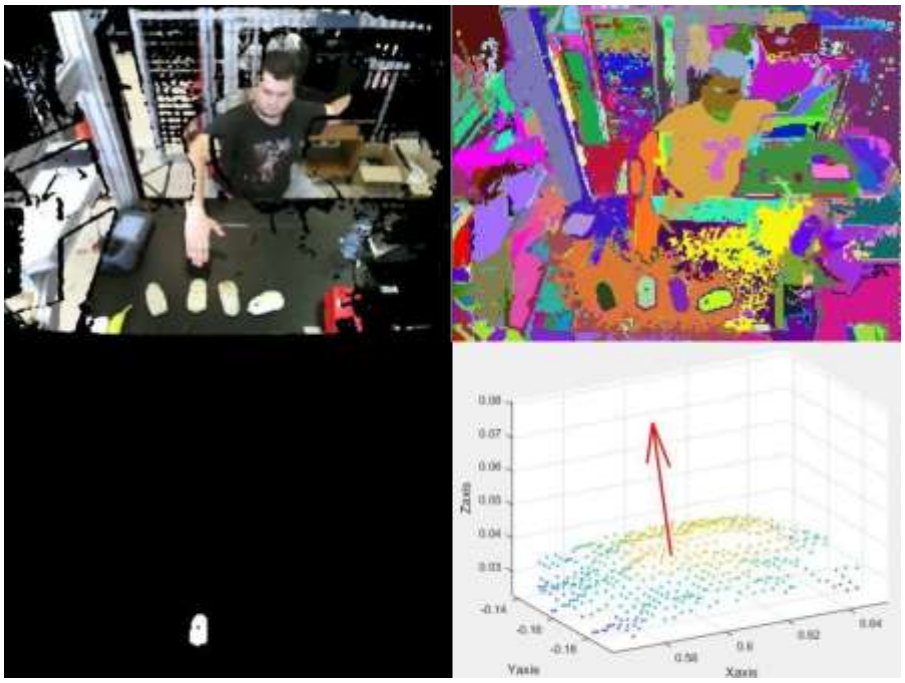


Figure 43 - Second Experiment: Middle left mouse

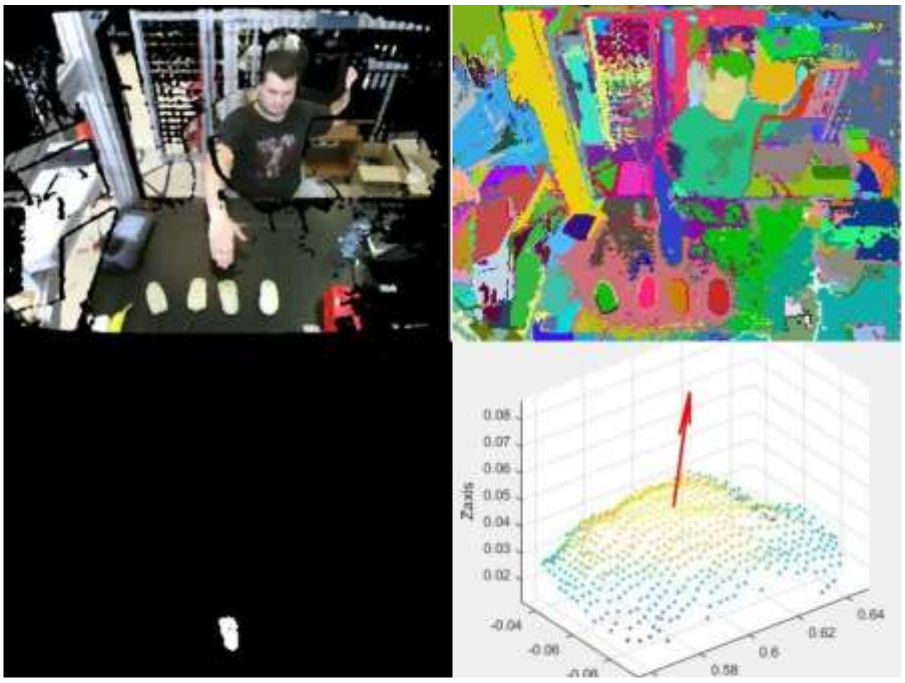


Figure 44 - Second Experiment: Middle right mouse

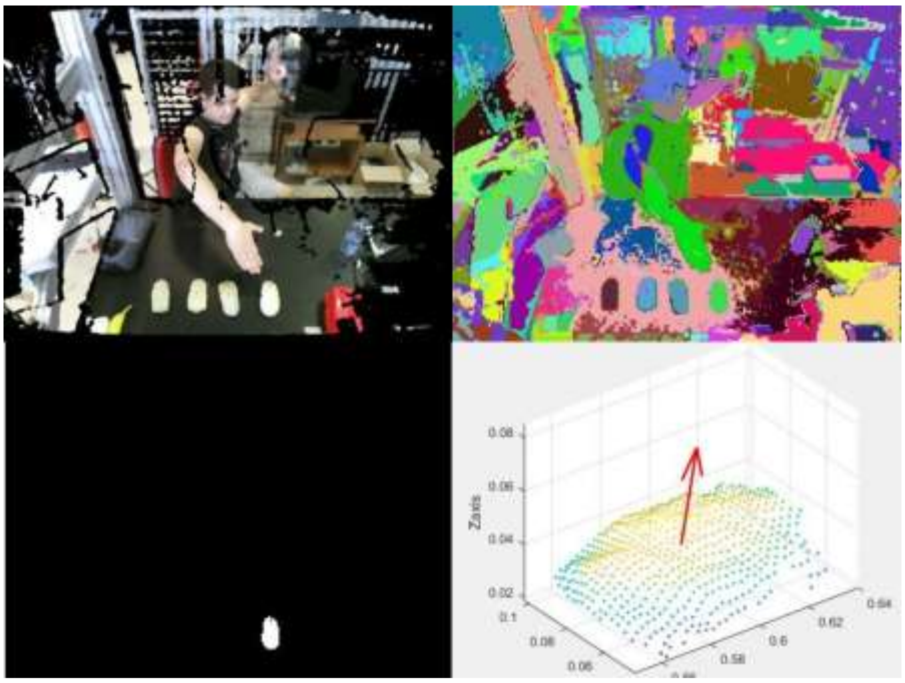


Figure 45 -Second Experiment: Rightmost mouse

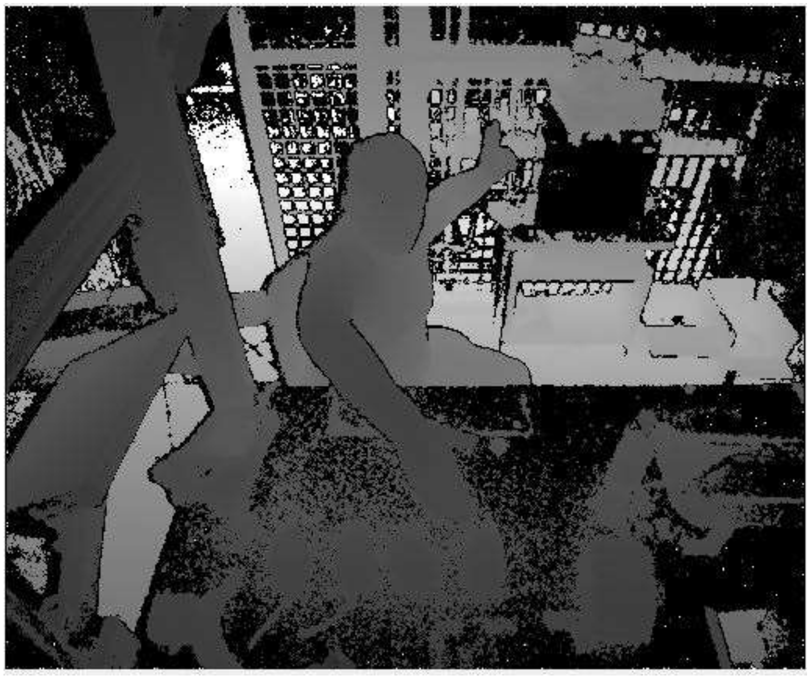


Figure 46 - Depth image from Figure 45

In the final set of evaluations, a similar procedure was followed, and the results are shown in Figure 47, Figure 48, Figure 49 and Figure 50 representing, respectively, the picking of the red box, the mouse, the white box and the banana and using the same structure of the previous photo montages. In Figure 47 one may see the fruits born by the alternative point cloud filter proposed in this work – as the white label on top of the red box had been cut out of the box in the segmentation steps, but its points were restituted to the final point cloud.

A more notable robustness result, however, can be seen in Figure 48. In this stage of the experiment, while trying to select the white mouse, two events that could lead to miss-picking an object happened simultaneously: One event was a miss-segmentation, which was probably due to the reflective surface of the mouse causing its borders to be put in a different section from the rest of the body of the mouse and a selection misinterpretation, likely caused by the imprecise skeletal estimation of the Kinect. This caused the algorithm to initially select only the borders of the mouse as the object to be grasped. However, due to the filtering step using the extended point cloud proposed in this work, the whole mouse was still identified and a grasping pose was still successfully picked. The same phenomenon can be observed for Figure 49, wherein a selection and segmentation error occurs, but the algorithm is still able to correctly identify the full object's point cloud and correctly pick a grasping pose for it.

Additionally, in Figure 50 we can see the robustness of the segmentation algorithm, which managed to correctly section the banana even though it did not have a consistent color scheme, unlike industrialized objects. It should be noted that the simple force-feedback that was implemented in the robot was enough to prevent it from crushing a delicate banana, as can be seen in Figure 51 and Figure 52, which depicts, a sequence of frames from a demonstration video where the banana is picked. In Figure 51 subplot a) indicated the moment in which the "grasp" is issued; subplot b) shows the moment the program plots the selected grasping pose on the screen and issues the commands to the robot; subplot c) shows the robot's approaching angle to the banana and subplot d) indicates the moment in which the robot makes contact with the banana and activates the suction gripper. In Figure 52 subplot e) represents the moment in which the banana is lifted from the table; subplot f) depicts the moment when the robot stops moving slightly above the bin where it will deposit the banana; subplot g) depicts moments after the releasing of the banana from the gripper and subplot h) depicts the robot returned to its standby position to wait for new instructions.

Finally, for the sake of completion, it should be noted that the average time elapsed between the HMI receiving the command with the right hand and the robot starting to move was of 6 seconds, a tolerable run-time performance for a complex HMI.

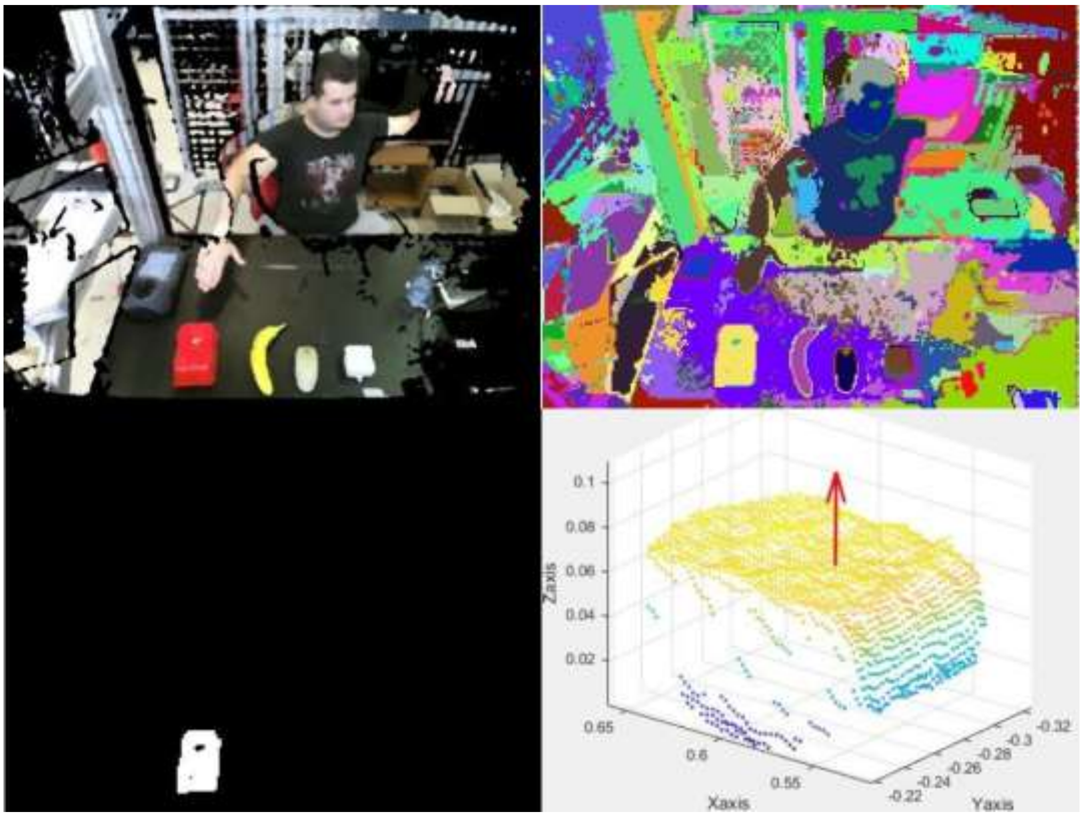


Figure 47 - Third Experiment: Red Box

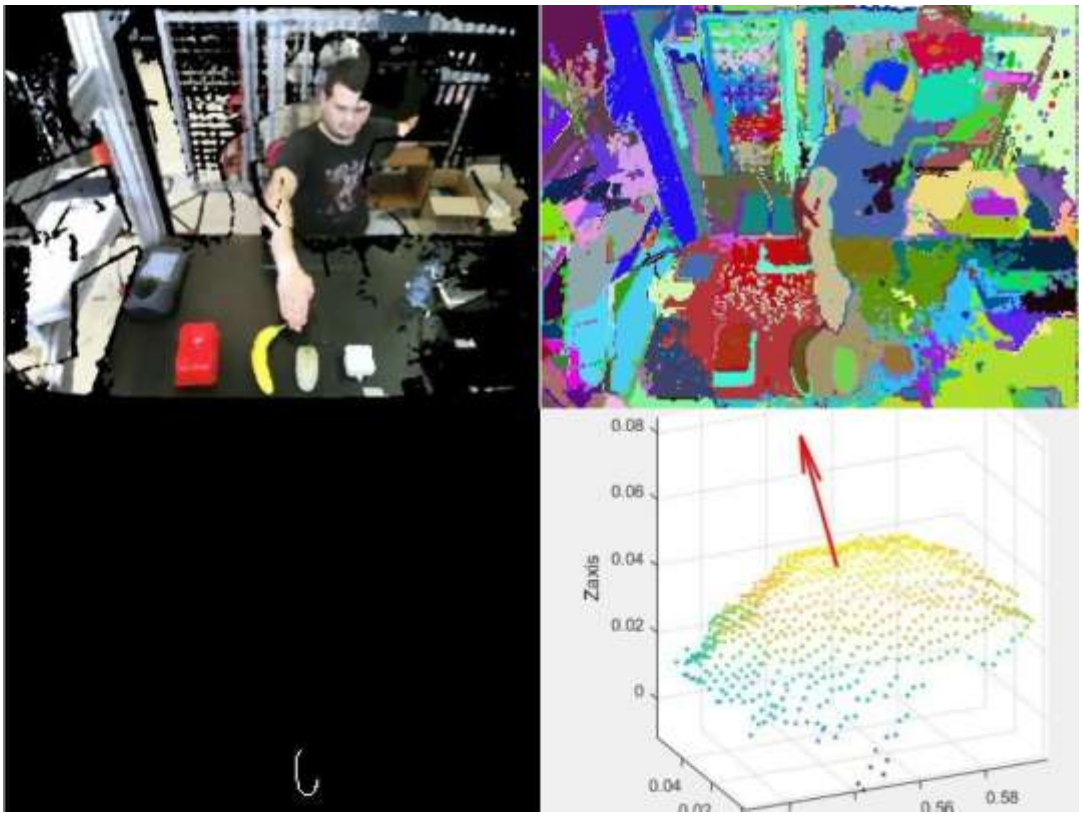


Figure 48- Third Experiment: White Mouse

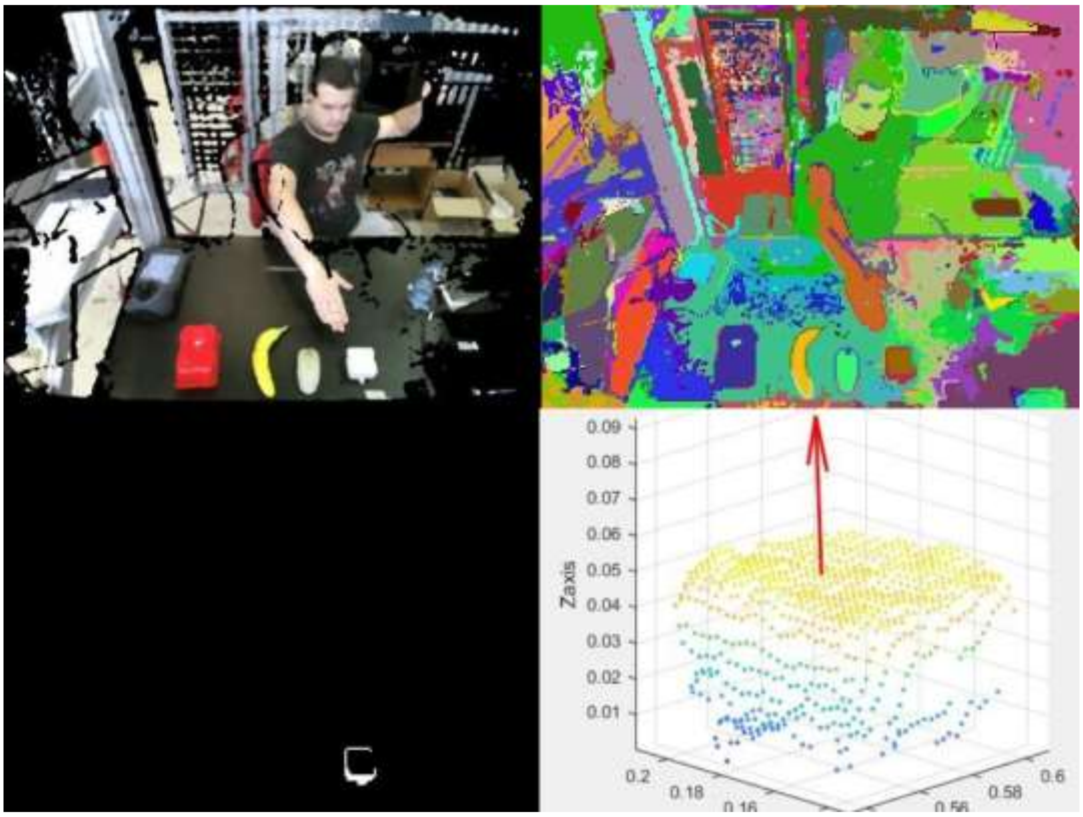


Figure 49- Third Experiment: White Box

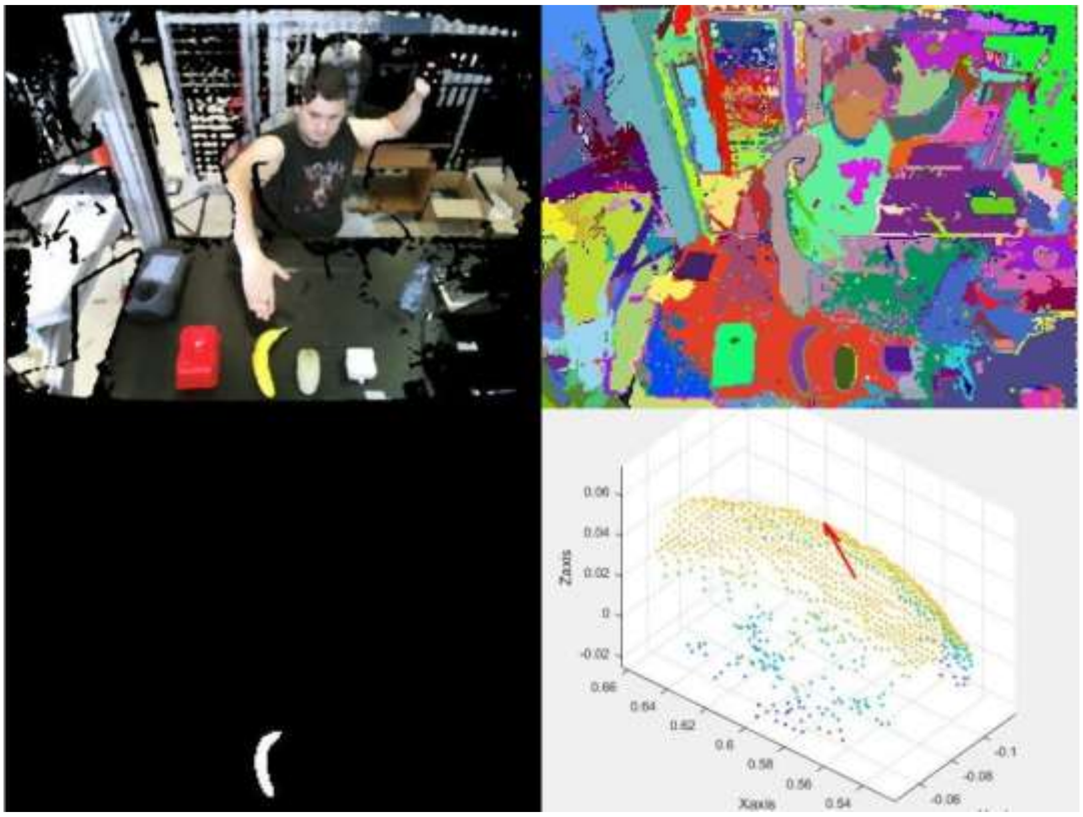


Figure 50- Third Experiment: Banana

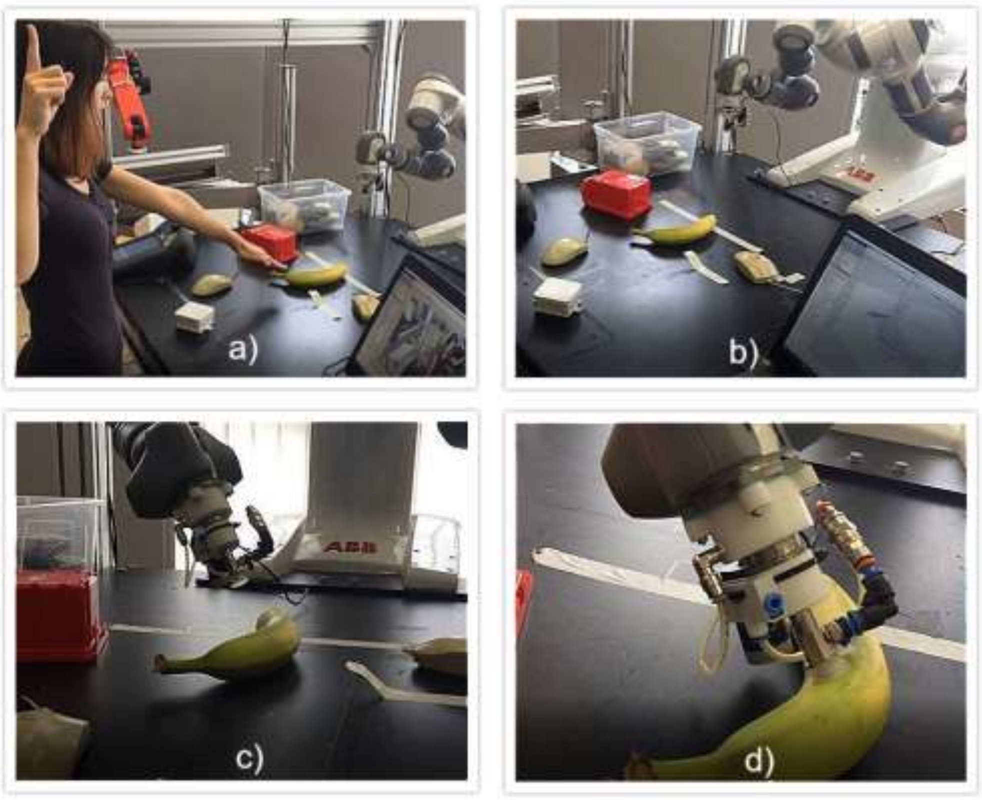


Figure 51- Banana Picking Video Screenshots

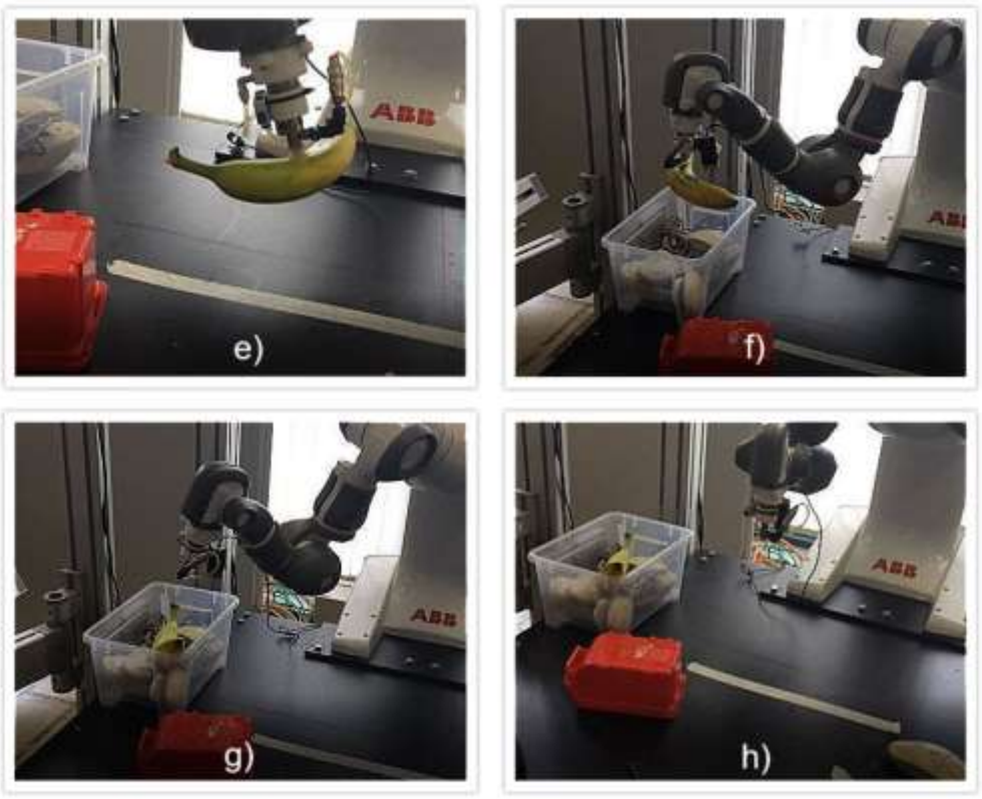


Figure 52 - Banana picking Video Screenshots (II)

4.3. Main issues

This HMI, however, is not free from issues. The greatest problem faced when using this HMI was the instability of the Kinect's skeletal tracking mode. As can be seen in Figure 53, the estimation of the joints is not very robust – and the estimation of the position of the tips of the hands is often times way off. This imprecision was already noted in literature. In [129] the authors noted that the Kinect is a good tool for tracking large scale movement, but rather imprecise in tracking fine motion and 3d positioning of joints. Whereas in [130] it is noted that the Kinect could present tolerable skeletal tracking performances, but that, in general positions when seen from different angles than the optimal observation angles (for which it was developed and trained), the Kinect skeletal estimation points could be off by over 10 cm. This effect was indeed observed as, often times, the estimation of the skeleton flickered uncertainly and at times the Kinect simply failed to track the skeleton at all.

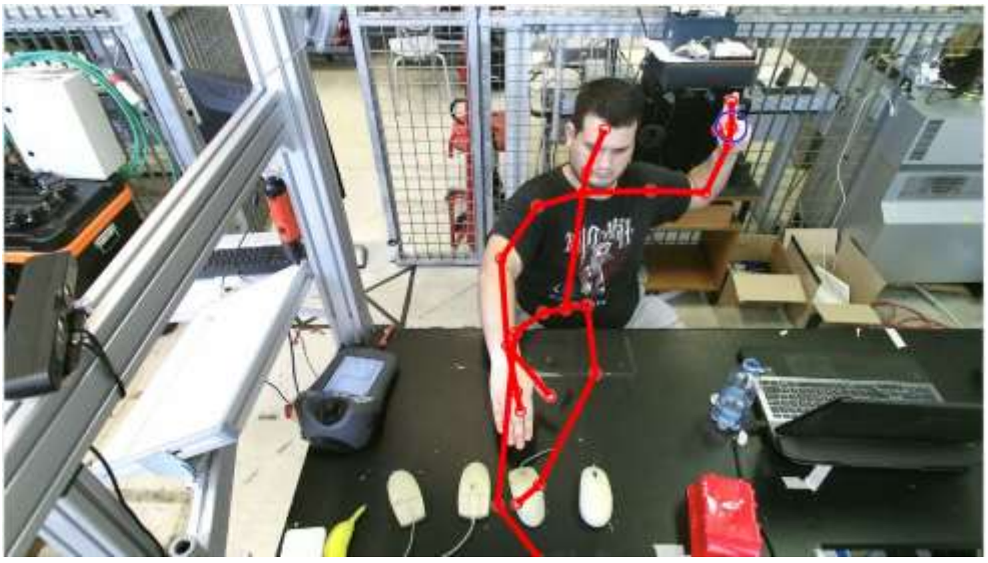


Figure 53 - Estimated Skeleton over the color image

Another issue that was also faced at times was that due to the removal of Ancona's uniformity filtering step (such as WLOP, described in [94]) in order to improve run-time performance of the HMI, sometimes the estimated object's point cloud ended up being too noisy, with the estimated surface normals being sufficiently skewed from the actual surface normals to prevent the vacuum seal from forming.

Another difficulty faced when using the suction grippers was that sometimes the picked grasping points contained crevices – like those associated with mouse buttons – which prevented the vacuum from forming at the smoothest surfaces of the object. This, in turn, made it impossible for the HMI to effectively grasp the object in the current grasp selection framework since these crevices cannot be detected by the Kinect at a long distance and cannot, thus, be accounted for during the grasp generation stage.

Two final criticisms to the HMI framework remain. One is that it is rather cumbersome to give a command with the right hand while pointing at the object you wish to be manipulated with the left hand. Often, while trying to have the state of the right hand acknowledged by the Kinect, the users ended up pointing away from the object they desired to manipulate, causing the system to either enter an error state or try and grasp something that it was not supposed to grasp. The final Issue is that even though the run time performance was still largely improved with the compilation of most of the Matlab code into MEX files, (from around 60 seconds to around 7), this execution time is still a bit sluggish for it to be acceptably applied to a domestic or industrial setting. Possible solutions to these issues are hypothesized in the next section.

5. Conclusion, Improvements and Future Work

5.1. Improvements and Future Work

As mentioned in section 4.3, many issues were observed during the execution of this HMI's proof of concept. This section will try to provide a few glimpses of possible solutions to these problems and leave them as suggestions for future development upon this work.

The first issue to be addressed here is the instability of the skeletal tracking of the Kinect under non-ideal conditions (viewing the user at an angle, for instance). There are a few ways to approach this problem. One of them would be to add more Kinect sensors, one responsible for tracking the user, being placed at an optimal angle and distance from him, and one responsible for tracking the robot's field of view and then later merging their point clouds. This approach could provide the HMI with better detailing of the scene at a comparatively low computational cost. However, adding another Kinect sensor would increase production cost of the HMI, as well as its energy consumption and, even more crucially, encumber the robot with one more element. Software solutions, thus, should be preferred. Two possible approaches may be suggested, both claiming to be applicable in real time. In [131] a Random Decision Forest is trained to classify body parts and the output of this classification process is then fed into a local mode finding algorithm to estimate the joint locations on a hand skeleton model. This framework, however, can be easily extended to the full skeleton. The second approach, presented in [132], proposes a piecewise linear predicting Kalman filter framework to fill in gaps in skeletal models missing some parts. This could be ideally used as a way to refine the raw skeletal estimation provided by the Kinect.

The second issue is the cumbersomeness of issuing commands with one hand while pointing with another. This issue is associated with another, more serious one: With the Kinect’s limited fine gesture recognition, the best possible command vocabulary is rather limited. One possible way of solving this problem would be to follow the same framework as the one proposed in [21], where a full speech recognition capability was added to the HMI, which vastly expands the possible array of commands that could be given to the robot. While still on the topic of extending the robot’s vocabulary of actions by introducing voice-based interaction to the HMI, a few modules could greatly benefit from this. By adding a memory (or database) to the system, one could allow the user to store the pictures, point clouds and optimum grasping poses for objects that were already manipulated by the robot in previous moments, associating them with a name. Such expansion could later allow users to merely order the robot to pick the object by name, eliminating the need to point at it to identify it. This model matching could be performed using the Nearest Neighbor algorithm or any of the other algorithms discussed in section 2.3.2.

The third issue that is addressed here is that of the crevices that keep the vacuum seal from being formed. Though no analytic solution was found to this problem, a simple heuristic is proposed here in order to solve it. In order to avoid this issue, one could establish an ***n***-try protocol for attempting to grasp a given object. In this protocol, the robot is allowed to try grasping an object ***n*** times. At each time it tries grabbing the object, should it fail, it should try a different position, calculated from the point cloud model. In order to avoid the same positions being selected over and over again, after each failed attempt, the algorithm should mark the previous grasping points – as well as a small area around them – as forbidden points – and compute the next viable candidate points from the new list of points.

The final issue addressed here is the run-time performance. This issue could be solved by using more efficient data structures and writing the code directly in a faster compiled language, such as C++ or C#, since even though compiling MEX functions helps with performance, it is still reportedly 30% slower than true C++ code [123]

As already mentioned in the introductory paragraphs, the focus of this work was developing and proposing a new HMI framework, with the grasping components being simplified in order to make a demonstration viable given the strict time constraints of this project. Thus, one final suggestion for improvement over this project would be to couple this HMI to additional robot control modules, such as obstacle avoidance, motion planning and proper force estimation modules in order to take the project one step closer to being applied in a real setting, outside of academia.

5.2. Conclusion

With the ever growing pervasion of robots in both industrial and domestic settings, automata are bound to be commanded by increasingly less technologically literate people in increasingly more chaotic environments. In the realm of human-robot interaction, grasping objects is one of the fundamental uses of assistive robots, both in the industry and at home. However, after performing a broad review of the current HMI technology, it was verified that there existed a significant gap in technology in the field that could be used for commanding a grasping task for completely unknown objects in a completely unknown environment. Therefore, in an effort to lower the barrier of entry to commanding automata in grasping tasks, this work aimed to create a simple and intuitive way through which a user could issue commands, by following the *point-and-command* paradigm.

Though not free of issues, the proposed framework managed to achieve considerable stability and consistency of performance with respect to the several object characteristics that would pose a problem to existing technologies by integrating a smarter 3D image segmentation algorithm with simple, yet effective, heuristic for grasping objects with suction-based grippers.

In conclusion, the author hopes that this work helps establish a little better the field of multimodal HRIs, lowering the barrier of entry to programming robots and allowing for a brighter, more efficient future for all humans, in better synchrony with the automata around them.

Bibliography

[1] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, “Industry 4.0,” *Bus. Inf. Syst. Eng.*, vol. 6, no. 4, pp. 239–242, Aug. 2014.

[2] P. A. Lasota, G. F. Rossano, and J. A. Shah, “Toward safe close-proximity human-robot interaction with standard industrial robots,” in *2014 IEEE International Conference on Automation Science and Engineering (CASE)*, 2014, pp. 339–344.

[3] C. Papadopoulos, I. Mariolis, A. Topalidou-Kyniazopoulou, G. Piperagkas, D. Ioannidis, and D. Tzovaras, “An Advanced Human-Robot Interaction Interface for Teaching Collaborative Robots New Assembly Tasks,” 2017, pp. 180–190.

[4] J. Allen and K. E. MacLean, “Personal Space Invaders,” in *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction Extended Abstracts - HRI’15 Extended Abstracts*, 2015, pp. 185–186.

[5] T. S. Tadele, T. de Vries, and S. Stramigioli, “The Safety of Domestic Robotics: A Survey of Various Safety-Related Publications,” *IEEE Robot. Autom. Mag.*, vol. 21, no. 3, pp. 134–142, Sep. 2014.

[6] M. Prats, P. J. Sanz, A. P. del Pobil, E. Martínez, and R. Marín, “Towards multipurpose autonomous manipulation with the UJI service robot,” *Robotica*, vol. 25, no. 02, Mar. 2007.

[7] L. Fortunati, A. Esposito, and G. Lugano, “Introduction to the Special Issue ‘Beyond Industrial Robotics: Social Robots Entering Public and Domestic Spheres,’” *Inf. Soc.*, vol. 31, no. 3, pp. 229–236, May 2015.

[8] D. R. Myers, M. J. Pritchard, and M. D. J. Brown, “Automated programming of an industrial robot through teach-by showing,” in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, 2001, vol. 4, pp. 4078–4083.

[9] G. Biggs and B. Macdonald, “A Survey of Robot Programming Systems,” *Proc. Australas. Conf. Robot. Autom. CSIRO*, vol. 1, p. 27, 2003.

[10] M. De Graaf, S. Ben Allouch, J. Van Dijk, M. M. A. De Graaf, S. Ben Allouch, and J. A. G. M. Van Dijk, “Long-Term Acceptance of Social Robots in Domestic Environments : Insights from a User’s Perspective,” *AAAI 2016 Spring Symp. “Enabling Comput. Res. Soc. Intell. human–robot Interact. a community-driven Modul. Res. Platf.*, no. 2004, pp. 96–103, 2016.

[11] D. Dereshev and D. Kirk, “Form, Function and Etiquette–Potential Users’ Perspectives on Social Domestic Robots,” *Multimodal Technol. Interact.*, vol. 1, no. 2, p. 12, Jun. 2017.

[12] S. Alexandrova, Z. Tatlock, and M. Cakmak, “RoboFlow: A flow-based visual programming language for mobile manipulation tasks,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 5537–5544.

[13] Yilu Zhang and Juyang Weng, “Action chaining by a developmental robot with a value system,” in *Proceedings 2nd International Conference on Development and Learning. ICDL 2002*, 2002, pp. 53–60.

[14] K. Mülling, J. Kober, O. Kroemer, and J. Peters, “Learning to select and generalize striking movements in robot table tennis,” *Int. J. Rob. Res.*, vol. 32, no. 3, pp. 263–279, Mar. 2013.

[15] F. Kirchner, “Q-learning of complex behaviours on a six-legged walking machine,” in *Proceedings Second EUROMICRO Workshop on Advanced Mobile Robots*, 1997, pp. 51–58.

[16] J. Kober, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” *Int. J. Rob. Res.*, vol. 32, no. 11, pp. 1238–1274, Sep. 2013.

[17] J. Peters and S. Schaal, “Reinforcement learning of motor skills with policy gradients,” *Neural Networks*, vol. 21, pp. 682–697, 2008.

[18] “ABB Teach Pendant 3HAC028357-001 DSQC679.” [Online]. Available: http://www.servo1.com/product/abb-teach-pendant-3hac028357-001-dsqc679/. [Accessed: 10-Sep-2017].

[19] J. Kober, B. Mohler, and J. Peters, “Learning Perceptual Coupling for Motor Primitives,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2008.

[20] K. Gräve, J. Stückler, and S. Behnke, “Learning Motion Skills from Expert Demonstrations and Own Experience using Gaussian Process Regression,” in *Joint International Symposium on Robotics*, 2010.

[21] R. G. Boboc, A. I. Dumitru, and C. Antonya, “Point-and-Command Paradigm for Interaction with Assistive Robots,” *Int. J. Adv. Robot. Syst.*, vol. 12, no. 6, p. 75, Jun. 2015.

[22] I. Rodomagoulakis, N. Kardaris, V. Pitsikalis, E. Mavroudi, A. Katsamanis, A. Tsiami, and P. Maragos, “Multimodal human action recognition in assistive human-robot interaction,” in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 2702–2706.

[23] M. A. Goodrich and A. C. Schultz, “Human-Robot Interaction: A Survey,” *Found. Trends® Human-Computer Interact.*, vol. 1, no. 3, pp. 203–275, 2007.

[24] J. Beer, A. Fisk, and W. Rogers, “Toward a Framework for Levels of Robot Autonomy in Human-Robot Interaction,” *J. Human-Robot Interact.*, vol. 3, Jan. 2014.

[25] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand, “An Architecture for Autonomy,” *Int. J. Rob. Res.*, vol. 17, no. 4, pp. 315–337, Apr. 1998.

[26] G. A. Bekey, *Autonomous Robots: From Biological Inspiration to Implementation and Control*. Cambridge, Massachusetts: The MIT Press, 2005.

[27] S. Franklin and A. Graesser, “Is It an agent, or just a program?: A taxonomy for autonomous agents,” Springer, Berlin, Heidelberg, 1997, pp. 21–35.

[28] H.-M. Huang, “Autonomy Levels for Unmanned Systems (ALFUS) Framework Volume I: Terminology Version,” *NIST Spec. Publ.*, 2004.

[29] R. Murphy, *Introduction to AI Robotics*. MIT Press, 2000.

[30] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards, *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, New Jersey: Alan Apt, 2003.

[31] S. Thrun, “Toward a Framework for Human-Robot Interaction,” *Human–Computer Interact.*, vol. 19, no. 1–2, pp. 9–24, 2004.

[32] M. Wooldridge and N. R. Jennings, “Intelligent agents: theory and practice,” *Knowl. Eng. Rev.*, vol. 10, no. 02, p. 115, Jun. 1995.

[33] A. Jaimes and N. Sebe, “Multimodal human–computer interaction: A survey,” *Comput. Vis. Image Underst.*, vol. 108, no. 1–2, pp. 116–134, Oct. 2007.

[34] D. Perzanowski, A. C. Schultz, W. Adams, E. Marsh, and M. Bugajska, “Building a multimodal human-robot interface,” *IEEE Intell. Syst.*, vol. 16, no. 1, pp. 16–21, Jan. 2001.

[35] A. Csapo, E. Gilmartin, J. Grizou, J. Han, R. Meena, D. Anastasiou, K. Jokinen, and G. Wilcock, “Multimodal conversational interaction with a humanoid robot,” in *2012 IEEE 3rd International Conference on Cognitive Infocommunications (CogInfoCom)*, 2012, pp. 667–672.

[36] F. Quek, D. McNeill, R. Bryll, S. Duncan, X.-F. Ma, C. Kirbas, K. E. McCullough, and R. Ansari, “Multimodal human discourse: gesture and speech,” *ACM Trans. Comput. Interact.*, vol. 9, no. 3, pp. 171–193, Sep. 2002.

[37] Frank Röck, and Nicolae Barsan, and U. Weimar*, “Electronic Nose: Current Status and Future Trends,” 2008.

[38] “Object recognition: A new application for smelling robots,” *Rob. Auton. Syst.*, vol. 52, no. 4, pp. 272–289, Sep. 2005.

[39] S. Martin and N. Hillier, “Characterisation of the Novint Falcon Haptic Device for Application as a Robot Manipulator,” in *Australasian Conference on Robotics and Automation*, 2009.

[40] D. Pepley, M. Yovanoff, K. Mirkin, D. Han, S. Miller, and J. Moore, “Design of a Virtual Reality Haptic Robotic Central Venous Catheterization Training Simulator,” in *Volume 5A: 40th Mechanisms and Robotics Conference*, 2016, p. V05AT07A033.

[41] P. Kormushev, S. Calinon, and D. G. Caldwell, “Imitation Learning of Positional and Force Skills Demonstrated via Kinesthetic Teaching and Haptic Input,” *Adv. Robot.*, vol. 25, no. 5, pp. 581–603, Jan. 2011.

[42] B. D. Argall and A. G. Billard, “A survey of Tactile Human–Robot Interactions,” *Rob. Auton. Syst.*, vol. 58, no. 10, pp. 1159–1176, 2010.

[43] C. D. Wickens, “Multiple resources and performance prediction,” *Theor. Issues Ergon. Sci.*, vol. 3, no. 2, pp. 159–177, Jan. 2002.

[44] A. Yilmaz, O. Javed, and M. Shah, “Object tracking : A survey,” *ACM Comput. Surv.*, vol. 38, no. 4, p. 13–es, Dec. 2006.

[45] Y. Li, S. Wang, Q. Tian, and X. Ding, “A survey of recent advances in visual feature detection,” *Neurocomputing*, vol. 149, no. Part B, pp. 736–751, 2015.

[46] D. G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” *Int. J. Comput. Vis.*, vol. 60, pp. 91–110, 2004.

[47] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, “Speeded-Up Robust Features (SURF),” *Comput. Vis. Image Underst.*, vol. 110, no. 3, pp. 346–359, 2008.

[48] P. F. Alcantarilla, A. Bartoli, and A. J. Davison, “KAZE Features,” in *Computer Vision -- ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part VI*, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 214–227.

[49] M. Donoser and H. Bischof, “Efficient Maximally Stable Extremal Region (MSER) Tracking,” in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1 (CVPR’06)*, 2006, vol. 1, pp. 553–560.

[50] R. Kimmel, Cuiping Zhang, A. M. Bronstein, and M. M. Bronstein, “Are MSER Features Really Interesting?,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 11, pp. 2316–2320, Nov. 2011.

[51] T. Tuytelaars and L. Van Gool, “Matching Widely Separated Views Based on Affine Invariant Regions,” *Int. J. Comput. Vis.*, vol. 59, no. 1, pp. 61–85, Aug. 2004.

[52] B. Heisele and W. Ritter, “Obstacle detection based on color blob flow,” in *Proceedings of the Intelligent Vehicles ’95. Symposium*, 1995, pp. 282–286.

[53] S. Kuhn and D. Henrich, “Fast vision-based minimum distance determination between known and unkown objects,” in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 2186–2191.

[54] S. Y. Elhabian, K. M. El-Sayed, and S. H. Ahmed, “Moving Object Detection in

Spatial Domain using Background Removal Techniques - State-of-Art," *Recent Patents Comput. Sci.*, vol. 1, no. 1, pp. 32–54, 2008.

[55] A. M. McIvor, "Background subtraction techniques," *Proc. Image Vis. Comput.*, vol. 4, pp. 3099–3104, 2000.

[56] A. R. Surabhi, S. T. Parekh, K. Manikantan, and S. Ramachandran, "Background removal using k-means clustering as a preprocessing technique for DWT based Face Recognition," in *2012 International Conference on Communication, Information & Computing Technology (ICCICT)*, 2012, pp. 1–6.

[57] K. K. Biswas and S. K. Basu, "Gesture recognition using Microsoft Kinect®," in *The 5th International Conference on Automation, Robotics and Applications*, 2011, pp. 100–103.

[58] Z. Zhang, "Microsoft Kinect Sensor and Its Effect," *IEEE Multimed.*, vol. 19, no. 2, pp. 4–10, Feb. 2012.

[59] M. Hasanuzzaman, V. Ampornaramveth, Tao Zhang, M. A. Bhuiyan, Y. Shirai, and H. Ueno, "Real-time Vision-based Gesture Recognition for Human Robot Interaction," in *2004 IEEE International Conference on Robotics and Biomimetics*, 2004, pp. 413–418.

[60] M. Budde, M. Berning, C. Baumgärtner, F. Kinn, T. Kopf, S. Ochs, F. Reiche, T. Riedel, and M. Beigl, "Point & Control -- Interaction in Smart Environments: You Only Click Twice," in *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*, 2013, pp. 303–306.

[61] Z. A. Mundher and J. Zhong, "A real-time fall detection system in elderly care using mobile robot and kinect sensor," *Int. J. Mater. Mech. Manuf.*, vol. 2, no. 2, pp. 133–138, 2014.

[62] S. Melax, L. Keselman, and S. Orsten, "Dynamics Based 3D Skeletal Hand Tracking," in *Proceedings of Graphics Interface 2013*, 2013, pp. 63–70.

[63] Q. Ji and X. Yang, "Real-Time Eye, Gaze, and Face Pose Tracking for Monitoring Driver Vigilance," *Real-Time Imaging*, vol. 8, no. 5, pp. 357–377, 2002.

[64] C. H.Morimoto and M. R.M.Mimica, "Eye gaze tracking techniques for interactive applications," *Comput. Vis. Image Underst.*, vol. 98, no. 1, pp. 4–24, Apr. 2005.

[65] A. Al-Rahayfeh and M. Faezipour, "Eye Tracking and Head Movement Detection: A State-of-Art Survey," *IEEE J. Transl. Eng. Heal. Med.*, vol. 1, pp. 2100212–2100212, 2013.

[66] T. Ohno and N. Mukawa, "A free-head, simple calibration, gaze tracking system that enables gaze-based interaction," in *Proceedings of the Eye tracking research & applications symposium on Eye tracking research & applications - ETRA'2004*, 2004, pp. 115–122.

[67] T. Ohno, "One-point calibration gaze tracking method," in *Proceedings of the 2006 symposium on Eye tracking research & applications - ETRA '06*, 2006, p. 34.

[68] Kyung-Nam Kim and R. S. Ramakrishna, "Vision-based eye-gaze tracking for human computer interface," in *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028)*, 1999, vol. 2, pp. 324–329.

[69] K. Krafka, A. Khosla, P. Kellnhofer, H. Kannan, S. Bhandarkar, W. Matusik, and A. Torralba, "Eye Tracking for Everyone," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[70] T. E. Hutchinson, K. P. White, W. N. Martin, K. C. Reichert, and L. A. Frey, "Human-computer interaction using eye-gaze input," *IEEE Trans. Syst. Man. Cybern.*, vol. 19, no. 6, pp. 1527–1534, 1989.

[71] D. North, "New Nintendo 3DS XL impressions: Face-tracking fixes the handheld's biggest problem | GamesBeat," 2015. [Online]. Available: https://venturebeat.com/2015/01/15/new-nintendo-3ds-xl-impressions-face-tracking-fixes-the-handhelds-biggest-problem/. [Accessed: 12-Nov-2017].

[72] F. Seide, G. Li, and D. Yu, "Conversational speech transcription using context-dependent deep neural networks," in *Twelfth Annual Conference of the International Speech Communication Association*, 2011.

[73] J. Norberto Pires, "Robot- by- voice: experiments on commanding an industrial robot using the human voice," *Ind. Robot An Int. J.*, vol. 32, no. 6, pp. 505–511, Dec. 2005.

[74] T.-H. Wen, D. Vandyke, N. Mrksic, M. Gasic, L. M. Rojas-Barahona, P.-H. Su, S. Ultes, and S. Young, "A Network-based End-to-End Trainable Task-oriented Dialogue System," Apr. 2016.

[75] X. Zhou, B. Hu, Q. Chen, B. Tang, and X. Wang, "Answer Sequence Learning with Neural Networks for Answer Selection in Community Question Answering," Jun. 2015.

[76] R. Ancona, "Robotic mobile manipulation in a completely unknown environment," Politecnico di Milano, 2014.

[77] J. Mahler, M. Matl, X. Liu, A. Li, D. Gealy, and K. Goldberg, "Dex-Net 3.0: Computing Robust Robot Suction Grasp Targets in Point Clouds using a New Analytic Model and Deep Learning," Sep. 2017.

[78] A. T. Miller and P. K. Allen, "GraspIt!," *IEEE Robot. Autom. Mag.*, vol. 11, no. 4, pp. 110–122, Dec. 2004.

[79] P. Violero, I. Mazon, and M. Taix, "Automatic planning of a grasp for a 'pick and place' action," in *Proceedings., IEEE International Conference on Robotics and Automation*, pp. 870–876.

[80] C. Goldfeder, P. K. Allen, C. Lackner, and R. Pelossof, "Grasp Planning via Decomposition Trees," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 4679–4684.

[81] Z. Xue, A. Kasper, J. M. Zoellner, and R. Dillmann, "An automatic grasp planning system for service robots," in *2009 International Conference on Advanced Robotics*, 2009, pp. 1–6.

[82] D. G. Lowe, "Object recognition from local scale-invariant features," *Proc. Seventh IEEE Int. Conf. Comput. Vis.*, vol. 2, 1999.

[83] N. Curtis and J. Xiao, "Efficient and effective grasping of novel objects through learning and adapting a knowledge base," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 2252–2257.

[84] J. Mahler, F. T. Pokorny, B. Hou, M. Roderick, M. Laskey, M. Aubry, K. Kohlhoff, T. Kroger, J. Kuffner, and K. Goldberg, "Dex-Net 1.0: A cloud-based network of 3D objects for robust grasp planning using a Multi-Armed Bandit model with correlated rewards," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1957–1964.

[85] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-View Convolutional Neural Networks for 3D Shape Recognition," in *The IEEE International Conference on Computer Vision (ICCV)*, 2015.

[86] M. Laskey, J. Mahler, Z. McCarthy, F. T. Pokorny, S. Patil, J. van den Berg, D. Kragic, P. Abbeel, and K. Goldberg, "Multi-armed bandit models for 2D grasp

planning with uncertainty," in *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, 2015, pp. 572–579.

[87] M. W. Hoffman, B. Shahriari, and N. de Freitas, "Exploiting correlation and budget constraints in Bayesian multi-armed bandit optimization," Mar. 2013.

[88] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design," Dec. 2009.

[89] K. Yamazaki, M. Tomono, T. Tsubouchi, and S. Yuta, "A grasp planning for picking up an unknown object for a mobile manipulator," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, 2006, pp. 2143–2149.

[90] N. Vahrenkamp, T. Asfour, and R. Dillmann, "Simultaneous Grasp and Motion Planning: Humanoid Robot ARMAR-III," *IEEE Robot. Autom. Mag.*, vol. 19, no. 2, pp. 43–57, Jun. 2012.

[91] N. Correll, K. E. Bekris, D. Berenson, O. Brock, A. Causo, K. Hauser, K. Okada, A. Rodriguez, J. M. Romano, and P. R. Wurman, "Analysis and Observations from the First Amazon Picking Challenge," Jan. 2016.

[92] "Certified International® Tunisian Sunset 4pc Mugs Multicolored : Target." [Online]. Available: https://intl.target.com/p/certified-international-174-tunisian-sunset-4pc-mugs-multicolored/-/A-50639740. [Accessed: 19-Nov-2017].

[93] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 1–4.

[94] H. Huang, D. Li, H. Zhang, U. Ascher, D. Cohen-Or, H. Huang, D. Li, H. Zhang, U. Ascher, and D. Cohen-Or, "Consolidation of unorganized point clouds for surface reconstruction," *ACM Trans. Graph.*, vol. 28, no. 5, p. 1, Dec. 2009.

[95] S. Chitta, E. Jones, M. Ciocarlie, and K. Hsiao, "Mobile Manipulation in Unstructured Environments: Perception, Planning, and Execution," *IEEE Robot. Autom. Mag.*, vol. 19, no. 2, pp. 58–71, Jun. 2012.

[96] B. Peng, L. Zhang, and D. Zhang, "A survey of graph theoretical approaches to image segmentation," *Pattern Recognit.*, vol. 46, no. 3, pp. 1020–1038, 2013.

[97] A. N. Tropea, J. L. Valerio, M. J. Camerino, J. Hix, E. Pecor, P. G. Fuerst, and S. S. Long, "Computer Assisted Segmentation Tool: A Machine Learning Based Image Segmenting Tool for TrakEM2," in *Bioinformatics Research and Applications: 13th International Symposium, ISBRA 2017, Honolulu, HI, USA, May 29 -- June 2, 2017, Proceedings*, Z. Cai, O. Daescu, and M. Li, Eds. Cham: Springer International Publishing, 2017, pp. 246–257.

[98] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," *CoRR*, vol. abs/1606.0, 2016.

[99] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3213–3223.

[100] F. Yi and I. Moon, "Image segmentation: A survey of graph-cut methods," in *2012 International Conference on Systems and Informatics (ICSAI2012)*, 2012, pp. 1936–1941.

[101] M. H. J. Vala and A. Baxi, "A review on Otsu image segmentation algorithm," *Int. J. Adv. Res. Comput. Eng. Technol.*, vol. 2, no. 2, p. pp–387, 2013.

[102] D. Liu and J. Yu, "Otsu Method and K-means," in *2009 Ninth International Conference on Hybrid Intelligent Systems*, 2009, pp. 344–349.

[103] J. Canny, “A computational approach to edge detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, no. 6, pp. 679–698, 1986.

[104] S. A. Raut, M. Raghuwanshi, R. . Dharaskar, and A. Raut, “Image Segmentation – A State-Of-Art Survey for Prediction,” in *2009 International Conference on Advanced Computer Control*, 2009, pp. 420–424.

[105] L. Luccheseyz and S. K. Mitray, “Color image segmentation: A state-of-the-art survey,” *Proc. Indian Natl. Sci. Acad.*, vol. 67, no. 2, pp. 207–221, 2001.

[106] L. Shafarenko, M. Petrou, and J. Kittler, “Automatic watershed segmentation of randomly textured color images,” *IEEE Trans. Image Process.*, vol. 6, no. 11, pp. 1530–1544, 1997.

[107] D. Martin, C. Fowlkes, D. Tal, and J. Malik, “A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics,” in *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, vol. 2, pp. 416–423.

[108] P. F. Felzenszwalb and D. P. Huttenlocher, “Efficient Graph-Based Image Segmentation,” *Int. J. Comput. Vis.*, vol. 59, no. 2, pp. 167–181, Sep. 2004.

[109] Jianbo Shi and J. Malik, “Normalized cuts and image segmentation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 888–905, 2000.

[110] Song Wang and J. M. Siskind, “Image segmentation with ratio cut,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 6, pp. 675–690–4, Jun. 2003.

[111] D. Comaniciu and P. Meer, “Mean shift: a robust approach toward feature space analysis,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 5, pp. 603–619, May 2002.

[112] D. Rao, Q. V Le, T. Phoka, M. Quigley, A. Sudsang, and A. Y. Ng, “Grasping novel objects with depth segmentation,” in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 2578–2585.

[113] P. K. Nathan Silberman Derek Hoiem and R. Fergus, “Indoor Segmentation and Support Inference from RGBD Images,” in *ECCV*, 2012.

[114] R. Andreas, “OSD – Institut für Automatisierungs- und Regelungstechnik.” [Online]. Available: https://www.acin.tuwien.ac.at/vision-for-robotics/software-tools/osd/. [Accessed: 21-Nov-2017].

[115] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, “Algorithms for Hyper-Parameter Optimization,” in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 2546–2554.

[116] C. Hernandez, M. Bharatheesha, W. Ko, H. Gaiser, J. Tan, K. van Deurzen, M. de Vries, B. Van Mil, J. van Egmond, R. Burger, M. Morariu, J. Ju, X. Gerrmann, R. Ensing, J. van Frankenhuyzen, and M. Wisse, “Team Delft’s Robot Winner of the Amazon Picking Challenge 2016,” *CoRR*, vol. abs/1610.0, 2016.

[117] Y. Domae, H. Okuda, Y. Taguchi, K. Sumi, and T. Hirai, “Fast graspability evaluation on single depth maps for bin picking with general grippers,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 1997–2004.

[118] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, “SLIC Superpixels Compared to State-of-the-Art Superpixel Methods,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 11, pp. 2274–2282, Nov. 2012.

[119] Microsoft, “Body tracking.” [Online]. Available: https://msdn.microsoft.com/en-us/library/dn799273.aspx. [Accessed: 25-Nov-2017].

[120] Richard Hoover, “Adventures in Motion Capture: Using Kinect Data (Part 2).”

[Online]. Available: http://www.sealeftstudios.com/blog/blog20160708.php. [Accessed: 25-Nov-2017].

[121] "MATLAB Coder - MATLAB." [Online]. Available: https://www.mathworks.com/products/matlab-coder.html. [Accessed: 28-Nov-2017].

[122] D. Kragic, M. Björkman, H. I. Christensen, and J.-O. Eklundh, "Vision for robotic object manipulation in domestic settings," *Rob. Auton. Syst.*, vol. 52, no. 1, pp. 85–100, Jul. 2005.

[123] J. R. Terven and D. M. Córdova-Esparza, "Kin2. A Kinect 2 toolbox for MATLAB," *Sci. Comput. Program.*, vol. 130, pp. 97–106, Nov. 2016.

[124] J. R. Terven and D. M. Cordova, *Kin2 . User Guide*, vol. 1. 2017.

[125] "Coordinate mapping." [Online]. Available: https://msdn.microsoft.com/en-us/library/dn785530.aspx. [Accessed: 25-Nov-2017].

[126] "Create TCP/IP client object to communicate over TCP/IP - MATLAB tcpclient." [Online]. Available: https://www.mathworks.com/help/matlab/ref/tcpclient.html. [Accessed: 27-Nov-2017].

[127] P. Rocco, "Control of industrial robots Robot dynamics Control of industrial robots – Robot dynamics – Paolo Rocco."

[128] A. de Luca and R. Mattone, "Sensorless Robot Collision Detection and Hybrid Force/Motion Control," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 999–1004.

[129] B. Galna, G. Barry, D. Jackson, D. Mhiripiri, P. Olivier, and L. Rochester, "Accuracy of the Microsoft Kinect sensor for measuring movement in people with Parkinson's disease," *Gait Posture*, vol. 39, no. 4, pp. 1062–1068, 2014.

[130] Jungong Han, Ling Shao, Dong Xu, and J. Shotton, "Enhanced Computer Vision With Microsoft Kinect Sensor: A Review," *IEEE Trans. Cybern.*, vol. 43, no. 5, pp. 1318–1334, Oct. 2013.

[131] C. Keskin, F. Kıraç, Y. E. Kara, and L. Akarun, "Real Time Hand Pose Estimation Using Depth Sensors," in *Consumer Depth Cameras for Computer Vision*, London: Springer London, 2013, pp. 119–137.

[132] Q. Wu and P. Boulanger, "Real-Time Estimation of Missing Markers for Reconstruction of Human Motion," in *2011 XIII Symposium on Virtual Reality*, 2011, pp. 161–168.