A Framework for Robotic Grasping and Handover With an Underactuated Three-Fingered Hand

BY

MATTEO PALLOMO
B.S. Politecnico di Torino, Turin, Italy, 2017

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2019

Chicago, Illinois

Defense Committee:
Miloš Žefran, Chair and Advisor
Shuo Han
Alessandro Rizzo, Politecnico di Torino

To my parents,

Lucia and Valerio,

the best teachers in my life.

ACKNOWLEDGMENTS

I want to deeply thank all the people that have helped me out during this five years. With the end of this document, I'll probably close the academic part of my life and I could not be more pleased of the people I met and the experience I had.

I want to thank professor Žefran from the bottom of my heart, its guidance and encouragement helped me out during the entire research work. He has shown particular kindness in my regard and always understood my personal issues without any judgment or concern. I'm very grateful to had the occasion of working with him. His way of approaching students has astonished me, for its simplicity and humanity but at the same time ability to push them past their limits.

My utmost gratitude goes to professor Rizzo for his steadfast support and patience. Despite the distance, he has always showed support and interest in my research, making himself available in every moment I needed help or suggestions.

When I first started Politecnico I could not even imagine to go this far, I've reached goals that I did not even know existed. For this, I want to thank in particular my parents, that has always supported me, both in psychological and monetary sense; none of this could even be possible without their will to make me chase my dreams. Thank to both of you, for always lifting me up when I felt I was falling, for pushing me through all the challenges and for not letting me live in the fear of failure.

ACKNOWLEDGMENTS (Continued)

Another special thanks goes to my girlfriend Giulia, for supporting me without thinking of obstructing my path, even if this meant being thousands of kilometers apart. Thank you for enduring all my futile preoccupations, always pushing me to give my best and helping me believe in myself even through difficulties.

Thanks to Filippo, Alberto and Samuele, the former has been my roommates for four years in Torino, the latter for one year in Chicago. I know that I'm not an easy person to stand, but all of you helped me feel at home even when I was far from it.

I want also to thank all the people I have met in all these five years, in particular Enrico, Pierpaolo, Lorenzo and Zhanibek; thank to you I've been growing both professionally and as a person. You made the long Politecnico and UIC days funny and enjoyable.

Last but not least, I want to thank both Politecnico and UIC for giving me this amazing opportunity of growth. Thank to this experience I had a better understanding of what I want for my future and the people I want to stay with; this knowledge is priceless and means to me more than any Master or PhD.

MP

TABLE OF CONTENTS

CHAPTER **PAGE**

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DOF	Degree Of Freedom
PID	Proportional Integrative Derivative
IMU	Inertial Measurement Unit
FPS	Frames Per Second
RPM	Revolutions Per Minutes
LED	Light Emitting Diode
CMOS	Complementary Metal-Oxide Semiconductor
DPS	Digital Signal Processor
DAQ	Data Acquisition
ROS	Robot Operating System
OS	Operating System
GUI	Graphical User Interface
RF	Reference Frame
API	Application Programming Interface
ROI	Region Of Interest
CPI	Count Per Inch

LIST OF ABBREVIATIONS (Continued)

IK	Inverse Kinematic
IK	PCI eXtensions for Instrumentation

SUMMARY

Robotic grasping is a challenging problem that has been a central topic in research for many years. In the past this problem was tackled with analytic approaches, such as [5]. These methods typically assume that object, friction coefficients and contact locations are known, and considered the ability of the grasp to resist external wrenches (*i.e.* forces and torques along X,Y and Z) or constrain the object motion. They often rely on a more or less precise knowledge of the 3D model of the object, to pre-compute the grasp [6]. Recent developments in machine learning dramatically improved what was achievable with analytic approaches[4]. These new methodologies are mostly based on 2D (RGB) or 2.5D (RGB+Depth) [6] images, instead of needing a complete 3D model. Nevertheless, a challenging task in this context is how to generate training data. Collecting samples from human volunteers seems to be the most straightforward way but it is time consuming and can only produce grasps that are human like. Unfortunately, many robotic manipulators can not be easily mapped to the human hand[5]. This problem can be partially solved by creating synthetic custom data for a particular end-effector. While this approach has several advantages, it is strongly dependent on the hardware. A training-set created for a particular gripper may not be suitable for other hardware configurations, requiring other researchers to start from scratch.

Accordingly to grasp taxonomies, [1] [2], a stable grasp requires at least two opposite points on the object. This motivates us to investigate whether or not it is possible to use a data-

set and an algorithm for a parallel gripper to drive a more complex hand. We consider this approach interesting not only because it tries to reduce at its minimum the complexity of the task, but also because data-sets and algorithms for this simpler configuration are more common and easier to find, rather than ones for more complex grippers. Furthermore, if the creation of new a training set is needed, the labeling procedure will be much simpler since only few points per object must be highlighted. Obviously, we do not expect to obtain optimal grasps, but at least to get a good trade-off between complexity and optimality. This could be quite attractive in many applications where speed and simplicity are more important than an optimal result.

A natural complement to the grasping, when studying human robot interactions, is the handover. Once the object is picked up we want to safely transfer it to a receiver. In this regard, we propose an extension of one of our previous work [8]. While it was focused on a handover with a parallel gripper, we aimed to use a similar approach with an articulated hand. We not only want to achieve a smooth handover, but also take care of object's safety during the task.

Our main goal with this thesis is to provide our robot the abilities to grasp different objects and, eventually, hand them over. Hopefully, this will help in future studies on human-robot interactions, where these two skills can be taken as granted, so to properly focus the research on the interaction.

CHAPTER 1

INTRODUCTION

1.1 Motivation and Research Questions

This research has two main goals, both of them concur in laying down the foundation for future studies in the broad field of robot-human interactions.

Firstly we want to understand if it is possible to extend a machine learning algorithm that computes grasps for a parallel-jaw gripper to a more complex and sophisticated manipulator, such as a three fingered hand. This question arises from the engineering will to reduce at its minimum the problem of grasping an object. In fact, a parallel grip is the simplest way to grasp an object, both when considering human and robotic scenarios. Moreover, we did this also to cope with an hardware limitation: our three-fingered hand is under-actuated, so it cannot achieve very sophisticated grasps configurations.

Grasps must account for many factors such as shape, orientation, obstruction and final task. Humans have evolved to use our hands with ease, and it comes natural to us to find and perform a good and stable grasp given an object. On the other hand, accomplishing this for a robot is remarkably difficult and even if substantial body of work exists, there is still no a clear recipe for computing and performing optimal grasps. Furthermore, this problem is strictly dependent on the hardware used, both when considering analytic as well as machine learning methods. These factors make existing approaches highly specialized and non-trivial to adapt

to other manipulators. The most basic end-effector that a robot can have is a parallel-jaw gripper, that consists of two plates that open or close parallel one to the other. This kind of gripper is sufficient to grasp simple objects since, if the orientation is good enough, it can guarantee two opposite contact points that constrain object motion and compensate gravity through frictional forces. It makes sense to start from this simple gripper to try to obtain grasps for a more capable end-effector. Furthermore, a lot of work has already been done for this simpler hardware piece and this will help us in reducing the development time. Our goal is to understand what advantages and drawbacks this approach shows. We do not expect to achieve optimal or human-like grasps, but at least a good trade-off between stability of the clamp and simplicity of solution.

We tackle this problem by considering a fixed approach vector, that is perpendicular to the table's surface where objects stand. This simplification will help us in correctly align the hand with the object, when performing the grasp. We consider two kind of grasp, cylindrical (Figure 1-a) and spherical grasp (Figure 1-b). The former generalizes grasp by a parallel gripper when considering multi-fingered hands. Instead, the latter improves on the parallel gripper when grasping approximately spherical objects.

The second question we want to address is what kind of control we need to deploy to hand the object over once it has been picked up. In particular, we not only want to open the hand when the object is subject to an external forces, but also to re-grasp it if, somehow, a potential fall is detected. Our methods will avoid to use a model of the hand. This is always done

(a) (b)

Figure 1: Different grasp configurations in the open state.

to cope with the simple kinematic of our manipulator. In fact, how fingers interact with the object strongly depend on the shape of the latter. This stimulated us in finding a more general approach.

These two questions are closely related since they are part of the complete handover task, that start with the grasp and finishes when a receiver get the object. With this work we aim to provide our robot the basic skills to further study this complex interaction between human and robot.

1.2 Related Works

Grasping and handover are two central topic in research. Regarding the first one, a lot of effort has been made during the year in defining and classifying humans grasps and how

they relate with task and constraints. Difficulties arise from the very complex kinematics on the human hand, consisting of 20 DOF [1], and the dependency of the grasp to final task and constraints.

Feix et al.[1] propose the following definition for a grasp:

A grasp is every static hand posture with which an object can be held securely with one hand, irrespective of the hand orientation [1].

By the previous definition it follows that holding an object on the palm is not considered a grasp, since a rotation of the wrist will make it fall. *Cutkosky et. al.*[2] proposed a grasp taxonomy for industrial environments which suggests not only a dependency from the object's shape but also on the final task that must be accomplished [2], they also propose a series of analytical measures to describe and classify grasps. *Iberall*[3] introduces the concept of virtual fingers and opposite vectors [3] when planning a grasp, she suggests that not only fingers can concur in creating opposite surfaces to achieve a stable grasp, but also other parts of the hand such as the palm.

Despite the effort, there is no a clear recipe for classifying human grasps. Nevertheless, this has not stopped the research in trying to emulate human grasp in robotic applications. Effort has been spent trying to tackle the problem analytically, by building grasp planners that relies on an approximate 3D model of the object based on shapes primitives [11][5], such as *GraspIt!*. This tool in particular scores the possible grasps using the metric introduced by Ferrari and Canny in [12] involving a wrench space analysis of the grasp. *Ciocarlie et. al.*[13] propose an

optimization algorithm based on the eigengrasp, a technique for dimensonality reduction, based on the hypothesis, verified by *Santello et al.*[14], that most useful grasp can be found in the vicinity of a small number of discrete points [13].

Recently many deep learning approaches has been used to tackle this problem. *Rai et al.*[15] use synthetic data for training a neural network and then use Microsoft Kinect to sense and perform the grasp. *Lenz et al.*[7] use a two-stage cascade detection to compute the best grasping rectangle using 2.5D images taken from the Kinect.

Even if these approaches have produced good results they partially lack of generalization, databases or synthetic images are produced for a particular manipulator and may not generalize well to other end-effectors. In this work we try to partially address this generalization problem, trying to apply a machine learning algorithm for a parallel gripper to our three fingered hand. For this purpose we borrow part of the algorithm from [7] and use it with our three-fingered hand.

Regarding the handover, many studies have been made in trying to characterize the social process behind this simple and natural task. *Strabala et al.*[10] tried to characterize the signals that humans exchange before the handover, and how this unspoken communication helps us in understand *when, where and how* the handover will take place [10]. *Huang et al.*[16] studied humans strategies for coordination based on task demand and then tried to emulate them using a robotic arm as giver. In one of our previous works [8], we conducted a human study to characterize the handover task and then implemented a controller with a failure recovery

mechanism to avoid the object to fall. In this work, we try to emulate part of this previous work using a three fingered hand instead of the canonical parallel gripper.

1.3 Contribution of this work

Our contribution with this work is to analyze what advantages and drawbacks can be obtained using a parallel gripper planner, to perform grasps with a three-fingered, under-actuated, hand. In particular, we aim to demonstrate that this solution can achieve a good trade-off between optimality and development time. This could be quite attractive in those cases where grasp is only an intermediate step for a more complex human-robot interaction.

Secondly, we propose two algorithms to perform a failure recovery handover with the same three-fingered hand. We seek to show that is possible to achieve a smooth handover, without using any model of the hand to directly control the forces applied. This approach can be interesting for all those cases where the manipulator is under-actuated, making difficult to precisely control its behavior.

1.4 Thesis Outline

This work is organized in 6 chapters:

- **Chapter 1:** provides an introduction to the document, states the research question and gives an overview on previous works.
- **Chapter 2:** will give an overview of the software and the hardware involved in the research as well as a recap of some basic concepts.

- **Chapter 3:** will explain the grasping algorithm, the experimental setup and the obtained results.
- **Chapter 4:** describe the two approaches we have been using to tackle the handover problem, along with the experimental setup and a discussion on the results.
- **Chapter 5:** will state the conclusions based on the obtained results and will briefly re-state the contribution of the work.
- **Chapter 6:** will take a look at the possible future works that may spawn from this thesis.

CHAPTER 2

BACKGROUND AND BASICS

2.1 Hardware Description

2.1.1 Baxter Robot

The robot we have been using during the entire research is the Baxter robot built by Rethink Robotics, in particular the Baxter Research Robot version. This is a humanoid, anthropomorphic robot with two seven degree-of-freedom arms, with all rotational joints, and state-of-the-art sensing technologies, including force, position, torque sensing and control at every joint, cameras in support for computer vision applications, integrated user input and output elements such as a head-mounted display, buttons, knobs and more [19].

This robot is widely used for research purposes due to its user friendly interface and the wide support provided by the vendors which includes useful built in functions such as: inverse kinematics solver, end-point position, orientation and joint angles retrieval. Among these, the inverse kinematic solver is very useful, since building an algorithm from scratch can be quite tedious and it would require a research all by itself. On the other hand, vendors themselves state that this tool has been made only to provide an out of the box method, and for precise and custom applications a software should be developed [20]. This robot has some limitations such as a position accuracy of ± 5 mm and a maximum payload of about 2.2 kg (including

the end-effector), maximum rotational speed is 2 rad/s for the elbow and shoulder joints that increases up to 4 rad/s for the wrists ones. These features make the Baxter much more suitable for research purposes rather than industrial ones, since such an hardware is not ideal for precise and fast tasks, that maybe are mandatory in industrial scenarios. On the other hand, the Baxter provides a cost effective solution in research, where high productivity is not the main goal.

Baxter robot comes with two end-effectors tools: a parallel and a vacuum gripper. For the purposes of our research we removed the left harm end-effector and substituted it with a more sophisticated one, described in the next section.

Baxter also includes an on-board computer featuring the following main components[19]:

- **Processor:** 3rd Gen Intel Core i7-3770 Processor (8MB, 3.4GHz) w/ HD4000 Graphics.
- **Memory:** 4GB, NON-ECC, 1600MHZ DDR3.
- **Hard Drive:** 128GB Solid State Drive.

This on-board computer runs the ROS master (see *Software Description* section for details) and it is connected to our desktop computer via an ethernet cable.

2.1.2 ReFlex SF

As we mentioned in the previous section, Baxter's left end-effector has been substituted with a more sophisticated hand, in particular the ReFlex SF by RightHand Robotics. This is a 3 fingered 4 DOF robotic hand. Each finger is controlled by a single actuator that drives a tendon

spanning both the proximal and distal joint. It is possible to control directly only the first half of the finger, the second half, instead, passively conform itself to the object's shape accordingly to the strain applied to the tendon. The first three joints approximately spans from 0 radians (finger fully opened and parallel to the palm) up to nearly π radians (finger fully closed and resting on the palm). The fourth DOF is the so called *"pre-shape"*, it allows to change the angle between finger 1 and 2. For this particular model the *"pre-shape"* actuator is shared between finger 1 and 2, so is not possible to move the former letting unchanged the latter. This joint can again span from 0 to π radians, the first configuration is obtained when finger 1 and 2 are aligned with finger 3, the second when they both are perpendicular (Figure 2). Every finger is covered with a soft and rubbery material that not only increase the friction between fingers and object, enhancing the grip, but also introduce some compliance, so the fingers can better adapt to the various application scenarios.

Figure 2: Reflex Sf with around 90° of pre-shape angle.

All the actuators are Dynamixel MX-28T servo motors which, accordingly to the data sheet, are controlled with a PID algorithm that allows an angular precision of 0.088°. The no load speed at 12V is 55 rpm(≈ 5.76 rad/s) [21].

This particular model does not offer any other special feature such as tactile sensors, proximal encoders or IMUs. These improvements are included with other models offered by the same vendor.

From this brief hardware description it is easy to understand why this hand is under-actuated. There are seven movable parts but only four can be directly controlled. Moreover, having a shared actuator for the *pre-shape* joint further limits the achievable configurations. Due to these reasons, it's very difficult to obtain precision grasps or a direct control of the force applied to the object, since the way fingers interact with the item strongly depends on its shape. These considerations motivated us in finding more general approaches both for the grasping and handover.

2.1.3 Microsoft Kinect V1

The Kinect V1 by Microsoft offers a very cost effective and practical solution for computer vision applications. In fact, despite is very low cost, this sensor offers a wide range of capabilities. It features and RGB camera with a 640 x 480 resolution, able to stream video up to 30 fps. All pixels for the three channels (red, green and blue) are stored as 8-bit integers (values form 0 to 255). Kinect also includes an infrared camera that in combination with a CMOS sensor can create a depth image of the environment, the detection range is about 0.8 to 3.5 meters[26]. Each pixel of the depth channel is stored as an unsigned 11 bit integer whose value directly represents the distance, in mm, form the sensor's reference frame. The depth information is very useful because it allows to easily retrieve the world coordinates from pixels ones, more details about this will be given in a specific section. For both the RGB and depth images the field of view is about 57 ° horizontally and 43 ° vertically. There is also a motor at the base to adjust the tilt of the sensor, it can span ± 27° vertically. The last feature included in the

Kinect is a microphone array with 4 sensing capsules, each channel can process 16 bit audio at a sampling rate of 16 kHz, this last feature was not used in our experimental setup.

In order to read the data from Kinect we used the OpenNI drivers, version: 1.5.4.0. These drivers directly allow the Kinect to publish topics in ROS.

2.1.4 ATI Gamma F/T Sensor

The force and torque sensor that has been used during the experimentation is the Gamma F/T sensor from ATI. It is able to measure all six component of force (F_x, F_y, F_z, T_x, T_y and T_z) using silicon strain gauges to detect the force/torque applied. The working principle takes advantage of the influence of geometry on electrical conductance. Applying a strain to the resistive foil cause a deformation and therefore a variation in the resistance, by measuring this perturbation it is possible to retrieve the force applied. The output from the sensor is a voltage that is then converted in forces and torques using the calibration file, provided by the vendor.

This sensor offers a satisfactory sensing range for robotics application, especially when dealing with robot-human interaction. The range and resolution specifications are given in the following table:

TABLE I: ATI GAMMA SI-65-5 SPECIFICATIONS [27].

	Range	Resolution
F_x,F_y	65 N	1/80 N
F_z	200 N	1/40 N
T_x,T_y	5 Nm	10/13333 Nm
T_z	5 Nm	10/13333 Nm

The sensor is connected to our desktop computer via an integral cable that goes in power supply box, which is then connected to our DAQ board (National Instruments PCI-6034E). The acquisition board is directly plugged in one of the available PXI of the computer's motherboard. Data are acquired using MATLAB by MathWorks using the *Data Acquisition Toolbox* and the additional package for interfacing it with National Instruments devices. This solution is more of a workaround than a proper setup. Our main development computer (*i.e.* connected to the robot), running Ubuntu, has a kernel version that does not support the drivers for the acquisition board. This lead us setting up another desktop computer running Windows. We acquire data in MATLAB on this latter and we stream them, using a TCP-IP connection, to our Ubuntu computer. This solution was the only way to read data from the sensor, even if it introduces some delay in the readings.

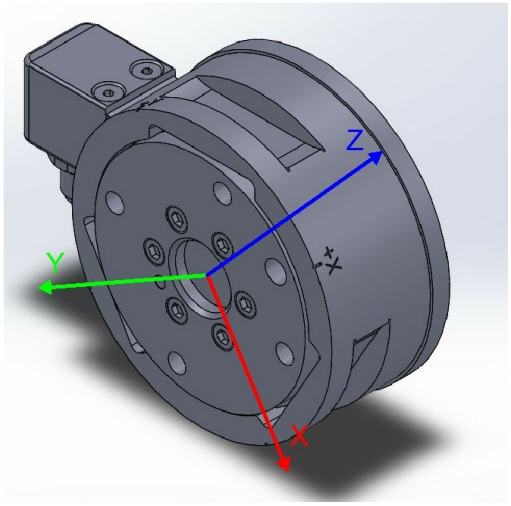


Figure 3: ATI Gamma 3D model and reference frame.

2.1.5 Optical Sensor

The last hardware element is an optical sensor obtained from a mouse, namely ADNS2051 from Agilent. The working principle of such a device is quite simple: a small, red LED illuminates the surface underneath the sensor, reflecting textural features in the area. A plastic lens collects the reflected light and forms an image on a CMOS sensor[31]; each image is sent to a DSP for analysis. The latter is able to detect salient points in the images and see how they have moved since the previous frame. Based on these changes, the DSP calculates how far the mouse has moved and writes the corresponding data to a buffer file, that is then read by the OS and traduced into cursor movements. The data written in the buffer file are 9-bit integers.

This sensor provides a resolution of 400 CPI (*Counts Per Inch*) in default configuration, that can be increased up to 800. Moreover, with the default settings images are taken at 1500 fps and processed as explained above[18].

2.2 Software Description

2.2.1 ROS Overview

ROS or Robot Operating System provides a set of libraries and tools to make easy software development in robotic applications. Despite its name, ROS is not an operating system (since it relies on a proper OS to run correctly), even if it provides OS like services such as hardware abstraction, low-level devices control and drivers and many more. ROS is currently targeted for Linux based systems since it relies on a large number of open source libraries, that may not be easily available for other operating systems. Even though, there exist experimental versions for other OS such as macOS and Microsoft Windows.

The main idea behind ROS is to allow software development in complete independence from hardware and programming languages. This makes easy to take advantages of packages or tools developed by other users, without the need to pay too much care at the specific programming languages or APIs.

ROS environment is based on the following basic components:

- *ROS master:* takes care of providing name and registration services to all the rest of the nodes in the ROS system[29]. It tracks publisher and subscribers as well as topics and services. The main role of the master is to enable individual ROS nodes to locate one

another, once this happens the master can allow a peer-to-peer communication within them. In our setup the master was running on the Baxter robot itself. There is no need to have the ROS master running on the same machine where the software is being developed since it has is own URI, therefore it is enough that it is on the same network with nodes and services it needs to communicate with. In general a ROS master is started running the command: *$ roscore*.

- *ROS node:* a node is a single process that performs computation, it can be written in C++ or Python (respectively relying on the packages *roscpp* and *rospy*), even if packages and tools exist to use other programming languages such as Java. Several nodes can run in parallel and perform various computations, the results can be sent to other nodes, via messages or services, for further manipulations. ROS nodes can either publish or subscribe to *ROS topics* that are one way for different nodes to communicate. To start a node usually the following command is invoked: *$ rosrun <packagename> <nodename>*. It is possible to check the current running nodes under the master using the command: *$ rosnode list*

- *ROS topic:* topics are named buses over which nodes exchange messages[30]. Topics have an anonymous publish and subscribe semantics, therefore the production of the message is completely decoupled from its consumption. A node that publish over a topic is not aware of which nodes will later subscribe to it, and vice versa. Furthermore, ROS allows multiple nodes to publish or subscribe to the same topic. The information is carried

with *ROS messages* or *services*. It is possible to check what has been published to a topic by invoking the command: *$ rostopic echo <topicname>*. Similarly to nodes, it is possible to see the entire list of topics handled by the current master with the command: *$ rosotpic list*. In the end, it is possible to publish a topic by command line, this is very useful for debugging purposes when someone wants to check the behavior of a node without running other parts of the system, to do so it is enough to use: *$ rostopic pub <topicname> <messagetype> <message>*.

- *ROS package:* packages are the basic building blocks in ROS. Each package can include nodes, dataset, third-party software and more. Roughly speaking, each package add a set of "skills" to ROS. Every package must include a *CMakeList.txt* and a *package.xml*, the former include instruction on how to build the code in the package and where to install it, the latter include information such as package name, authors and dependencies. To enable ROS to use tools inside packages these must be built using the command: *$ catkin_make*.

- *ROS message:* messages are the data structures that carry the information needed for the various ROS nodes to communicate. These are defined as C++ classes with specified a *field type* that describes the data type of the information (eg. float32, uint8 etc..) and a *field name* that specify how the user will refer to that particular field. ROS supports a series of primitive data types for exchanging messages; the most common among these are listed in the following table:

TABLE II: MOST COMMON ROS MESSAGE TYPES [28].

Primitive Type	**Serialization**	**C++**	**Python 2 and 3**
bool	unsigned 8-bit int	uint8_t	bool
int8	signed 8-bit int	int8_t	int
uint8	unsigned 8-bit int	uint8_t	int
int16	signed 16-bit int	int16_t	int
uint16	unsigned 16-bit int	uint16_t	int
int32	signed 32-bit int	int32_t	int
uint32	unsigned 32-bit int	uint32_t	int
int64	signed 64-bit int	int64_t	long int
uint64	unsigned 64-bit int	uint64_t	long int
float32	32-bit IEEE float	float	float
float64	64-bit IEEE float	double	float
String	ascii string	string	string

Moreover, ROS comes with a built-in package called *std_msgs* which includes the data types shown in table II and other basic message constructs (eg. Int16MultiArray). It is possible for users to create custom messages to satisfy individual requirements. This can

be done by including in the package of interest a folder called *msg* and place inside it a series of text files that describe the messages. Since they are compiled in C++, some additional lines in the *CMakeList.txt* and *package.xml* are required to make them work.

- *ROS services:* services are another method for various nodes to communicate. In this case, a node directly request an interaction with another node that can directly reply to it, this is achieved by defining a pair of messages, one for the request and another for the reply. Services are built in a similar way as messages.

From this brief overview into ROS it appears how it has been designed to be a distributed system. This design pattern offers several advantages while developing applications. For example, the communication is entirely done with messages and services that are independent from the programming language (even if they are compiled in C++): a node written in Python can easily subscribe to a topic published from a node written in C++. It is also possible to debug one node at the time without the need of running the entire system. Furthermore, the use of packages, in combination with the very active community, helps to save a lot of time since it is possible to look for a package that already implements certain functions, without the need to start from scratch.

ROS also offers a some built-in tools for 2D and 3D visualization, live plotting and analysis of sensor data. Among these tools the one we've been using the most is *RVIZ*, that is a 3D visualizer tool that can show, in real time, sensor data and state information. In our context, it has been used mainly to check the images from the Kinect.

For our experiments we used ROS Indigo whose primary target is Ubuntu 14.04 (LTS) that is the exactly the operating system we were running on our desktop PC. In the future, we plan to upgrade both the OS and the ROS version to a more recent one.

2.2.2 MATLAB

MATLAB by MathWorks is a well known tool in the scientific community. It is at all effect a program that produces files with their own extensions (.m or .mat usually), and with is own GUI. Inside the MATLAB environment it is possible to write scripts to perform the most various computations, it support by default vector calculus (*i.e.* to multiply two matrices no loop are required) complex number and, by adding suitable toolboxes, a wide variety of scientific branches can be reached (from image processing and computer vision to control theory and mechanical designs).

MATLAB has its own programming language, that is largely based on Octave. It is not a compiled language but a scripting one, therefore it does not need a compiler, even if some core MATLAB functions, written in C or C++, are pre-compiled as MEX (MATLAB Executables). This language is designed to be at the most high level possible, in order to make easy the scripting also to users without a strong background in programming.

Even in this case there is a strong and active community that helps developers to save time by downloading pre-programmed toolboxes or functions to implement a wide variety of tasks.

In the context of our work, the most useful toolbox has been the *Robot Operating System (ROS) Support from Robotics System Toolbox*. As the name says, this toolbox allows a direct

communication between MATLAB and ROS. When using the function *rosinit()* MATLAB creates a node under the ROS master, if one does not exists is MATLAB itself to create one. Once the initialization is done it is possible to publish, subscribe and use all the core ROS functionalities inside the MATLAB environment. This is very useful since, as we will explain later, the most part of the computation is done inside MATLAB but, in the end, to communicate with the Baxter and the ReFlex SF ROS messages are needed.

The main limitation is that to run MATLAB's scripts it's mandatory to have MATLAB installed on the machine. This implies to have at least around 10GB of free space or even more (depending on the number of toolboxes installed). Therefore algorithms developed in such an environment are not suitable for easy "transportation", since they depends on MATLAB's GUI to run. On the other hand, MATLAB offers embedded tools to create C or C++ executables from scripts, the main drawback is that they are quite difficult to debug and understand.

For the experiments and the implementations we used MATLAB R2017b, since it was the one UIC kindly provided the license for. Even if it is a 2 years old version there are no substantial differences with respect to most recent ones, to the best of our knowledge.

2.3 Basic Concepts

2.3.1 Rotation Matrices and Reference Frames Transformations

Rotation matrices represents a linear transformation that allows to map a given point in a reference frame, into a new rotated one. Consider the following picture:

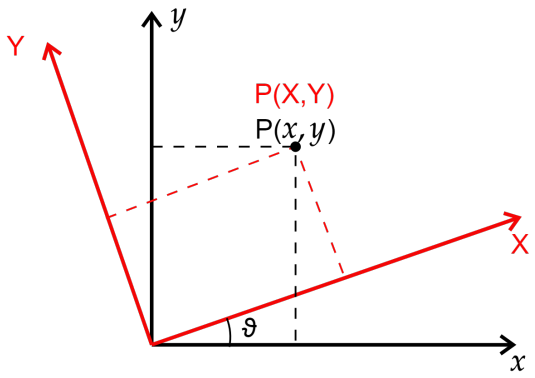


Figure 4: Plane rotation of two reference frames.

Now we want to express the coordinate of P in the $x-y$ frame, by knowing its coordinates in the $X-Y$ frame and angle of rotation θ between the two. Doing simple trigonometry it is possible to find out that:

$$x = Xcos(\theta) - Ysin(\theta) \tag{2.1}$$

$$y = Xsin(\theta) + Ycos(\theta) \tag{2.2}$$

The previous equations can be rewritten in the following matrix form:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \underbrace{\begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix}}_{R} \begin{pmatrix} X \\ Y \end{pmatrix} \tag{2.3}$$

Where in 2.3, R represent the rotation matrix between the two reference frames. Obviously Figure 4 and 2.3 refer to a planar case. If we pretend Figure 4 to represent a three dimensional space (with the z and Z axis coming out of the paper) we'll have the following relation within the coordinates:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \underbrace{\begin{pmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{R_z} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{2.4}$$

By solving 2.4 one can find out that $Z = z$. R_z represents the elementary rotation around the Z/z axis. We also have two other elementary rotations, respectively around x and y:

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{pmatrix} \qquad R_y = \begin{pmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{pmatrix}$$

By opportunely composing these elementary matrices it is possible express any complex rotation, even not around the three main axis.

All rotation matrices $\in \mathbb{R}^{3x3}$ (or $\in \mathbb{R}^{2x2}$ in the planar case) have, by definition, the following salient proprieties:

- They are orthonormal, this means that $R^T = R^{-1}$, or in other words $RR^T = RR^{-1} = I$. Moreover, multiply a vector for one of these matrices does not affect its magnitude, but only the orientation and/or direction.
- $det(R) = \pm 1$. Usually rotation matrices are constructed such that the determinant is $+1$, this guarantee to have a right-handed orthogonal triad. If the determinant is -1 the associated reference frame follows the left-hand rule instead of the usual right-hand one. Roughly speaking, this means that rotation becomes positive clockwise.

In the previous examples we did not take in account a possible translation between the origins. If this happens, we can write the transformation between two frame with following, more generic, linear relation:

$$P' = RP + t \tag{2.5}$$

Where $t \in \mathbb{R}^{3x1}$ represents the translation vector between the two frames.

If more transformations are needed, for example in kinematic chains, it is enough to apply 2.5 successively within the different frames that compose the chain. In these cases is usually preferred to use *homogeneous* coordinates. This latter representation allows to use a single transformation both for rotation and translation.

2.3.2 The Pinhole Camera Model

Let us consider the problem of retrieving 3D world coordinates with respect to the camera reference frame O_c, given the pixel point P_i in the image plane. Figure 5 resume how a generic point P_c is projected on the image plane.

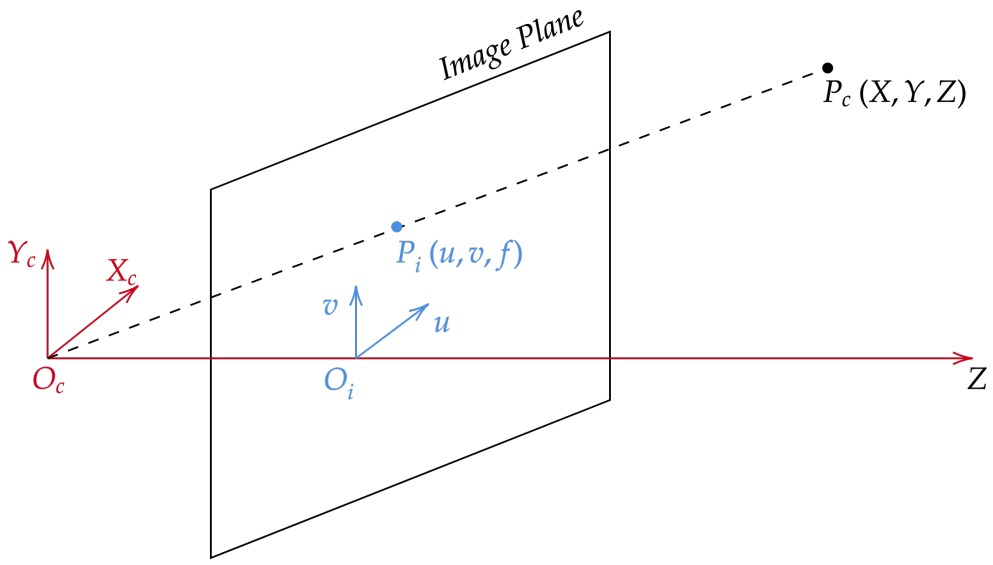


Figure 5: Projection of a 3D point P_c into to the image plane.

Given the pixel coordinates of $P_i = (u, v, F/P)^T$, where F is the focal length of the camera, defined as the distance between the camera RF and the image plane, in some world units (m, mm, inches etc): $F = \overline{O_c O_i}$. P is the size of a pixel in world unit. Now we want to obtain the original 3D coordinates of $P_c = (X, Y, Z)^T$ that is the point whose projection produces P_i.

First thing first, notice that the two reference frames, namely O_c and O_i, share the same Z axis, the only difference is an offset of a factor F. If we now redraw the scheme focusing on the $X_c - Z$ plane , we'll get something like Figure 6.

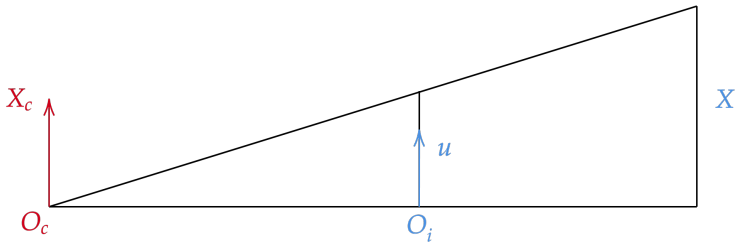


Figure 6: $X_c - Z$ plane of picture Figure 5.

We can now use the proprieties of similar triangles in the $X_c - Z$ plane to retrieve X from u. We have that:

$$\frac{X}{Z} = \frac{uP}{\overline{O_c O_i}} \longrightarrow \frac{X}{Z} = \frac{uP}{F} \tag{2.6}$$

Therefore:

$$X = \frac{uZP}{F} \tag{2.7}$$

The very same situation happens when focusing on the $Y_c - Z$ plane, we just need to substitute Y to X and v to u, therefore:

$$Y = \frac{vZP}{F} \tag{2.8}$$

If we call $f = F/P$, that is the focal length expressed in pixels we can rewrite 2.7 and 2.8 in the following way:

$$X = \frac{uZ}{f} \tag{2.9}$$

$$Y = \frac{vZ}{f} \tag{2.10}$$

Now we have to consider that the Z axis does not usually intersect the image plane at its center, but rather more often in the bottom left corner, so we need to translate u and v by this distance:

$$X = \frac{(u - c_x)Z}{f} \tag{2.11}$$

$$Y = \frac{(v - c_y)Z}{f} \tag{2.12}$$

Where the point (c_x, c_y) is the so called optical center (expressed in pixel values).

These equations are known as the *Pinhole camera model*. They can be rewritten in the following matrix form using homogeneous coordinates:

$$\begin{pmatrix} uZ \\ vZ \\ Z \end{pmatrix} = \underbrace{\begin{pmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{K} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.13}$$

The matrix K is know as *intrinsic camera parameters matrix*, and usually has a coefficient s in position 1x2. This is the *skew coefficient* that is non-zero if the image axes are not perpendicular. Since this is not our case, s has been deliberately omitted during the derivation. f, c_x and c_y are know as intrinsic camera parameters.

From the previous equations it is easy to see how, when projecting a 3D point in a 2D space, at least the depth information is lost. Equations 2.11 and 2.12 can be solved only when the Z coordinate (depth in the case of the used reference frames) is known. Fortunately, the Kinect provides this information using the IR camera, therefore the previous equation can be easily solved.

To retrieve the camera parameters usually a calibration procedure is required. Fortunately, the OpenNI drivers provide some default parameters that lead to a satisfactory estimation. These values can be read by subscribing to the camera info ROS topic. Doing so we obtained the following:

- $f_x = f_y = 525$ pixels
- $c_x = 319.5$ pixels
- $c_y = 239.5$ pixels

2.3.3 Robots Kinematic

Robots are composed by *kinematic chains*, which are a series of ideal arms/links connected by ideal joints. Kinematic studies the relation between the task space (*i.e.* physical 3D world where the robots move) and the joints space (*i.e.* mathematical structure whose elements are the joint variables). Kinematic does not care about the causes of motion (*i.e.* forces and torques), but only in the mathematical relation between the two spaces. In robotic applications there are two kind of kinematic to care about:

- **Position Kinematic:** it's the relation within the end-effector position and orientation (or another significant robot point), with respect to some world inertial reference frame, and the joint variables that produce this pose.
- **Velocity Kinematic:** it's the relation between the end-effector speed (both linear and angular), with respect to some base reference frame, and the joint speeds that produce it.

In both cases we have a direct kinematic, that maps position or velocity from the joint space to the task space, and the inverse kinematic that does the opposite. For this work we are interested in giving a brief overview about position kinematic. We can define the two following generic kinematic relations:

$$p(t) = f(q(t)) \tag{2.14}$$

$$q(t) = f^{-1}(p(t)) \tag{2.15}$$

Where:

$$p(t) = (p_x(t),\ p_y(t),\ p_z(t),\ \alpha_x(t),\ \alpha_y(t),\ \alpha_z(t))^T \tag{2.16}$$

Represent the pose (=position+orientation) of the end-effector reference frame with respect to some base reference frame. Orientation is expressed with some representation such as *roll-*

pitch-yaw or *axis-angle* or *Euler's angles* etc... In this context which representation is used is not relevant. Moreover:

$$q(t) = (q_1(t)\ ,\ q_2(t),\ \ldots\ ,\ q_n(t))^T \tag{2.17}$$

Represents the vector of joint variables, n is the total number of joints. Notice that $q_i(t)$ could represent either a linear or a rotational actuator. $f : \mathbb{R}^n \rightarrow \mathbb{R}^6$ represents a generic function that maps from joint variables to 3D euclidean space (consisting of both position and orientation).

The direct problem (equation 2.14) is quite easy, for example by using Denavit-Hartenberg parameters it is possible to find the homogeneous transformation between the base frame and the end-effector one. The inverse problem tends to be far more complex since it consists of the determination of the joint variables corresponding to a given end-effector position and orientation [24]. In particular difficulties in solving this problem arise from:

- The equations to solve are in general nonlinear, and thus it is not always possible to find a closed-form solution [24].
- Multiple or infinite solutions may exist[24].
- No solutions may exist.

When multiple solution are present we say that the robot is redundant, this means that a pose can be reached with more than one joint configuration. For some kind of kinematic chains

it is possible and easy to find closed form solutions for the inverse problem, for example if we think about a 3D printer, usually there is a one-to-one mapping between task and joint space. On the other hand, find closed form solution for complex kinematic chains maybe unfeasible, in these cases the problem is usually solved using numerical methods.

In the case of the Baxter, we surely have a redundant robot since its kinematic emulate the human one. Therefore, for a given pose we could have multiple joint solutions. For example, if we try to touch a point on the table, it can be reached with different elbow configurations. This make difficult to find good solutions. Fortunately Rethink Robotics provide an out of the box ROS service so solve the inverse kinematic of their robot.

CHAPTER 3

THE GRASPING

In this chapter we are going to take an overview to the algorithm we used to perform the grasp, the physical mounting of the hand on the robot (including the force sensor) and the experimental results.

3.1 Physical Hand Mounting

In order to properly connect our robotic hand to the Baxter we did not use the default interface plate provided by the hand's vendors. With this plate it was not possible to include the force sensor in the system. For this reason, we designed a custom plate to obtain a proper connection. Obviously, to perform the grasp a force sensor is not mandatory, so we could have excluded it from the system during this part of the work. On the other hand, it is required in the handover, so we decided to built this mechanical connection only once. This will allow us, in the future, to have a mounting that can be used to perform the grasp and follow it with the handover.

The complete mounting counts four pieces (excluding Baxter's arm) described in the following table and Figure 7:

TABLE III: FIGURE 10 LEGEND.

Label	Piece
1	ReFlex SF
2	ReFlex to Force sensor plate
3	Force sensor
4	Baxter to Force sensor plate

Figure 7: Complete mounting of the hand on the arm.

It is immediately possible to notice that this configuration moves the Baxter endpoint, since the black plate (label 4) has a non-negligible length. Perhaps, this was the only way to mount

the force sensor in the middle of the system, since the holes pattern on Baxter's wrist (Figure 8 left holes) interferes with the ones behind the sensor (Figure 8 right). Therefore, if the plate was made shorter it would have been impossible to mount the two pieces together, because the sensor will have covered the holes for screwing the plate on the robot. This plate was already in the our lab so we decided to use it and do not waste material in building a shorter one, since this will not solve the previously described problem.

Figure 8: Back plate detail.

The orange plate (Figure 7 label 2) was specifically designed for connecting the hand to the force sensor, its shape (Figure 9) is quite strange to meet some restrictions about the holes patterns. In fact, the force sensor has flange with 8 diametric holes that must not be covered in order to screw it to the sensor. Therefore, the plate in Figure 9 is a custom design to meet this

requirement. The 3D model of the this plate was designed in SolidWorks and then 3D printed at the UIC Makerspace [1] that we kindly thank.

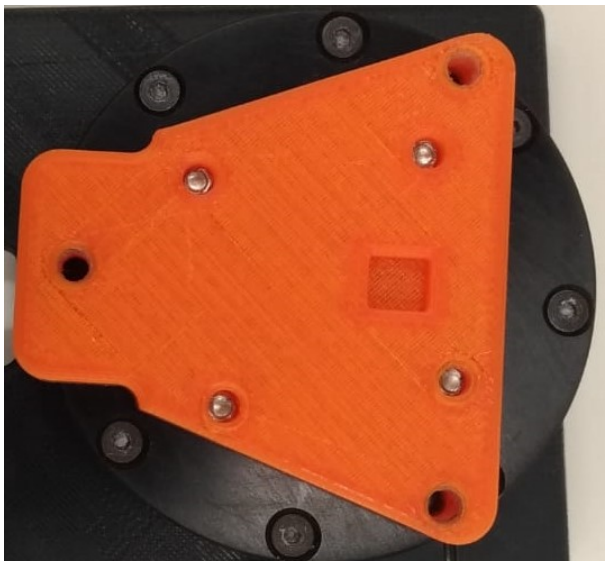


Figure 9: Orange plate screwed on the force sensor's flange. The three empty holes are used to connect it to the hand.

In the end, we used the three remaining holes in Figure 9 to connect the hand to the overall system.

---

[1] https://makerspace.uic.edu/

This solution, despite being bad looking, has performed successfully during all the experiments. The only drawback is the offset introduced by the black plate. Ideally the hand should be the continuity of Baxter's arm, exactly how it happens in humans.

3.2 Object Detection

The first thing we do, before starting up the research, is to run an object detection algorithm. This is done both to have a systematic approach to compute the bounding boxes around the object, since in the original algorithm they were manually selected, but also to have a user friendly interface where the user can select the object he/she wants to be grasped by the robot. The bounding box will later define the sub-image where to look for the grasp.

Since this task is not mandatory and does not improve particularly the grasp, we decided to use one of the many pre-trained models available at the tensorflow object detection zoo page. In particular, we were looking for a good trade-off within computational complexity and detection robustness. Some models are very robust but require an high computational power to be deployed, since our desktop computer was already running MATLAB, OpenNI and the drivers to control the robotic hand, a heavy model will require more time than the actual grasp computation and execution. On the other hand, very fast models were not robust enough for our purposes, for example they were able to detect bottles only if they where placed vertically. We found a good solution in using $faster\_rcnn\_nas\_lowproposals\_coco$ model. This is a low proposals neural network trained on the *COCO data-set*, which consists of 91 everyday objects, of 2.5 million labeled instances in 328 thousands images [22].

This solution greatly served the purpose of recognizing the objects on the table. Moreover, we used a look up table to decide which grasp to apply, cylindrical or spherical, based on the detected object. For example a ball is grasped with a spherical grasp and a bottle with a cylindrical one.

This algorithm is implemented as a Python ROS node, that is called directly in MATLAB. Once the computation has finished, it is asked to the user if the object he/she wanted has been found, if yes it can be selected by typing the corresponding number (Figure 10), if not the algorithm stops. This node publishes over a topic called */rectangle_pos*. Depending on the result of the detection, the published message can slightly change, we have the following three cases:

- **Both detected and in look-up table**: the published message contains the type of grasp and the top left and bottom right coordinates of the bounding box (in pixel coordinates).
- **Detected but not in look-up table**: the published message contains the coordinates of the top left and bottom right corner of the bounding box, the grasp type is left empty.
- **Not detected**: the published message is a 0.

We then subscribe to this topic in MATLAB. If the message is a 0 the research sub-image is manually selected by the user. If the grasp type is empty (cases 2 and 3), it is computed later using an analytic method.

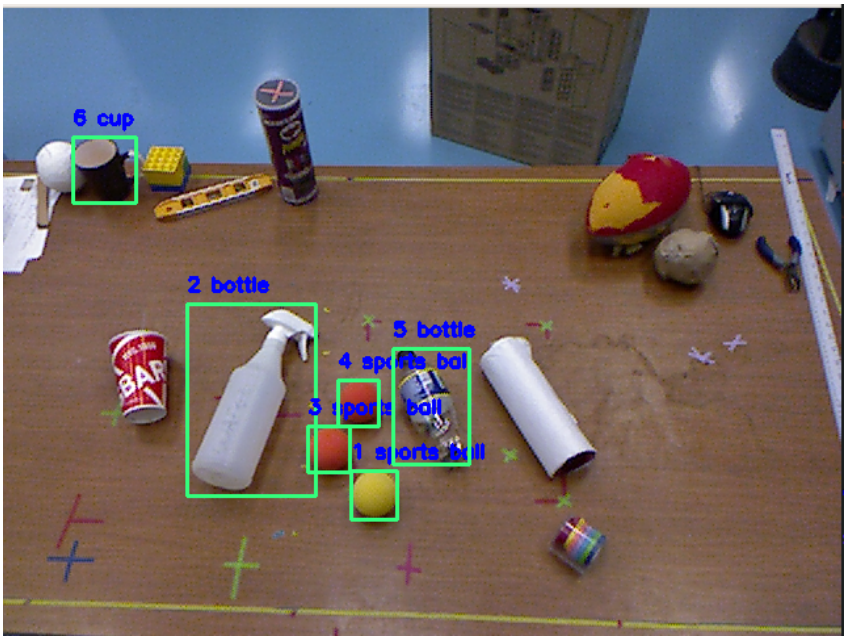


Figure 10: Objects detected on the table. The number over each bounding box is the one the user have to select for grasp it.

3.3 Grasp Research

Once the region of interest for the search is defined we did some pre-processing before deploying the real search. In the original implementation [7] angles, width and height of the grasping rectangles were hard-coded. This could lead to a long search since very improbable angles and dimensions are cheeked. We decided to prune the search by applying the following steps.

3.3.1 Edge detection and Contours Retrieval

Given the bounding box around the object of interest we firstly apply an edge detection algorithm. In particular, we used Robert's method that is based on the estimation of the image gradient along x and y directions, this is obtained by convolving the image with the following two masks:

$$M_x = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \qquad M_y = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \tag{3.1}$$

After the convolution, we obtained the estimation of derivatives along x and y, respectively G_x and G_y. Then the gradient magnitude is estimated using:

$$|G(u,v)| \approx |G_x(u,v)| + |G_y(u,v)| \tag{3.2}$$

The magnitude of the gradient is then compared, point by point, to a threshold. If the magnitude of the gradient is higher than this value, the corresponding pixel is set to a logical 1 (true), if not to a logical 0. In this way a binary image is obtained with the edges marked as true logical values. With this algorithm also edges inside the object are detected, instead we wanted an image where the inside of the object is filled with ones, and the outside is filled with zeros, so to have a clear distinction between object and background. To accomplish this

we then deployed a convex-hull algorithm, to find the smallest convex polygon that contains all the points found during the edge detection, we then filled this polygon with ones.

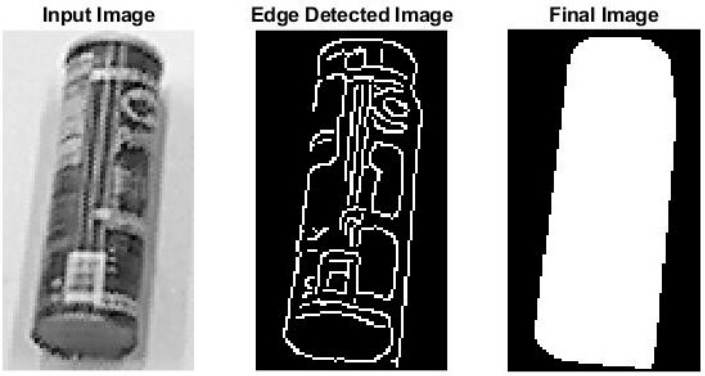


Figure 11: Results of the edge detection and convexhull application.

This approach has produced very good results in combination with a white uniform background. The latter helps the edge detection not to find points outside of the object. If the background has some characteristic patterns, it is possible to opportunely tune the edge detection threshold in a way that only salient edges of the object are found, then the convex-hull will take care of filling the object with ones.

3.3.2 Principal Axes and Dimensions Estimation

The final image in Figure 11 has been used to calculate the principal axes of the object. Firstly we use the MATLAB's *find* function to obtain the indices of the pixels that have a true value. This function returns two vectors, namely X and Y. The former contains the rows indices that contain a 1, the latter does the same but for the columns. We then apply the definition of covariance (equation 3.3) to build the covariance matrix:

$$cov(X,Y) = \frac{1}{N-1}\sum_{i=1}^{N}(X_i - \overline{X})(Y_i - \overline{Y}) \tag{3.3}$$

Where $\overline{X}$ and $\overline{Y}$ are the mean value of X and Y respectively.

The covariance matrix is then calculated as:

$$C_{X,Y} = \begin{pmatrix} cov(X,X) & cov(X,Y) \\ cov(Y,X) & cov(Y,Y) \end{pmatrix} \tag{3.4}$$

Notice that, by definition, $cov(X,Y) = cov(Y,X)$.

Once the $C_{X,Y}$ is known, its eigenvectors gives the direction of maximum variance, therefore they represents the principal axis of the object.

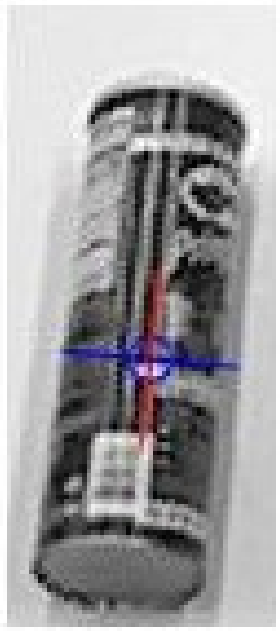


Figure 12: Principal axis of the object in Figure 11.

Furthermore, by comparing the eigenvalues of $C_{X,Y}$ it is possible to retrieve information about the shape of the object. If the two values are very different, the covariances in the two perpendicular directions are distant, this imply that the object is longer then larger. If the two eigenvalues are comparable the object is approximately round.

All these information were used to decide which grasp to apply when the object is not recognized by the object detection or not in the look-up tables. Furthermore, this helped us

to prune the search: by knowing the principal axis we can set the research angles to be in the neighborhood of their orientation. In particular, if the object is cylindrical we look for a grasp where the fingers will be placed along the axis of smallest variance, for not being too stringent we use the angle of the axis $\pm$ 5°. For the spherical objects we look for grasp along both axis always using a $\pm$ 5° tolerance. Therefore, three angles are checked in the cylindrical case, and six for the spherical one. This helped us in achieving a faster search. Obviously this approach works well under the assumption that object is composed by one main blob.

Another step we did for improving the research of the grasp is to estimate the object's dimensions. This is done with the information obtained from the previous steps. Firstly we rotate the final binary image: the rotation angle is given by one of the two eigenvectors by simply calculating the tan^{-1}. It makes no difference which one is used, the final result is always to have the principal axis aligned with the X and Y axis of the image. After doing so, we find the indices of the points that are more far apart both in X and Y direction. Having this information and applying the camera model is possible to have a rough estimate of the object dimensions. This last value is later used to set the width of the grasping rectangle to search for.

3.3.3 Grasp Rectangle Search

For finding the best grasp rectangle we used the code provided together with [7]. It is a machine learning approach that not only learns the weights used to rank the grasps, but also the features used to rank them [7]. This method consist of a *two-stage cascaded detection* where

a first small neural network is used to find the set of top candidates rectangles, using a reduced number of features. Then a larger and more robust network is used to rank these rectangles and finding the best one. The inputs to the networks are RGB-D sub-images of the original picture of the table. The sub-image is selected manually or from the bounding box found during the object detection. The input is re-scaled to be a 24x24 pixels image to fit in the networks receptive field. For each image seven channels worth of features are extracted. The first three channels are the image in YUV color space, the fourth is the depth channel and the last three are the X,Y and Z components of the surface normal computed based on the depth channel [7]. These features are obviously multi-modal (*i.e.* not of the same kind). For this reason, in [7] is proposed a structured regularization process to discourage the algorithm from learning low correlations within modalities. For further details we refer the reader to [7].

The two network are trained on the *Cornell grasping data-set*[25] which contains 1035 images of 280 graspable objects. Each image is labeled with more than one ground truth positive and negative rectangle.

The code provided include both the proper machine learning part, where the network are trained on the above data-set, and the research algorithm. We only made a few modifications to this code to make it work properly with our setup. Firstly we do not use the Kinect handle for MATLAB, but instead we directly subscribe to the image topics in ROS. Furthermore, we modified the research algorithm to look for grasp only with a surface normal that is perpendicular to the table, instead of letting it work at every possible orientation. This was done to

easily handle the offset introduced by the hand's mounting (section 3.1). In fact, by knowing that the grasp will be performed with the arm approaching the table perpendicularly it was easy to make some 2D trigonometry to compensate for this offset. Since we are only validating the approach, we thought to make it work properly for this simpler case and then, in the future, handle other orientations.

In the end, we changed the way MATLAB interface with ROS to make the robot move and the calibration procedure. These last two steps are described in the following section.

(a) (b) (c)

Figure 13: Examples of the grasping rectangles found. Green lines represents the area where fingers are gonna grasp the object.

3.4 Robot Movement

3.4.1 Kinect To Robot Transform

Baxter and Kinect have two different reference frames. Since we are using Kinect images to compute the grasp is mandatory to estimate the rigid body transformation within the two, because the IK solver needs point in the Baxter's torso frame to find valid joint solutions. This problem has been solved by performing a sort of calibration procedure that we've been calling *Kinect to Robot calibration.*

The problem that must be solved is the following:

$$P_r = RP_k + t \longrightarrow \begin{pmatrix} P_{rx} \\ P_{ry} \\ P_{rz} \end{pmatrix} = \begin{pmatrix} r_{xx} & r_{xy} & r_{xz} \\ r_{yx} & r_{yy} & r_{yz} \\ r_{zx} & r_{zy} & r_{zz} \end{pmatrix} \begin{pmatrix} P_{kx} \\ P_{ky} \\ P_{kz} \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \tag{3.5}$$

Where P_r is a point in the robot frame, P_k is a point in the Kinect reference frame, $R \in \mathbb{R}^{3x3}$ is the rotation matrix between the two frames, similarly $t \in \mathbb{R}^{3x1}$ represents the translation between the two origins. Retrieve P_k from the image is quite easy using the camera model presented in section 2.3.1, so now we need to solve 3.5 in the two unknowns R and t.

This problem is known *absolute orientation* problem. There has been many attempt to solve it; we used Horn's unit quaterions method [9] that tries to solve the following:

$$\underset{R\in\mathbb{R}^{3x3}\ ,\ t\in\mathbb{R}^{3x1}}{min} \sum_{i=1}^{M} ||e_i||^2 \tag{3.6}$$

Where M is the total number of points collected in the two frames, and:

$$e_i = P_{r,i} - RP_{k,i} - t \tag{3.7}$$

Is the residual error for a pair of points, one in the robot frame and the other in Kinect's one.

It is shown in [9] that to retrieve the rotation matrix we must compute the following:

$$N = \begin{pmatrix} (S_{xx} + S_{yy} + S_{zz}) & S_{yz} - S_{zy} & S_{zx} - S_{xz} & S_{xy} - S_{yx} \\ S_{yz} - S_{zy} & (S_{xx} - S_{yy} - S_{zz}) & S_{xy} + S_{yx} & S_{zx} + S_{xz} \\ S_{zx} - S_{xz} & S_{xy} + S_{yx} & (-S_{xx} + S_{yy} - S_{zz}) & S_{yz} + S_{zy} \\ S_{xy} - S_{yx} & S_{zx} + S_{xz} & S_{yz} + S_{zy} & (-S_{xx} - S_{yy} + S_{zz}) \end{pmatrix} \tag{3.8}$$

Where:

$$S_{xx} = \sum_{i=1}^{M} x'_{r,i} x'_{k,i} \tag{3.9}$$

$$S_{xy} = \sum_{i=1}^{M} x'_{r,i} y'_{k,i} \tag{3.10}$$

...

x'_i represent the x component of the i^{th} point (using k as subscript for a Kinect point and r for a robot point), when the points are referred to the centroid using the following equations:

$$P'_{r,i} = P_{r,i} - \overline{P}_r = P_{r,i} - \frac{1}{M} \sum_{i=1}^{M} P_{r,i} \tag{3.11}$$

$$P'_{k,i} = P_{k,i} - \overline{P}_k = P_{k,i} - \frac{1}{M} \sum_{i=1}^{M} P_{k,i} \tag{3.12}$$

Now it's enough to calculate the eigenvalues of N and find the eigenvector associated with the largest eigenvalue. The quaternion representing the rotation is a unit vector in the same direction [9]. Once the quaternion is found it is enough to transform it in the corresponding rotation matrix (for example passing through the axis-angle representation). Once R is known it is easy enough to find t solving 3.5. For further detail we refer the reader to [9]. This method guarantees to minimize 3.6 in a least square sense.

Fortunately enough, we found a pre-programmed function by Matt Jacobson [23] in the MATLAB File Exchange section. Therefore we did not have to manually implement it.

To perform the actual calibration we marked some points on the table and then we moved the robot left arm over them trying to be as perpendicular as possible to the surface. Then we used the functions in the *baxter_interface* ROS package to retrieve the endpoint position. We recorded each point, consisting of the X,Y and Z component, as a MATLAB vector. We then took an image of the table with the Kinect and manually selected the very same points. After converting them from pixel coordinates to real world ones, we applied the above algorithm to retrieve the rotation matrix and translation vector. After some trials and errors we found that 6 points gave enough precision to our purposes.

The calibration has been carried out without the hand mounted on the robot, because given the mechanical structure that connect the the two, a non negligible offset is introduced into the system, as discussed in section 3.1. If we've performed the calibration with this setup we for sure had obtained a rotation matrix and a translation vector for the new, translated, endpoint, but this transformation will only hold for a given wrist angle (namely the angle used during the calibration). Therefore make much more sense to calibrate for the true Baxter's endpoint and then use some simple trigonometry to compensate for the offset. This last task was easy by knowing that the approach vector was perpendicular to the tabletop. The distance between the wrist's endpoint and the center of the hand is $\approx$11cm. By applying the above procedure we obtained the following rotation matrix and translation vector:

$$R = \begin{pmatrix} 0,0814 & -0,8168 & 0,5712 \\ -0,9958 & -0,0428 & 0,0806 \\ -0,0414 & -0,5754 & -0,8169 \end{pmatrix} \qquad t = \begin{pmatrix} 0,2270 \\ -0,0281 \\ 0,8112 \end{pmatrix}$$

It is possible to verify that R is orthonormal and $det(R) = 1$, therefore R respects the basics proprieties of rotation matrices.

3.4.2 Robot Motion

Once the grasp is computed, the center point of the grasping rectangle is converted into a point in Baxter's frame by using the just found rotation matrix and translation vector in combination with the camera model. This point is published over a ROS topic called */grasp_position*. This message is caught by a ROS node called *move_baxter*. This program runs in background and it listens to the above topic, once a message is received the built in service for solving the inverse kinematic is called to find the joint solution relative to the given point.

The grasp motion is carried out with two movements: firstly Baxter's arm is moved above the object but approximately 25 cm over the tabletop, then the arm goes down to a height that guarantees the closure of the hand, without the fingers being obstructed by the tabletop. This two step motion guarantees that the arm does not collide with the table or the object while reaching the grasp position. Baxter's starting position is similar to the one showed in Figure 14-a. This initial state further helps in avoiding collisions with the table, since the end-point at the starting pose is way above table's surface.

Once the final position is reached the hand is closed accordingly to the established grasp.

3.5 Experimental Setup

In order to evaluate the grasps we performed two types of experiment, they both shared the setup but had two different goals: the first is intended to test the ability in grasping different objects, the second aims to test the repeatability of these grasps.

Objects were placed on the table in front of the robot, the height of the table is approximately 75 cm and the Kinect was mounted on Baxter's head approximately at 95 cm above the tabletop with an angle of around 60° pointing towards the table. After an object was placed the algorithm computed the best grasp rectangle. Then the robot moved as explained in the previous section. The only help we gave to the Baxter during the motion was someone to keep away the various cables from interfering with the grasp. If the object was correctly grasped the robot is firstly brought to a rest position (Figure 14-a) and then to the evaluation one (Figure 14-b). The first movement is obtained by manually driving the arm, so to obtain a fast motion and stress the grasp with relatively high inertial forces. The second motion was performed by imposing the joint angles to the robot, therefore the speed was the default one. This movement was chosen to be quite complex and involved a complete rotation of the wrist, so during this last motion the gravity stressed the grasp at 360°. If the object was still grasped after the two movements the trial was considered successful and the relative score was evaluated. On the other hand, if the object felt during the motion or not even grasped, the trial was considered unsuccessful and the corresponding score was set to 0.

To score the grasp we used an retrospective metric, taking advantage of the availability of the force sensors described in section 2.2.4. After the Baxter reached the evaluation position we used the first five force readings to calculate the offsets introduced by the weights of the hand, mounting plates and object itself. By averaging these readings we obtained two values: $F_{z,mean}$ and $F_{p,mean}$. The former represents the average force, due to weights, applied in the direction perpendicular to the palm. The latter is the same but for the force direction parallel to the palm (the subscript p stands for parallel), in fact we have that:

$$F_p = \sqrt{F_x^2 + F_y^2} \tag{3.13}$$

Where x and y are in reference to the force sensor frame (Figure 3).

After obtaining the offsets we started applying forces directly to the object. First we started in the perpendicular direction (F_z). We did so until the difference $F_z - F_{z,mean}$ reached 20 N ($F_{z,thresh}$) or the object slipped completely out of the hand. If the object was still grasped after the application of F_z we repeated the same in the parallel direction until the difference $F_p - F_{p,mean}$ reached 10 N ($F_{p,thresh}$) or the object slipped. The relative score is calculated as:

$$G_{score} = \frac{2}{3}\frac{F_{z,max} - F_{z,mean}}{F_{z,thresh}} + \frac{1}{3}\frac{F_{p,max} - F_{p,mean}}{F_{p,thresh}} \tag{3.14}$$

G_{score} is therefore a value $\in [0, 1]$ that is a sort of efficiency of the grasp. The choice of this metric is driven by concept of *resistance to splipping*, proposed by *Cutkosky et al.* in [2].

The evaluation position (Figure 14-b) is the same we have been using to test the handover. This is done on purpose since after the object is picked up our final goal is to hand it to a human. In this regards, we deliberately omitted the torque when scoring the grasp since, as we will see in next sections, they are not taken in account during the handover.

(a) Rest

(b) Evaluation

Figure 14: Baxter's rest and evaluation positions.

3.5.1 Different Objects Test

For this test we selected a 19 different objects for a total of 50 trials (Figure 15), some of them are in the training set of the neural networks and some other not. The objects where placed randomly, one by one, on the table in front of the robot, we only checked that they can be detected by Kinect's camera and reachable by Baxter's arm. Non spherical objects

where tested at different angles (0°=vertical, +45°, -45°, 0°=horizontal, rotation is positive clockwise), but every pair object-angle was tested only once.

The entire table of results is given in the appendix of this document. It must be interpreted as follows:

- **Object:** describe the object under test.
- **Orient:** describe the orientation of the object. If N/A the object is grasped with a spherical grasp.
- **Det:** if *"y"* means that the object has been detected correctly. If *"n"* means that the object has not been detected or detected but not correctly.
- **Succ:** If *"1"* the trial is successful. If *"0"* the trial is unsuccessful.
- **Grasp:** describe the type of grasp applied. *"C"* stands for *"cylindrical"*, *"S"* for *"spherical"*.
- $F_{z,max}$**:** self explanatory. Measured in [N].
- $F_{z,mean}$**:** self explanatory. Measured in [N].
- $F_{p,max}$**:** self explanatory. Measured in [N].
- $F_{p,mean}$**:** self explanatory. Measured in [N].
- **Sc:** score of the grasp accordingly to 3.14.

Figure 15: Objects used for the testing procedure. From left to right, top to bottom: transparent bottle, amazon bottle, green bottle, tube, soap bottle, Sbarro cup, measuring tape 2, brown mug, measuring tape 1, cube, level, plastic box, SSD box, mouse, stapler, scissors, cutter, brown ball and orange ball.

3.5.2 Repeatability Test

For testing the repeatability of the grasp we took two objects, one per grasp configuration (*i.e.* cylindrical and spherical). We marked a spot on the table and then we performed the grasp 20 times per object, being sure to re-place the object in the very same position. The two objects we chose for this procedure were the Amazon bottle and the brown ball. While performing this test we excluded the object detection algorithm so the ROI were selected manually. We did so

both for speed purposes but also to introduced some variance in the ROI, so to be sure that the algorithm performs consistently also in presence of small perturbations. In fact, it is humanly impossible to select always the same rectangle. Even in this case, the entire tables of results are given at the appendix of this document.

3.6 Multiple Objects Results and Discussion

By performing the test as described in the previous section, we obtained a success rate of 86%. With respect to the original implementation [7] we gained a 2% success rate, even if the experiment proposed in [7] is much more comprehensive, involving 100 trials and 35 different objects. With respect to [15] our implementation seems to be better, since they achieved an average success rate of 80.1% on known objects. Probably with a tailored approach and a custom data-set this result can be greatly improved. On the other hand, our goal was not to obtain an optimal grasp, but to understand if this hybrid approach could lead to satisfactory results in fewer time. In this regard, we think to have accomplished our goal, reaching a good trade-off between time spent and final result. All the above has been done in approximately two and a half months, with the working power of only one man that was not only working on this algorithm. This is probably much shorter than the time it would take to built a custom data-set and design a machine learning algorithm from scratch. Moreover, this approach has more potential in adaptability than a custom one since it can be used with our three-fingered hand, but also with the default parallel gripper. We also discovered that using the hand we gained some fault tolerance with respect to the original implementation. It could happen that

the algorithm find quite bad grasping rectangle, but thank to the physic of our hand the object can be grasped anyway.

What we consider less satisfactory is the average score, that is only 0,56. We expected not to obtain optimal grasp but such a result is quite discouraging. There are a couple of factors that concur to this result. Firstly, it is possible that the values of $F_{z,thresh}$ and $F_{p,thresh}$ were too optimistic: 20 N, that roughly corresponds to 2 kg, will easily surpass the maximum payload of the Baxter. This is further confirmed by looking at Table V, most of the object slipped during the application of F_z suggesting that 20 N is quite an high force to be stood. Furthermore, by taking a look at Figure 16, it is possible to see that scores are quite good for cylindrical objects at all the orientations, but spherical grasps provide worse results. This is quite coherent with the approach: is far easier to find and perform a good parallel gripper grasp for a long a thin object than for a spherical one. In this second case, there are infinitely many perfect grasp, but is difficult to find and drive the robot in a way such that the center of the palm corresponds to the center of the sphere. This is also empathized by the various errors that sum up over the entire procedure, for example Baxter's position accuracy, errors in the rotation matrix and translation vector, error in measuring the distance from the hand center and the true Baxter's endpoint. Considering all of these, even if a perfect grasp is computed, it is possible that the execution is not precise enough, leading to a worse grasp.

Moreover, grasps are not very human like. The robot grasp the object but in some cases not at the most intuitive location. For example, a coffee mug is not grasped by the handle but

more likely as a simple cylinder. This further suggests that our approach can be used to obtain a grasp but not correctly emulate the human behavior.

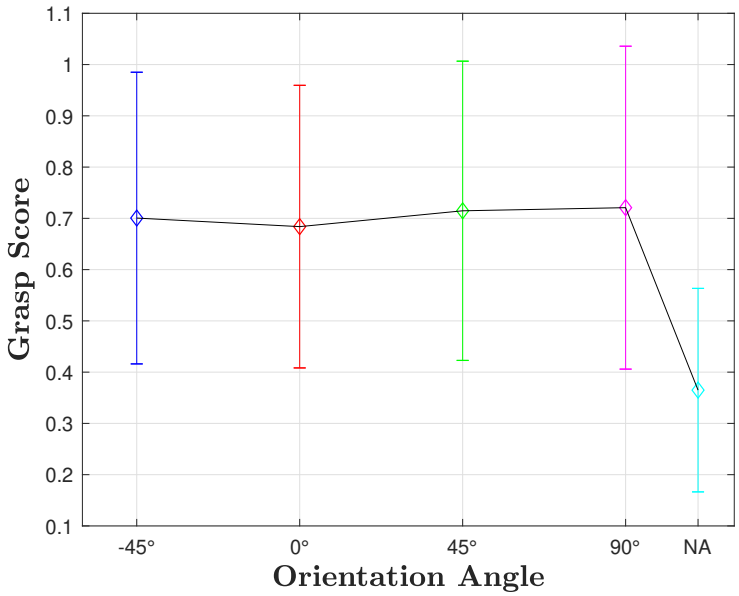


Figure 16: Average score and standard deviation for various orientations. N/A represents spherical objects for which is difficult to define a particular orientation or object grasped with a spherical grasp.

A fault we discovered during the testing concerns objects whose height is small with respect to the tabletop, for example the scissors, cutter and level. In these cases, the robot grasps

the objects only with the tip of the fingers, so the grasp is not stable and could easily slip off during the motions or with a low value of F_z. This happened even if we have been driving the hand at the lowest possible limit, a couple of millimeters more and the fingers will be blocked by the table when closing. If we think about it, also us humans have this problem; we usually workaround it by moving the object to a more comfortable grasping location. A solution for the robot could be to put these objects on some kind of pedestal, but obviously this is not a feasible solution in real world scenarios where the object must be grasped as it is. In this regard, a limitation comes also from the kinematic of the hand: ideally these objects must be grasped with a precision clamp, obtained by closing the distal joint of the fingers and letting the proximal ones almost opened. This is not feasible with our hand since, as we mentioned, the distal joints are passive and cannot be controlled directly. All these considerations suggest that in the future we might extend our algorithm to work with different approach vectors and not only perpendicular to the table, so short object can be reached from a different and more clever direction.

3.7 Repeatability Test Results and Discussion

First thing first it is worth to mention that by the time we have performed this test, we got our robotic hand broken. In particular the string connecting finger 3 to the motor ripped. The causes of the accident are not clear, probably it is simply usury. Since it was quite hard to drive the string in its own rail, due to its flexibility and the rubbery material of the fingers, we

decided to workaround this problem by letting the string pass outside the runner. Figure 17 *a* and *b* show the hand before and after the fix. This has somehow changed the kinematic of the hand, in fact the distal joint of finger 3 tends to close before than the others two for a given command. This is not a big problem during the grasp test since a grip is always guaranteed. In any case, this has slightly modified the experimental conditions with respect to the previous experiment.

(a) (b)

Figure 17: ReFlex SF before and after the fix.

By looking at Figure 18, it is quite immediate to see how the standard deviation for the spherical grasp is much wider than the cylindrical case. This suggests that spherical grasp are more prone to variate and so less repeatable. This is also confirmed by the taking a look at the single values of the score: Table VI and Table VII. In fact, even if the average score in the

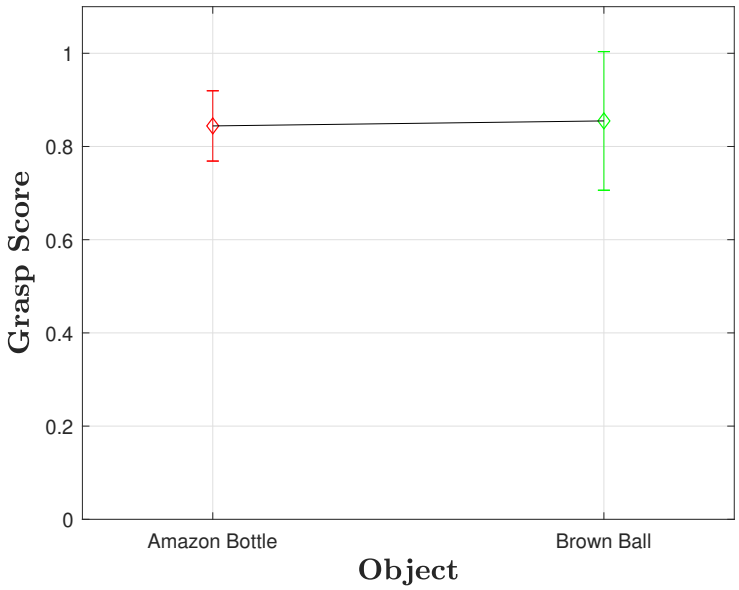


Figure 18: Average score and standard deviation.

spherical case (0,8549) is a little higher than the cylindrical one (0,8441), the former case has six values below 0,75, the latter only one. This partially confirm our idea that the algorithm performs worse in the case of round objects. Obviously, it is possible to argue that the average score is higher in the case of the brown ball and this seems to be in contrast with our previous thesis. We have to consider that when a sphere is grasped with a spherical configuration and the center of the palm is aligned with the center of the sphere, is very easy to constrain the motion in all the directions, because the fingers physically oppose to the motion. On the other hand, in the cylindrical case the direction parallel to the palm is only constrained by friction. This somehow bias our performance metric, especially when applying the force in the parallel

direction. In our opinion, this can explain the higher score in the spherical case, since when the grasp has tolerated the perpendicular force, the parallel direction physically favored.

For sure, if our aim is to declare which grasp is more repeatable, there is no argue that the cylindrical configuration is much more stable over time. As we already mentioned, this could be quite expected given the fact that is easier to find an optimal grasp rectangle on a long and thin object with respect to a spherical one.

CHAPTER 4

THE HANDOVER

Concerning the handover, we propose two different approaches. Both of them try to reach a smooth handover without using a model of the hand. In both cases, we try to introduce a failure recovery mechanism able to detect if the object is falling so to re-grasp it, preventing damages.

This task has been particularly challenging due to the palm geometry and the sensors we've been using. An optical sensor from a mouse is surely a cost effective solution but probably not the best choice for our setup. In every working place there are unused or broken mouses from which is possible to obtain this device. It is surely capable of measuring high accelerations in real time, but it performs poorly on non-plane or textureless surfaces. Moreover, the distance from the reading surface plays a crucial role, because these devices are designed to be in continuous contact with a surface, if this distance increase more than some millimeters the sensor cannot read at all.

Figure 19: Optical sensor mounted on the robotic hand.

To connect the optical sensor to the hand we designed, in SolidWorks, and then 3D printed a custom plate, taking care that it does not interfere with fingers closure. This plate was then attached to the palm using double-sided tape. We chose this solution because allows a solid connection (the hand can by lifted by the optical sensor plate), that is also easily removable or replaceable in case of failures. Mechanical solutions such as hole drilling or others, will let a permanent mark on the hand and are much more complex to design, given the geometry and the limited space on the palm. In addition, the palm is made of rubbery material, that is certainly not ideal for screws to work properly.

The sensor is then connected to our desktop computer via USB cable as the vast majority of modern mouses. Despite the fact that the right and left buttons have been cut away due to space constraint, the sensor is still able to move the cursor. Therefore is possible to treat it like a normal mouse.

4.1 Acceleration Estimate

To estimate the acceleration of the object we read the buffer file that make the mouse communicate with the OS. In this file are stored the mouse movements as Δs along the X and Y directions. These values are 9-bit integers (1 bit for the sign and 8 bit for the motion). To have a proper estimation of the acceleration we firstly made a couple of test to understand what these values represents. We hypothesized that each value represent the number of counts detected between two successive readings. To verify these hypothesis we wrote a simple program that sum up all the Δs, then we slid the sensor side by side with a ruler for a known distance. We then performed the following calculation:

$$D = \frac{\sum_{i=1}^{N} \Delta X}{CPI} \tag{4.1}$$

Equation 4.1 is written for the X direction but it obviously holds also for the Y one.

CPI is the number of counts per inch of our sensor, from the data-sheet we discovered that this value was 400 [18]. Using equation 4.1 we estimated the traveled distance and compared it with the length of the ruler to see if our initial guess was correct. We repeated this test multiple times with different distances and sliding the sensor at various speeds, every time we obtained a good estimate of the traveled path, therefore our hypothesis was correct.

With this in mind we then proceed in estimating the acceleration using the following consideration. A single reading from the mouse file, so a ΔX or Y, represent the number of counts, in bit, between two successive readings, so it represents mouse's speed in computer units:

$$v_{x,bit} = \frac{num.\ counts}{T} = num.\ counts \cdot f = \Delta X \tag{4.2}$$

Where f is the frequency used to pull the mouse data. Then we wanted to convert the above speed in world units, therefore:

$$v_m = \frac{num.\ counts}{CPI} \cdot 0.0254 \cdot f = \frac{\Delta X}{CPI} \cdot 0.0254 \tag{4.3}$$

Where 0.0254 is the conversion factor from inches to meters (since CPI represents the counts per inches), the subscript m indicates that this value is in $\frac{m}{s}$.

We then estimated the acceleration as follows:

$$a = (v_{m,t+1} - v_{m,t}) \cdot f = (v_{bit,t+1} - v_{bit,t}) \cdot \frac{0.0254}{CPI} \cdot f \tag{4.4}$$

Equation 4.4 returns a value in $\frac{m}{s^2}$ that is directly usable. To check if these calculations where correct we dropped some objects in front of the sensor and checked if the acceleration was roughly 9.81 $\frac{m}{s^2}$.

Obviously we cannot have an analog output since, by definition, bit are discrete. Anyway this was more than enough for our purposes. Acceleration values are published under a ROS

topic called *mouse_readings*, to which we then subscribed in MATLAB. The value of f we used was 100 Hz.

4.2 Approach 1

In this first methodology we simply use thresholds for deciding whether or not someone is trying to pick the object and, if it is falling or not. The algorithm works as follows. Firstly we use eight force readings to obtain an estimate of the average force applied to the hand, exactly as in section 3.5. After obtaining the values $F_{z,off}$ and $F_{p,off}$ (where the subscript off stands for offset) we start reading the forces, always keeping separate the perpendicular component (F_z) and the parallel one (F_p). At each iteration we check if the following condition:

$$(F_z - F_{z,off}) > F_{z,th} \quad or \quad (F_p - F_{p,off}) > F_{p,th} \tag{4.5}$$

If this happens we increase a counter (called N_p) that indicates how many successive readings have passed the threshold. If a reading does not verify condition 4.5 we reset this counter to 0. This last variable aims to add some compliance to external perturbation, since the hand is not opened immediately when an high enough force is detected.

If enough successive reading verify 4.5, the hand is opened and we start monitoring the acceleration of the object. Again if we find that the acceleration pass a predefined threshold value the object is declared falling and the hand is re-closed. The loop continue until the user stop the program.

This approach is probably the most basic that can be tried in this context; nevertheless it is worth to give it a try especially for this reason. We think that by selecting the right values of the variable N_p and good thresholds we can achieve a smooth handover and also a good robustness to external perturbations.

4.3 Approach 2

In this case we decided to use a more sophisticated control law, that does not involve any threshold but simply update the state of the hand based on force and acceleration readings. The law we use to control the joint of the hand is:

$$J_{t+1} = J_t - \alpha\Delta|F_z| - \beta\Delta|F_p| + \gamma|a_y| \tag{4.6}$$

Where $\Delta F_z = F_z - F_{z,off}$ and $\Delta F_p = F_p - F_{p,off}$, and J is the joint angle of hand.

The idea behind this control law is rather simple. If someone is trying to pick the object ΔF_z and/or ΔF_p increase, therefore the values of the joint angle should diminish and the hand tends to open up. The acceleration term tends to have the exact opposite effect, higher it is the more the hand tends to close.

We used absolute values in order to make sure that the effect is the desired one. Unwanted directions of the force, such as compression, are handled separately with an *if* statement. This is not done for the acceleration, so we can use the same algorithm when the hand is oriented as in Figure 19 or rotated by 180°.

We chose to use a linear relationship between the force applied and the hand state driven by the considerations in [8] and [17], where they showed that, during human to human handovers, the relation between load and grip force is linear. Hopefully, by applying the above relation between joint angle and force applied we emulate this behavior.

Again this approach is rather simple, but it's an improvement with respect to the previous one, since it uses a dynamic update instead of a static one. With this law we expect to be able to get the object by simply compensating its weight, for example placing a finger below the it.

4.4 Experimental Setup

Our goal is to evaluate three characteristics of the algorithms, namely: smoothness, resistance to external perturbations and ability to re-grasp the object. For these purposes we conducted three different tests. All these were performed using as object the tube (Figure 15 fourth object in the top row). We chose this item since it is quite long and also plenty of texture, so the optical sensor can get proper readings.

Firstly we performed a test for the smoothness. This test simply involved trying to pick the object first by applying a force perpendicular to the palm (Figure 20-a), and then a force parallel to the palm (Figure 20-b). We repeated this test three times per algorithm, each time a different weight was placed inside the tube, so to understand if this affects the smoothness of the handover. The weight of the empty tube was $42 \pm 1g$ and then we used two other weights, 200 and 500 g, we also tried to use a 1 kg weight but it exceeded the maximum payload of the Baxter and the arm could not lift up.

(a) (b)

Figure 20: The two direction of pulling for smoothness test

The second test is performed to understand the resistance of the algorithms to external disturbances. We simply pushed Baxter's arm randomly and checked the behavior of the hand, if it opened and if the object was re-grasped. For this test we only used the empty tube.

Last but not least, we checked if the two approaches are able to re-grasp the object. In this case the hand start in the open state, we then dropped the object 10 times in front of the optical sensor. We obviously expected that the hand was able to re-grasp the tube before it felt down.

To test the algorithms we used the following parameters, V_1 refers to the thresholding algorithm, and V_2 to the dynamical law one:

TABLE IV: PARAMETERS USED FOR TESTING.

V_1	
$F_{z,th}$	0.5 N
$F_{p,th}$	0.5 N
N_p	3
$a_{y,th}$	3.8 $\frac{m}{s^2}$

V_2	
α	0.4
β	0.1
γ	0,3

We decided to use low force thresholds for the V_1 algorithm in order to make the handover as smooth as possible. Furthermore, the value of $a_{y,th}$ is chosen accordingly to the human experiment in [8] where, during human-to-human handovers, the acceleration has never exceeded values around $4\frac{m}{s^2}$, we kept this value a little lower.

α, β are chosen by looking at the average force applied during the handover and degrees range of the hand (approximately $1.1rad$). We tuned them such that when the average force is applied, the hand opens enough to take the object. For γ we did the same reasoning always using $4\frac{m}{s^2}$ as reference.

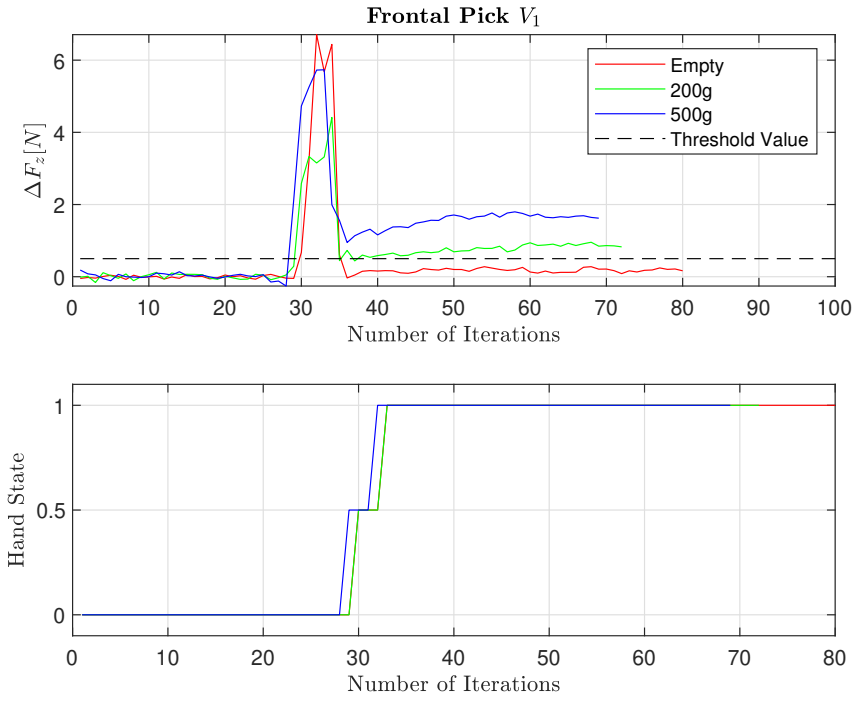
In all the cases the position of Baxter's arm was fixed to the evaluation pose we used during the grasp experiment (Figure 14-b).

4.5 Smoothness Test Results and Discussion

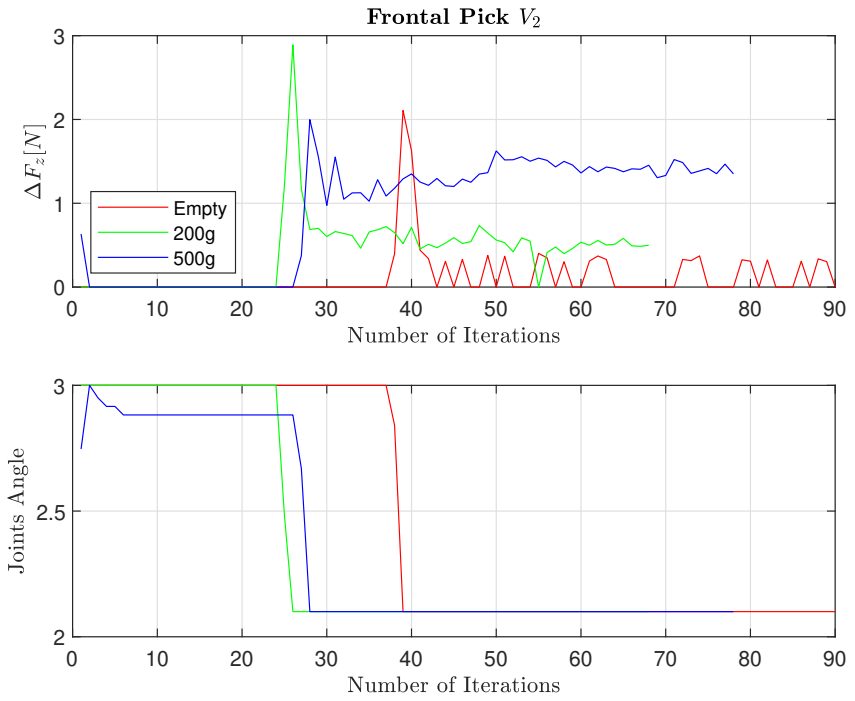
Figure 21 and Figure 22 show the results obtained during the smoothness test. For what concerns the V_1 algorithm the hand state must be interpreted as follows:

- HandState = 0: Hand is closed.
- HandState = 0.5: Force has passed the threshold but the hand is still closed, N_p is increasing.
- HandState = 1: Hand is open.

For the V_2 the object is let go when the joint angle assumes a value around 2, and its grasped at values around 3.

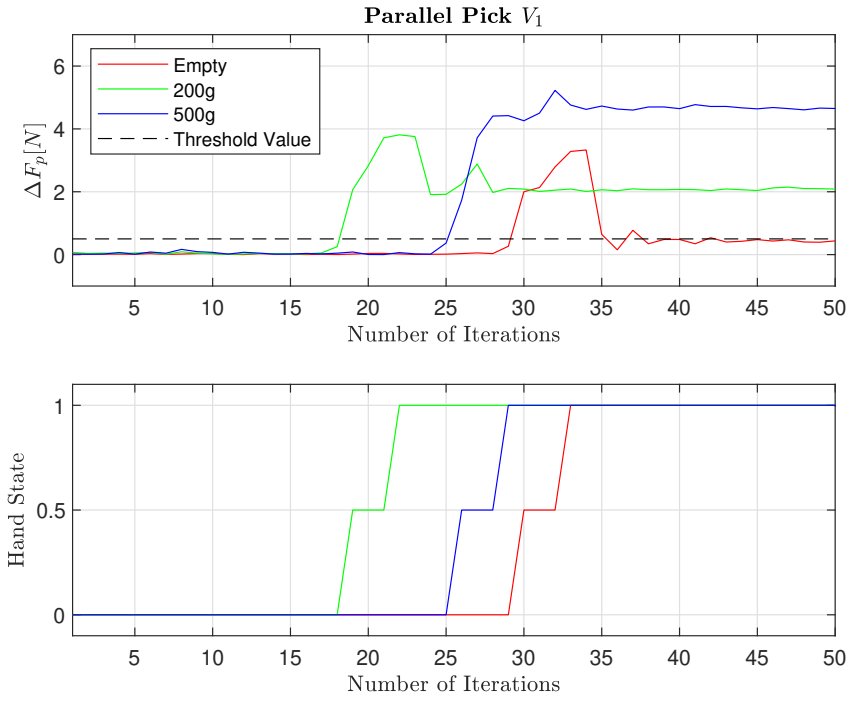


(a)

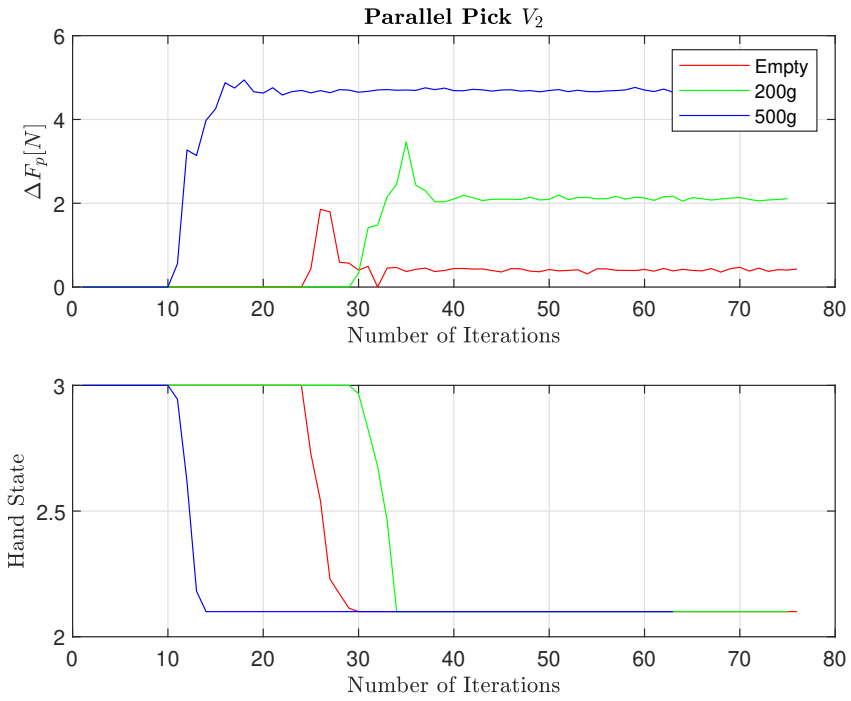


(b)

Figure 21: Frontal pick of the object with the two algorithms.



(a)



(b)

Figure 22: Parallel pick of the object with the two algorithms.

It is possible to observe how in the V_1 algorithm the forces applied, for both the pulling directions, greatly exceed the threshold value. This is caused by the presence of N_p. Since the object is not left go after the first application of the force, the user tends to increase it to get the object. This cause a very bad feeling when the release state is finally reached. Due to the greater force the acceleration at the release instant tends to be quite high, and so are the inertial forces that the human arm need to compensate for stopping the motion. This effect accentuates as the weight increases, since mass concurs to increase the inertial effects. Moreover, this effect is felt more in the parallel case, because Baxter's joints are slightly compliant, so part of the force goes into their movement. At the release instant all the force is transferred to the object and this cause an even higher acceleration. This behavior is present for all the weights we have tested and leads to a very uncomfortable handover. This can be improved by reducing the thresholds or the number of successive readings that cause the hands to open. Surely this first approach does not lead to very smooth and human like handover.

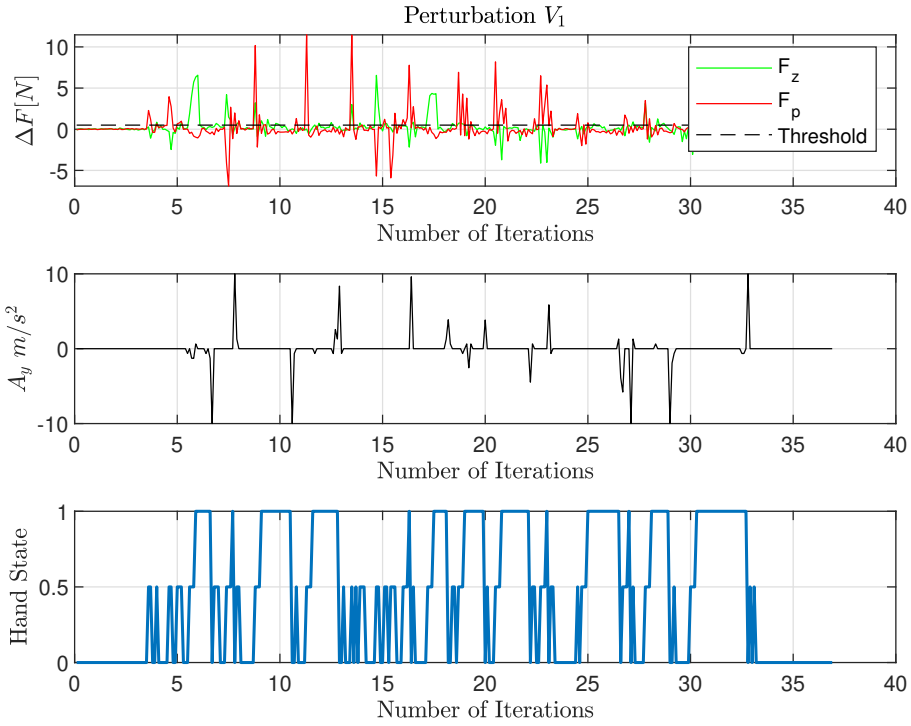
On the other hand, with the second approach the feeling is much more human like. The object is released gradually, so our arm can compensate the weight step by step, without any additional inertial effect. Figure 21 shows very well this difference: for the same weight the maximum ΔF_z applied with V_1 algorithm is almost double with respect to the other case.

We conclude that an algorithm simply based on thresholds is not enough to ensure a good handover. Even if it works flawlessly (*i.e.* the object is successfully handed to the user), the handover does not feel natural at all. The best way to describe this feeling, in our opinion,

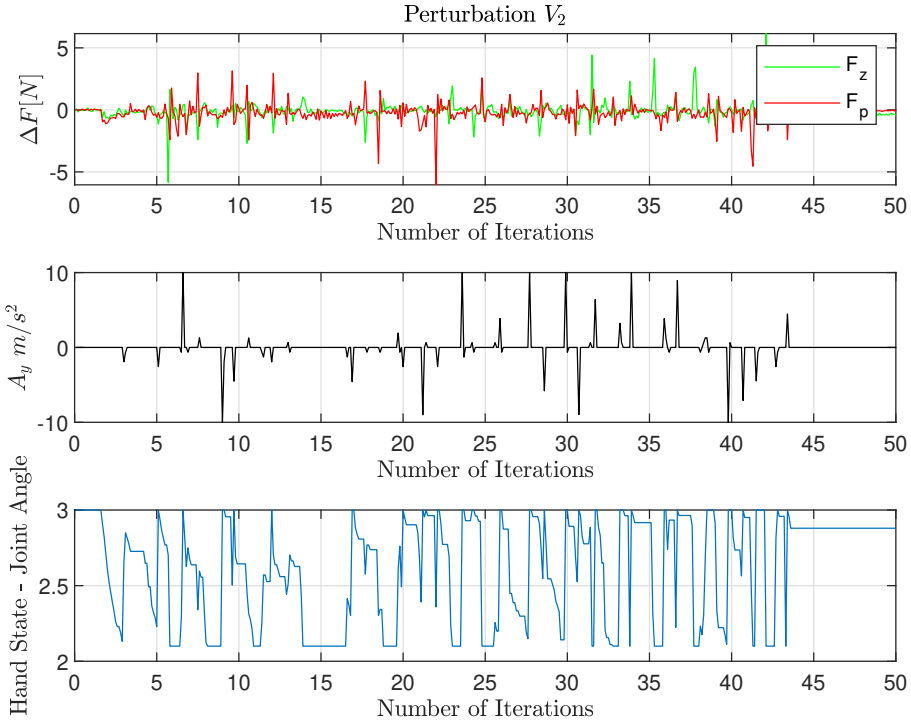
is trying to remove something that got stuck somewhere, or when someone does not want to let go an object. On the other hand, the sensation we got with the second algorithm is very similar to the usual one in human-to-human handovers. We partially expected this result: it is intuitive to understand that human-to-human handovers do not work in a binary way, but the grip is adjusted based on the force applied by the receiver.

4.6 Perturbation Test Results and Discussion

Figure 23 *a* and *b* shows the obtained results for both the version of the algorithm:



(a)



(b)

Figure 23: Lateral perturbations.

Due to the nature of the test it's difficult to have a proper comparison between the two, since the applied perturbations are randomly generated. On the other hand, this better reflects what can happens in real world scenarios.

From Figure 23 it is possible to see that both the algorithms performs quite well. When the external forces are such that the hand opens, the failure recovery mechanism triggers correctly and the grasp is promptly re-established. The first approach performs slightly better because is more difficult to open the hand, therefore there is no need to re-establish the grasp.

The biggest limitation for both the algorithms is the optical sensor itself. Two common situations leads to a fall: the sensor go in the idle state, so the first detected movement simply wakes up the sensor. The second failure mode happens when the objects fall too far form the device, so no reading are obtained. These two situations are difficult to handle properly. We think that there are mainly three viable solutions to solve these issues: the first one is to use a proper acceleration sensors instead of adapting an optical device, secondly it is possible to re-design the way the sensor is attached to the hand, trying to accommodate it in a way that allows it to always sense properly. For example, using linear springs in between the palm and the sensor so to ensure always a proper contact with the surface. Lastly, it is possible to try to install an optical device on each finger of the hand. All of the three are surely doable but required at least a preliminary trial to validate the approach, this is what this work is intended to do.

All in all, we think that the faulty part of the system is the sensor, or at least the way it interacts with the object. Both the approaches are quite good at tolerating external perturbations and recovery faulty handovers, therefore, in our opinion, the idea behind them seems to be quite good

4.7 Fall test Results And Discussion

There is not much about to say in this section, since in both the cases the object was re-grasped all the time. This confirms the hypothesis that both the algorithms are able to recovery eventual failures. Obviously this test is more biased with respect to the previous one, since object falls in a controlled way, guaranteeing always proper interaction between the sensors and the sliding object. This further confirms that the idea behind the failure recovery mechanism is good. The biggest limitation comes from the way the sensor interact with the object and, by extension, the geometry of the hand.

CHAPTER 5

CONCLUSIONS

With this thesis, we have demonstrated that it is possible to achieve good grasps using a parallel gripper planner to drive a three fingered under-actuated hand. This approach surely offers a good trade-off between simplicity and optimality. Stable grasps can be achieved for a good variety of objects. Moreover, most of the grasps can tolerate external forces and also be quite repeatable over time.

In our opinion, the main advantage of this approach is its simplicity. A parallel gripper constitute the most basic end-effector that can be mounted on a robot and a lot of work for this hardware piece has already been done. It is quite encouraging to know that is possible to take advantage of past research to perform grasps even with more complex manipulators, without the need of going through the procedure of creating a custom database and machine learning algorithm. This makes our approach easily extendable for other under-actuated manipulators, since they only need to emulate the parallel gripper behavior.

Obviously there are some drawbacks to this method. Grasp are sub-optimal and objects are grasped not always at the most intuitive location. Moreover, we discovered that by using a fixed approach and placing objects on a plane surface, makes difficult to obtain good grasps for short objects.

All these considerations suggest that our method is a good path to take when the optimality of the grasp is not the final goal of the research. In this latter case, it could be mandatory to develop a tailored data-set and algorithm for the gripper in use. On the other hand, if the task does not require an optimal grasp, this approach could easily reduce the development time. We consider the outcomes of our experiments solid enough for future studies in human-robot interactions, where a subject can communicate with the robot to make it grasp a particular object. In this regard, the presence of the object detection algorithm is very useful, since, in combination with speech recognition, can achieve a natural interaction between human and robot.

Moreover, we have demonstrated that is possible to achieve a failure recovery handover even with simple, hardware independent algorithms.

We saw that a simple thresholding approach can be used to accomplish the task of failure recovery but it lacks in smoothness, even if it is slightly more tolerant to external disturbances. We have also noticed that the variable which cause the un-smoothness is also the same that allows the algorithm to easily tolerate external, unexpected forces. This suggests that with a fine tune of this variable and the various thresholds it is possible to improve the performances of this approach on the smoothness side and, at the same time, keep its robustness.

On the other hand, an algorithm with a dynamical update of the hand state can easily accomplish a smooth handover being also quite good in tolerating external forces.

We discovered that in both cases the performances are limited by the optical sensor, so it make sense in the future to study a more clever mounting device or directly change the sensor to a more specific one.

What is most interesting, in our opinion, is the complete independence of both approaches from a dynamic model of the end-effector. This allows a easy generalization of these algorithms to other under-actuated grippers, for which is difficult to directly control the force applied to the object.

In the end, we conclude that the way to tackle this problem is surely better to use a dynamic update of the hand state. Accomplishing smoother handovers with less design effort.

5.1 Contribution

With this work we have contributed in showing that it is possible to grasp objects with a multi-fingered, under-actuated hand, using a parallel gripper planner. This technique can achieve a good trade-off between complexity and final result, that was exactly what we was trying to demonstrate with this document. Our approach is attractive both for its simplicity and the fewer time that it requires to be deployed, with respect to specialized algorithms.

Moreover, we have studied how to achieve a smooth and safe handover always using a three fingered hand. Our method was able to accomplish this goal without using any particular knowledge on kinematic and dynamic of the end-effector. So it is easily extendable to other under-actuated grippers.

CHAPTER 6

FUTURE WORKS

With this study we have partially addressed parts of the human-robot interaction that goes from picking an object, selected by the user, find the handover location and then exchange it safely to the receiver. To accomplish the whole movement we surely need to study how to compute a handover location based on the receiver configuration. This may be achieved by using the skeleton tracking with the Kinect and then setting up a human experiment where people are asked where, when and how they want the object to be handed over. With enough data will then be possible to train a machine learning model to find the most probable handover location based on the current posture of the user.

Furthermore, we are also interested in developing a set of skills that will allow our robot to easily interact with humans. As we already mentioned, this work was intended to provide it with the grasping and the handover skills.

In the future we want to expand this work, by introducing a speech recognition algorithm so the user can directly select the object to be grasped using its voice.

We are also want to improve the grasp part, since our approach has shown some limitations. Surely we want to work at different approach angles and not only perpendicular to the tabletop. Moreover, we would also expand our method for pruning the grasp search also to more complex

objects. This could involve diving the object in different blobs and compute the principal components and dimensions for each of them.

A step further can be taken by trying to study better the kinematic our hand. As we mentioned this could be difficult given its joints configuration. Probably the best way to tackle this problem is to introduce some kind of feedback, such as pressure or haptic sensors. This will allow us to understand which finger is in contact with the object and, eventually, the force applied. Another improvement could be to install sensors to track the distal joint angle, similar to the one used in cyber-gloves. This will obviously not allow us to control them, but we could try to relate the distal joint angle to different geometries and shapes.

A more precise knowledge of hand's kinematic will allow us to improve both the grasping, eventually creating custom data, and the handover, directly controlling the force.

APPENDIX

TABLES OF RESULTS

TABLE V: MULTIPLE OBJECT RESULTS.

Object	Orient.	Det.	Succ.	Grasp	$F_{z,max}$	$F_{z,mean}$	$F_{p,max}$	$F_{p,mean}$	Sc.
Bottle Green	0	y	1	C	9,26	-10,83	16,82	6,81	1,00
Bottle Green	45	y	1	C	12,43	-9,33	0,00	0,00	0,73
Bottle Green	-45	y	0	C	0,00	0,00	0,00	0,00	0,00
Bottle Green	90	y	0	C	0,00	0,00	0,00	0,00	0,00
Pringles Tube	0	n	1	C	11,71	-8,29	16,19	13,97	0,74
Pringles Tube	45	n	1	C	11,15	-8,30	0,00	0,00	0,65
Pringles Tube	-45	n	1	C	10,94	-8,14	0,00	0,00	0,64
Pringles Tube	90	n	1	C	10,67	-9,39	24,48	13,87	1,00
Brown Ball	N/A	n	1	S	9,97	-8,69	0,00	0,00	0,62
Orange Ball	N/A	y	1	S	-3,78	-8,72	0,00	0,00	0,16
Plastic Box	0	y	1	C	12,45	-7,82	22,96	14,43	0,96

APPENDIX (Continued)

Plastic Box	45	y	1	C	11,31	-8,81	21,62	14,33	0,91
Plastic Box	-45	y	1	C	11,28	-8,81	22,53	14,34	0,94
Plastic Box	90	y	1	C	3,39	-9,02	0,00	0,00	0,41
Cube	90	y	1	S	11,75	-8,95	17,51	14,20	0,80
Cube	45	n	1	S	6,59	-9,02	0,00	0,00	0,52
Mouse	0	y	1	C	4,23	-8,94	0,00	0,00	0,44
Mouse	90	n	0	C	0,00	0,00	0,00	0,00	0,00
Mouse	45	y	1	C	0,00	0,00	0,00	0,00	0,00
Mouse	-45	y	1	C	6,21	-8,80	0,00	0,00	0,50
MeasuringTape	N/A	n	0	S	0,00	0,00	0,00	0,00	0,00
Amazon Bottle	0	n	1	C	6,17	-9,06	0,00	0,00	0,51
Amazon Bottle	45	n	1	C	11,47	-9,00	24,04	13,84	1,00
Amazon Bottle	-45	n	1	C	11,51	-8,82	24,63	13,23	1,00
Amazon Bottle	90	n	1	C	11,14	-9,05	24,86	14,25	1,00
Transp. Bottle	0	y	1	C	10,47	-9,54	23,86	13,69	1,00
Transp. Bottle	45	y	1	C	10,69	-9,52	24,84	13,64	1,00

APPENDIX (Continued)

Transp. Bottle	-45	y	1	C	11,69	-8,89	23,80	13,74	1,00
Transp. Bottle	90	y	1	C	11,09	-9,27	24,58	13,68	1,00
Scissors	0	y	0	C	0,00	0,00	0,00	0,00	0,00
Cutter	0	n	0	C	0,00	0,00	0,00	0,00	0,00
Level	0	n	1	C	-0,78	-9,49	0,00	0,00	0,29
Level	45	n	1	C	-3,59	-9,19	0,00	0,00	0,19
Level	-45	n	1	C	-4,26	-9,35	0,00	0,00	0,17
Level	90	n	1	C	-2,39	-9,20	0,00	0,00	0,23
SSD Box	N/A	n	1	S	-0,98	-9,54	0,00	0,00	0,29
Brown Mug	0	n	0	S	0,00	0,00	0,00	0,00	0,00
Brown Mug	90	n	1	C	11,03	-9,83	25,82	17,13	0,98
Stapler	90	n	1	C	10,77	-9,94	19,94	17,55	0,77
Stapler	0	n	1	C	1,03	-9,83	0,00	0,00	0,36
Soap Bottle	0	y	1	C	8,56	-10,29	0,00	0,00	0,63
Soap Bottle	45	n	1	C	2,64	-10,50	0,00	0,00	0,44
Soap Bottle	-45	y	1	C	10,12	-10,45	14,51	13,94	0,70

APPENDIX (Continued)

SoapyBottle	90	y	1	C	3,87	-9,60	0,00	0,00	0,45
Brown Mug	N/A	y	1	S	2,30	-9,95	0,00	0,00	0,41
MeasureTape2	N/A	n	1	S	-1,41	-9,34	0,00	0,00	0,26
SbarroCup	0	n	1	C	12,39	-8,23	20,28	13,69	0,91
SbarroCup	45	n	1	C	11,28	-9,17	24,48	14,25	1,00
SbarroCup	-45	n	1	C	7,74	-12,79	13,67	14,66	0,65
SbarroCup	90	n	1	C	11,52	-9,44	24,14	14,04	1,00

TABLE VI: AMAZON BOTTLE REPEATABILITY

Amazon Bottle					
Trial	$F_{z,max}$	$F_{z,mean}$	$F_{p,max}$	$F_{p,mean}$	Sc.
1	10,43	-10,02	25,38	14,91	1,00
2	10,75	-9,28	19,63	14,87	0,83
3	10,46	-9,56	18,85	14,85	0,80
4	10,59	-9,54	20,17	14,81	0,85
5	12,66	-9,33	20,15	14,86	0,91

APPENDIX (Continued)

6	11,28	-9,34	21,06	14,85	0,89
7	9,23	-9,36	0,00	0,00	0,62
8	10,82	-9,25	21,46	14,76	0,89
9	11,69	-9,51	18,70	14,66	0,84
10	11,04	-9,13	22,40	14,80	0,93
11	10,95	-9,33	19,80	14,74	0,84
12	11,64	-9,31	19,73	14,65	0,87
13	13,63	-9,09	18,23	14,72	0,87
14	10,74	-9,50	18,50	14,77	0,80
15	11,16	-9,30	20,90	14,78	0,89
16	10,86	-9,28	17,55	14,77	0,76
17	12,95	-8,87	16,73	13,88	0,82
18	10,96	-9,25	18,19	14,63	0,79
19	11,52	-9,20	19,51	14,59	0,85
20	12,66	-8,99	17,54	14,59	0,82

APPENDIX (Continued)

TABLE VII: BROWN BALL REPEATABILITY

Brown Ball					
Trial	$F_{z,max}$	$F_{z,mean}$	$F_{p,max}$	$F_{p,mean}$	Sc.
1	11,27	-8,76	24,24	14,00	1,00
2	12,85	-8,90	24,33	13,93	1,00
3	9,92	-9,03	0,00	0,00	0,63
4	12,42	-8,86	15,86	13,97	0,77
5	12,01	-8,73	18,47	14,09	0,84
6	12,37	-8,77	12,92	14,03	0,67
7	12,77	-9,07	17,88	13,97	0,86
8	11,30	-8,75	14,99	14,01	0,70
9	12,35	-9,85	25,15	14,02	1,00
10	12,44	-9,24	17,17	13,97	0,83
11	11,51	-8,87	23,26	14,08	0,99
12	12,27	-8,92	24,24	13,98	1,00
13	9,31	-8,78	0,00	0,00	0,60

APPENDIX (Continued)

14	12,04	-8,90	24,04	14,03	1,00
15	11,17	-8,95	18,31	13,93	0,82
16	13,09	-9,17	24,29	14,00	1,00
17	11,20	-9,17	25,21	14,09	1,00
18	11,64	-9,40	15,27	14,02	0,74
19	12,84	-8,80	23,94	13,31	1,00
20	11,46	-8,77	13,42	14,08	0,65

CITED LITERATURE

1. Feix, T., Romero, J., Schmiedmayer, H.-B., Dollar, A. M., and Kragic, D.: The grasp taxonomy of human grasp types. IEEE Transactions on Human-Machine Systems, 46:66–77, 2016.

2. Cutkosky, M.: On grasp choice, grasp models, and the design of hands for manufacturing tasks. IEEE Transactions on Robotics and Automation, 5(3):269–279, June 1989.

3. Iberall, T.: Grasp planning from human prehension. In Proceedings of the 10th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI'87, pages 1153–1156, San Francisco, CA, USA, 1987. Morgan Kaufmann Publishers Inc.

4. Mahler, J., Pokorny, F. T., Hou, B., Roderick, M., Laskey, M., Aubry, M., Kohlhoff, K., Kröger, T., Kuffner, J., and Goldberg, K.: 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 IEEE International Conference on Robotics and Automation (ICRA), pages 1957–1964. IEEE, 2016.

5. Goldfeder, C., Ciocarlie, M., Dang, H., and Allen, P. K.: The columbia grasp database. In Proceedings of the 2009 IEEE International Conference on Robotics and Automation, ICRA'09, pages 3343–3349, Piscataway, NJ, USA, 2009. IEEE Press.

6. Mahler, J., Liang, J., Niyaz, S., Laskey, M., Doan, R., Liu, X., Ojea, J. A., and Goldberg, K.: Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. 2017.

7. Lenz, I., Lee, H., and Saxena, A.: Deep learning for detecting robotic grasps. Int. J. Rob. Res., 34(4-5):705–724, April 2015.

8. Parastegari Sina, Noohi Ehsan, A. B. e. M.: Failure recovery in robot–human object handover. IEEE Transactions on Robotics, 34(3), June 2018.

9. Horn, B. K. P.: Closed-form solution of absolute orientation using unit quaternions. J. Opt. Soc. Am. A, 4(4):629–642, Apr 1987.

10. Strabala, K., Lee, M. K., Dragan, A., Forlizzi, J., Srinivasa, S., Cakmak, M., and Micelli, V.: Towards seamless human-robot handovers. Journal of Human-Robot Interaction, January 2013.

11. Miller, A., Knoop, S., Christensen, H., and Allen, P.: Automatic grasp planning using shape primitives. volume 2, pages 1824–1829, 01 2003.

12. Carlo Ferrari, J. F. C.: Planning optimal grasps. In ICRA, 1992.

13. T. Ciocarlie, M., Goldfeder, C., and Allen, P.: Dimensionality reduction for hand-independent dexterous robotic grasping. volume 20, pages 3270–3275, 10 2007.

14. Santello, M., Flanders, M., and F. Soechting, J.: Postural hand synergies for tool use. The Journal of neuroscience : the official journal of the Society for Neuroscience, 18:10105–15, 12 1998.

15. Rai, A., Kumar Patchaikani, P., Agarwal, M., Gupta, R., and Behera, L.: Grasping region identification in novel objects using microsoft kinect. pages 172–179, 11 2012.

16. Huang, C.-M., Cakmak, M., and Mutlu, B.: Adaptive coordination strategies for human-robot handovers. In Robotics: Science and Systems, 2015.

17. Chan, W. P., Parker, C. A., Van der Loos, H. M., and Croft, E. A.: Grip forces and load forces in handovers: Implications for designing human-robot handover controllers. In Proceedings of the Seventh Annual ACM/IEEE International Conference on Human-Robot Interaction, HRI '12, pages 9–16, New York, NY, USA, 2012. ACM.

18. Agilent(Hewlett-Packard): Adns2051. `http://www.alldatasheet.com/datasheet-pdf/pdf/102949/HP/ADNS2051.html`. [Online; Accessed 03/14/19].

19. RethinkRobotics: Hardware specifications. `http://sdk.rethinkrobotics.com/wiki/Hardware\textunderscoreSpecifications`. [Online; Accessed 04/10/19].

20. RethinkRobotics: Ik service example. `http://sdk.rethinkrobotics.com/wiki/IK\textunderscoreService_Example`. [Online; Accessed 04/10/19].

21. Robotis: Datasheet. `https://docs.isy.liu.se/pub/VanHeden/DataSheets/MX-28T.pdf`. [Online; Accessed 03/14/19].

22. Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L.: Microsoft COCO: common objects in context. CoRR, abs/1405.0312, 2014.

23. Jacobson, M.: Absolute orientation-horn's method. https://www.mathworks.com/matlabcentral/fileexchange/26186-absolute-orientation-horn-s-method, 2015. [Online; Accessed 02/07/2019].

24. Sicialiano, B., Sciavicco, L., Villani, L., and Oriolo, G.: Robotics Modelling, Planning and Control. Springer Publishing Company, Incorporated, 1 edition, 2008.

25. R.L. Lab: Cornell grasping dataset. http://pr.cs.cornell.edu/grasping/rectdata/data.php, 2013. [Online; Accessed 02/15/19].

26. Andersen, M., Jensen, T., Lisouski, P., Mortensen, A., Hansen, M., Gregersen, T., and Ahrendt, P.: Kinect depth sensor evaluation for computer vision applications. , . Department of Engineering, Aarhus University. Denmark, 2012. Technical report ECE-TR-6 .

27. ATI: ATI GAMMA SI-65-6 Specifications. https://www.ati-ia.com/products/ft/ft_models.aspx?id=Gamma. [Online; Accessed 02/07/19].

28. ROS-wiki: http://wiki.ros.org/msg. [Online; Accessed 02/07/19].

29. ROS-wiki: http://wiki.ros.org/Master. [Online; Accessed 02/07/19].

30. ROS-wiki: http://wiki.ros.org/Topics. [Online; Accessed 02/07/19].

31. Bonarini, A., Matteucci, M., and Restelli, M.: A kinematic-independent dead-reckoning sensor for indoor mobile robotics. In 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), volume 4, pages 3750–3755 vol.4, Sep. 2004.

VITA

NAME	Matteo Pallomo
EDUCATION	
	M.S., Electrical and Computer Engineering, University of Illinois at Chicago, Chicago, Illinois, 2019
	M.S., Mechatronic Engineering , Politecnico di Torino, Turin, Italy, 2019
	B.S., Mechanical Engineering, Politecnico di Torino, Turin, Italy , 2017
LANGUAGE SKILLS	
Italian	Native speaker
English	Full working proficiency
	2017 - IELTS examination (7/9)
	A.Y. 2018/19 One Year of study abroad in Chicago, Illinois
	A.Y. 2017/19. Lessons and exams attended exclusively in English
SCHOLARSHIPS	
Spring 2019	Italian scholarship for final project (thesis) at UIC
Fall 2018	Italian scholarship for TOP-UIC students
TECHNICAL SKILLS	
Basic level	Project management
Average level	Basic and applied mechanics, electrical and hydraulic machines, model-based design for automotive, controllers
Advanced level	Robotics and mechatronic systems
WORK EXPERIENCE AND PROJECTS	
Jul/Sep 2017	Intership at NUOVA SICMI S.R.L, technical designer.