POLITECNICO DI MILANO

SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING

MASTER OF SCIENCE IN MECHANICAL ENGINEERING
MECHATRONICS AND ROBOTICS

AUTOMATIC MOTION PLANNING OF REDUNDANT DOUBLE-ARMED COLLABORATIVE ROBOTS FOR BIN PICKING APPLICATIONS

Candidate:
Emiliano Staniscia

Supervisor:
Prof. Hermes Giberti

Company supervisor:
Dott. Federico Polito

Abstract

The aim of this thesis is the development of a software for collaborative redundant multi-armed robot's automatic trajectory planning.

Automatic trajectory planning is one of the most interesting research branches in the robotic field. In the last years various research groups have published tons of papers describing alternative methods to face the problem of modelling complex robot kinematics and environments, managing properly their interactions and the necessity of finding in a fast and computationally-efficient way a safe, free-of-collisions trajectory for the robot. Some of the most relevant examples of path planning algorithms and techniques developed in the last decades are the RRT family of planners and the SBL one.

In most of the industrial implementations, vision systems are the best way to provide to these algorithms informations about the environment and the targets that the robot have to reach.

Of course, automatic motion planning techniques have some typical implementation problems. For these reasons, their usage in the contemporary industrial sector is limited to few practical applications. One of the most important is of course the bin picking one. Industrial robots guided by vision-based devices are already extensively used to pick randomly-oriented objects from a bin, but the typical final aim of this technology has always been the feeding of an other machine, belt or palettizer.

The main novelty of this thesis is in the development of trajectory planning methodologies for of a relatively new type of industrial manipulator: the collaborative, redundant, double-armed ABB Yumi robot, specially designed to mimic human capabilities. Yumi has the potentialities to extend the basic bin picking operation, already performed by standard industrial robots, to new industrial contexts such as the assembling or the human co-operation ones. This robot is already used for picking operations, but it's always needed to organize its working area and programming its movements a priori. Combining vision systems and automatic motion planning features with Yumi capabilities, it's possible to fully automize small objects assembly operations without designing rigid and expensive feeding systems, but with the flexibility and simplicity of a bin picking procedure.

The developed software solves some intrinsic problems related to the use of the Yumi robot in this type of applications. First of all, the necessity of managing its complex, redundant kinematic structure. Secondary and most important, the problem that standard automatic planning algorithms are not studied to interact with this type of robot. The coordination of its two arms, in fact, is of fundamental importance in order to guarantee a high productivity of the future applications, but automatic planners are typically not capable to manage more than one moving entity.

The thesis presents different approaches, developed in the last years, for the resolution of this kind of problem and proposes a method capable of bypassing it.

The software is based on one of the most efficient, well-organized and reliable open-source libraries for robot programming free available on the web: Klamp’t. Its capabilities of managing redundant, multi-armed kinematic structures have been explored and a smart procedure for double-armed coordinated bin picking applications, based on its functionalities, has been developed.

The thesis has been developed in collaboration with the ISS company, in which most of the programming and experimental works were done. ISS is a Politecnico spin-off and one of the main Italian company in the development of robot guidance systems based on vision devices with applications in the bin picking field.

The thesis has the following structure:

- Chapter 1 is a short introduction to the industrial context and the possible reasons for the developing of this work;
- Chapter 2 introduces the current state of art about vision systems and motion planning algorithms and technologies;
- Chapter 3 is a brief explanation of the main concepts regarding robot kinematics;
- Chapter 4 presents the main motion planning algorithms and describes a procedure useful to choose the most suitable algorithm for the final application;
- Chapter 5 describes the development of the software and the tests on the real application;
- Chapter 6 describes future improvement for the software.

Keywords: motion planning, robotics, bin picking.

Sommario

Lo scopo della tesi è lo sviluppo di un software per la pianificazione automatica di traiettoria per robot collaborativi, ridondanti a doppio braccio.

La pianificazione automatica di traiettoria è uno dei rami di ricerca di maggior interesse industriale nell'ambito della robotica. Negli ultimi anni vari gruppi di ricerca hanno prodotto grandi quantità di articoli scientifici, che descrivono soluzioni alternative per confrontarsi con il problema di modellare ambienti e complesse architetture robot, gestendo accuratamente la loro interazione e la necessità di trovare velocemente traiettorie prive di collisioni per il robot. Alcuni dei più importanti esempi di algoritmi per la pianificazione automatica di traiettoria sviluppati negli ultimi decenni sono la famiglia degli algoritmi RRT e l'algoritmo SBL.

Nella maggior parte delle applicazioni industriali, i sistemi di visione sono il modo migliore di fornire a questi algoritmi le informazioni riguardo l'ambiente e l'obiettivo da raggiungere.

Ovviamente le tecniche per la pianificazione automatica di traiettoria hanno alcuni tipici problemi di implementazione. Per queste ragioni, il loro impiego nell'attuale panorama industriale è limitato a poche applicazioni pratiche. Una delle più importanti è di certo quella della presa da cassone. I robot industriali, guidati da sistemi di visione, sono già frequentemente utilizzati per prelevare oggetti disposti casualmente in un cassone, ma tipicamente lo scopo finale di questa tecnologia è sempre stato quello di rifornire altre macchine, nastri trasportatori o pallettizzatori.

La principale novità di questa tesi risiede nello sviluppo di metodologie di pianificazione di traiettoria per una tipologia di robot relativamente nuova: il robot collaborativo, ridondante, doppio-braccio ABB Yumi, progettato specificatamente per emulare le capacità cinematiche umane. Yumi ha le potenzialità per estendere l'operazione di presa da cassone standard, già realizzata da robot tradizionali, a nuovi contesti industriali come quelli dell'assemblaggio e della cooperazione uomo-robot. Questo robot è già utilizzato per operazioni di presa, ma è necessario organizzarne metodicamente l'area di lavoro e programmarne in anticipo i movimenti. Combinando le caratteristiche dei sistemi di visione e della pianificazione automatica di traiettoria con le capacità di Yumi, è possibile automatizzare le operazioni di assemblaggio di piccoli oggetti, senza la necessità di progettare rigidi sistemi di movimentazione, ma con la semplicità e flessibilità di un'operazione di presa da cassa.

Il software sviluppato risolve alcuni problemi intrinseci collegati all'uso del robot Yumi in questo tipo di applicazioni. Prima di tutto la necessità di gestire la complessa struttura cinematica ridondante. Inoltre, il problema che gli algoritmi di pianificazione automatica di traiettoria standard non sono studiati per gestire più di un'entità in movimento. La tesi presenta differenti approcci, sviluppati negli anni, per la soluzione di questo tipo di problematica e ne propone uno capace di aggirarla efficacemente.

Il software si basa su una delle più affidabili, efficienti e meglio organizzate librerie open-source per la programmazione robot disponibili in rete: Klamp’t. È stata studiata la sua capacità di gestire strutture cinematiche ridondanti e multi-braccio ed è stata sviluppata una procedura per applicazioni di prese da cassone a due braccia coordinate, basata sulle sue funzionalità.

La tesi è stata svolta in collaborazione con l’azienda ISS, presso la quale sono state svolte molte delle attività di programmazione e sperimentali. ISS è uno spin-off del Politecnico e una delle maggiori aziende italiane nello sviluppo di sistemi di guida robot con applicazioni nel campo della presa da cassone.

- Il capitolo 1 è una breve introduzione al contest industrial e alle possibili motivazioni per lo sviluppo di questo lavoro di tesi;
- Il capitolo 2 introduce all’attuale stato dell’arte per i sistemi di visione e le tecniche di pianificazione del moto;
- Il capitolo 3 è una breve descrizione dei principali concetti riguardanti la cinematica dei robot;
- Il capitolo 4 presenta i più importanti algoritmi di pianificazione del moto e descrive una procedura utilizzata per la scelta del codice più adatto all’applicazione finale;
- Il capitolo 5 descrive il software sviluppato ed i test sull’applicazione finale;
- Il capitolo 6 descrive I possibili futuri miglioramenti al software.

Parole chiave: pianificazione del moto, robotica, presa da cassone.

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

1.1 THE INDUSTRIAL SCENARIO

In the last few years, a new industrial revolution, named “Industry 4.0”, has begun. One of the main ideas behind this new approach to the world of industry and manufacturing is the flexibility of the production system. In the industrial field, the necessity of having machines able to adapt their behaviour to different products, tasks and requirements is becoming more and more important, in order to ensure the maximum productivity and allow fast and smart production systems reconfigurations.

This evolution of the production system is achievable only with a strong automatization of the production process. In this context, one of the most important roles is played by industrial robots. A robot is a reprogrammable, multifunctional manipulator designed to move materials, parts, tools or specialized devices through various programmed motions for the performance of a variety of tasks.

But robots are not a new technology. They are diffused in production plants all around the world since 60’s. In these years their use has been always affected by an intrinsic rigidity due to the necessity of programming them to reach points in the workspace that must be defined a priori. The robot in fact is not able to “see” the environment, so all trajectories, needed to perform a task must be already known during the programming phase in order to correctly avoid obstacles and grasp objects. This requirement involves the necessity to configure the entire working area of the robot in order to feed pieces in specific positions and orientation; In this way they can be correctly manipulated by the robot. For this reason, are often required complex and expensive machines such us conveyor belts, palletizers, shakers or rotary tables.

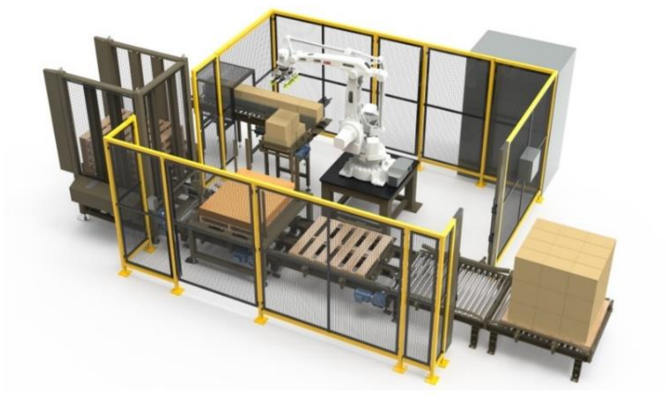


Figure 1: a robot working area

The main problem is that, if, for any reason, the production changes, also the entire sequence of machines used to feed the robot may be, at least partially, changed, with additional costs due to idle and configuration time.

In a modern production plant, this type of approach to the design of a robotic cell is no more efficient from an economic and logistic point of view. The idea that is growing fast in the last years is trying to give to the robot information about the surrounding environment and decide at each working cycle how to behave. In this way may be possible to develop algorithms able to "choose" and provide to the robot trajectories based on the current position of workpieces. The main advantage is that the position and orientation of the objects to be grasped can be not pre-defined, allowing a strong simplification of the working cell and a reduction of costs.

The most important technology developed according with this idea is the vision servoing. Artificial vision devices are capable to mimic the human sense of sight and allow to gather information from the environment without contact. Applications of vision devices are very different and already diffused in a lot of industrial fields. Is possible to use this type of sensors for line-following applications tasks (such us welding or contouring applications), or for real time feedback in order to improve position control of the end effector. But nowadays the main use of vision in industrial robotics is to detect objects in the robot scene, whose position and orientation is then used for online path planning in order to drive the robot to the identified object. This type of application is typically called "Bin Picking", because the most common work-area configuration is made of the robot, a bin in which workpieces are randomly posed and a structure to support the vision device, typically integrated in the cell used to delimit the workspace of the robot. As is possible to understand, with vision sensors, the cell configuration becomes very simple respect to a standard one, dedicated to standard programmed robot. That's because is now possible to let the robot works, without knowing a priori how workpieces are exactly posed.

This type of approach to the problem of object's grasping and manipulation trough robots, has of course a lot of advantages, but is affected by several difficulties in the implementation. First of all, there are a lot of issues related to sensor calibration, that must be faced in order to obtain a correct image acquisition in different environments affected, for example, by different light conditions. After the acquisition, is of fundamental importance to elaborate the image in order to reconstruct the geometry of the object that must be recognized. The reconstruction can be either 2D or 3D depending on the application and on the task's necessity. The reconstruction is based on complex algorithms of pattern matching able to verify that the fundamental features of the image acquired (such us lines, vertex or faces) have a correspondence with the ideal model of the object that should be provided to the elaboration unit of the sensor. Once the object has been recognized is needed to estimate its pose respect to a reference frame, usually the robot's one, that is a fundamental information for a correct trajectory planning. The planning phase requires other sets of algorithms capable of analizing the environment, typically provided by the user through CAD files, and generates a complete trajectory that must respect robot's kinematic limits and must be free of collisions with the environment.

All these passages are expensive from a computational point of view and are made by a sequence of algorithms that run on a computer, typically integrated with the sensor and physically separated from the elaboration unit of the robot, that has only the final purpose of interpolating the trajectory already elaborated. For these reasons, companies operating in this field should, not only develop the vision device, but also all the post-processing sequence and the trajectory generation. The final product should be a guidance system able, with a correct tuning, to adapt its features to different robots and applications, reinforcing the concept of flexibility that is the main idea behind all the technologies described in this thesis.

The technologies just introduced are able to eliminate all the feeding architecture of a traditional robotic cell, but the traditional robot remains a separated entity in the production plant, that must be insulated in order to ensure safe working condition for human operators. These conditions are no more efficient in modern factories that require always more integration between men and machines. In fact, this is the only way to guarantee flexibility of the plant when different, complex products should be manipulated or assembled combining the high productivity of an automated system and the capability of realizing complex and various tasks of a human being. For this reason, in the last years, the main robot producers start to develop collaborative robots. Collaborative robots are intended to physically interact with humans in a shared workspace. This is in contrast with other robots, designed to operate autonomously or with limited guidance, which is what most industrial robots were up until the decade of the 2010s.

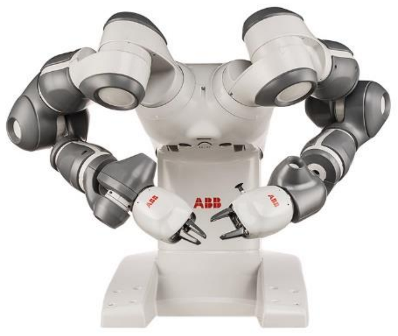


Figure 2: Yumi robot

Cobots can have many roles: from autonomous robots capable of working together with humans in an office environment that can ask users for help, to industrial robots having their protective guards removed. Collaborative industrial robots are highly complex machines which are able to work hand in hand with human beings. The robots support and relieve the human operator in a conjoint workflow.

The tendence is to give to collaborative robots more freedom in movements respect to common anthropomorphous robots. For this reason, ABB has chosen to realize its first ever collaborative robot with a double-armed kinematics. Each arm is redundant (made of more than 6 joints), in order to give them movement capabilities comparable to the ones of a human being.

1.2 Aim Of The Thesis

The technologies just introduced are already diffused worldwide. Collaborative robotics is considered as one of the nine main pillows of Industry 4.0. According to the SA Journal of Human Resource Management, "Industry 4.0 promotes technology innovation and human-robot collaboration (HRC). HRC on the manufacturing assembly line have been implemented in numerous advanced production environments worldwide. Cobots are increasingly being used as collaborators with humans in factory production and assembly environments" [1].

As already said, the idea behind the collaborative robotics is of giving the opportunity to a robot of realizing new tasks, mainly in collaboration with human workers. In this way, may be possible to combine the efficiency, repeatability and productivity of an automated system with the flexibility, capability to adapt to different situations and "sensibility" of human being. As reported in the paper "Human-Robot collaboration in industry", from A. Vysocky and P. Novak, "The main goal of this innovative strategy is to build up an environment for safety collaboration between humans and robots. There is an area between manual manufacture and fully automated production where a human worker comes into contact with machines. […] Collaborative robotics establishes new opportunities in the cooperation between human and machines. Personnel shares the workspace with the robot where it helps with non-ergonomic, repetitive, uncomfortable or even dangerous operations" [2].

Is important to underline how, although many cobots, in particular Yumi, have human-like kinematic capabilities, they are not intended to fully substitute nor human workers neither actual technologies. The idea is of enlarging the portfolio of operations that a robotic system can realize, bringing some crucial advantages in the industrial field:

- From a socio-economic perspective, the deployment of robots produces higher competitiveness of companies in comparison with countries with very cheap labor. Even a small company can focus on costumer demands and offer a product for a lower price;
- Robot's repeatable positioning accuracy and continuous operation provide better quality and lower requirements for post-processing and quality controls;
- Robots can speed up some operations and also adjust to special conditions, which can lead to the increase of productions;
- Limiting the uncomfortable, repetitive and tedious work results in lifting the burden from humans which can otherwise result in occupational disease. There is, in fact, a relation between the burden on laborers and the ergonomics of operations. Improving the working environment can lead to a decrease in the amount of occupational injuries.

Although robots are tough, fast and very accurate machines which can complete their tasks faster, with better quality and a lower price than humans, human factor remains important in some production systems. Some operations have to be adapted to actual conditions. Robots are not capable of thinking, they only execute commands, resulting in limitation of these machines. Advantages are evident when robots replace laborers in non-ergonomic duties. As an example, we can mention the manipulation of heavy payloads, manipulation in uncomfortable positions, or tasks which are dangerous. Cobots may be also installed in monotonous operations which are uncomfortably repetitive.

One of the main examples of repetitive and uncomfortable task in all industrial plants is the bin picking operation.

Provide objects to a manufacturing or assembling system posing them in a bin, is easy and cheap and pick them from the bin may seem trivial for a human being. In practical this operation results heavy and repetitive and is often subjected to errors and loses of time when realized by a human labourer. For these reasons, in many industrial contexts, such as machines feeding, it has become a fully automatized operation, performed by industrial robots guided by various types of vision systems. The situation is different in the case of manual assembling operations of small objects. This type of tasks remains a prerogative of human beings due to the necessary sensibility and capability to adapt to different situations. The problem is that feeding of parts and storing of the finished component are repetitive, monotonous and uselessly tiring operations. If a collaborative robot, adequately equipped with a guidance system, would be able to autonomously pick randomly posed objects and help the labourer to manipulate them, working conditions may improve, increasing at the same time the productivity and precision of the procedures.

Starting from this idea, the necessity of developing codes able to plan automatic trajectories for these complex kinematic entities becomes crucial. Is in fact important to underline how standard automatic planners are not yet fully studied for managing redundant double-armed robots as the Yumi one, so is fundamental to understand how take advantage of Yumi's capabilities in order to realize efficient and flexible bin picking operations easily integrable in HRC assembling tasks.

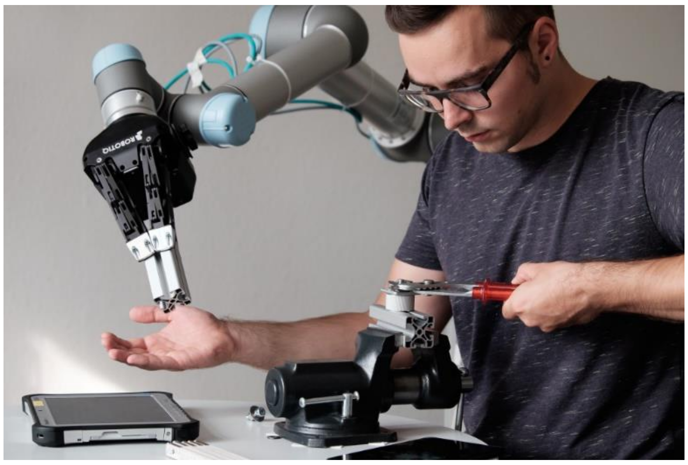


Figure 3: example of HRC

For these reasons, the aim of this thesis is of developing a program capable of integrating one of the most efficient and adaptive collaborative robots, the Yumi one, in bin picking operations, with the final future purpose of combining vision systems, automatic trajectory planning and collaborative robot technologies in co-operation tasks in the industrial context. The software will have the capabilities to elaborate informations coming from a vision system and use them to perform an innovative coordinated bin picking operation taking advantage of both arms of the Yumi.

One of the main Italian players in the bin picking field is the ISS – Innovative Security Solution srl. It is a Politecnico di Milano spin-off company which operates in the industrial automation sector and produces robot guidance systems based on 3D vision systems.

ISS, moreover, actively cooperates with Politecnico, other Universities, applied research institutes and numerous companies in developing several research projects.

Figure 4: ISS logo

Since 2010 ISS has theorized the concept of “virtual tray” or, in other words, the idea of eliminating the palletizer, giving the robot the ability to directly grasp objects randomly posed on a physical tray.

Nowadays the company provides different sensors for various application, such us bin or belt picking, mainly based on two different vision technologies: laser triangulation and stereoscopic light pattern. These types of technologies currently represent the state of art for vision systems in industrial applications.

Although its leadership in the bin picking sector, it has never faced the problem of integrate collaborative robots in this type of applications. From this point of view, the Yumi robot is the most challenging example of collaborative robotics due to its double-armed redundant configuration. The standard commercialized ISS softwares are not studied to manage this type of robots. For this reason, the developed software is completely new and ISS knowledge has only helped to define, from a theoretical point of view, some preliminary routines and procedures that are commonly useful in any kind of automated picking applications. Also the final experimental set-up has been realized in the ISS company, that provides the vision system needed to generate input files for the software and a real Yumi robot, in order to test the goodness of the developed algorithms also in terms of data transmission speed to the real system.

The software has been developed using the Klamp’t library. Klamp't stands for Kris' Locomotion and Manipulation Planning Toolbox. It is a cross-platform software package for modeling, simulating, planning, and optimization for complex robots, particularly for manipulation and locomotion tasks. It has been developed at Indiana University since 2009 primarily as a research platform and has been used in classrooms beginning in 2013. It has been used in several real-world projects, including the Amazon Picking Challenge, TeamHubo in the DARPA Robotics Challenge, and was the platform for the IROS 2016 Robot Grasping and Manipulation Challenge simulation track.

It provides various features in the fields of modelling and planning, including kinematic inversion algorithms for standard and redundant robots and many sampling-based motion planners.

Also other packages for robotic simulation and programming has been explored and tested. These are for example:

- ROS (Robot Operating System) is a middleware system designed for distributed control of physical robots, and Klamp't is designed to be interoperable with it. Various ROS software packages can replicate many of the functions of Klamp't when used together (Gazebo, KDE, Rviz, MoveIt!), but this approach is difficult since these tools are not as tightly integrated as they are in Klamp't. ROS is heavy-weight, has a steep learning curve especially for non-CS students, and is also not completely cross-platform (only Ubuntu is fully supported).
- OpenRAVE (Robotics and Animation Virtual Environment) is similar to Klamp't and was developed concurrently by a similar group at CMU. OpenRAVE has more sophisticated manipulation planning functionality. Does not support planning for legged robots, but simulation is possible with some effort. Simulation models are often conflated with planning models whereas in Klamp't they are fully decoupled. OpenRAVE is no longer actively supported.
- V-REP is a robot simulation package built off of the same class of rigid body simulations as Klamp't. It has more sophisticated sensor simulation capabilities, cleaner APIs, and nicer visualizations but is typically built for mobile robots and have limited functionality for modeling, planning, and optimization. It is also not created specifically for custom software development because it has its rigid and pre-defined user interface that cannot be easily re-implemented for real industrial applications. It is also not open-source, so its use for industrial purpose can become expensive.

It is currently implemented in two different programming languages: C++ and Python.

Although most of the softwares developed by the ISS are written in C++, for the aim of the thesis, we have decided to write the software in Python, that has the important advantages to be easier in the configuration and to provide a lot of simple additional tools for simulation and visualization of the numerical results, but also for the generation of server-client connections for the implementation of experimental tests.

The thesis is organized as follows:

- Chapter 2 will introduce some basic concepts about the state of art in robot guidance systems, both from the vision and motion planning point of view. The first sub-chapter describes shortly the most important technologies in 3D object recognitions, then the most common concepts about both standard and automatic motion planning will be introduced;
- Chapter 3 will be useful to clarify most of the mathematical concepts behind a standard and redundant robotic system: differential kinematics, kinematic inversions and kinematic chains will be shortly described. In this way, the syntax of the input files, needed to model the structure of the Yumi robot, should result clearer and the reader should be familiar with most of the terms and functions presented in the last chapters;
- Chapter 4 presents three algorithms analysed in the preliminary phase of the thesis. They represent the alternatives for the core part of the developed software, responsible of the automatic planning procedure. These algorithms will be firstly described from a theoretical point of view, then applied on a simple simulation and the results will be studied and used to choose the planner for the final application;
- Chapter 5 is divided in two parts: In the first part the development of the software will be presented, first in its basic version and then in the final version, capable of successfully use all the potentialities of the Yumi robot. all the relevant function of the algorithm needed to maximize its planning capabilities will be and its performances analysed in a simulated scenario in different input conditions. In the second part the experimental set-up will be described, underlying how the software can be easily integrated in a pseudo-real industrial bin picking task.
- The final chapter will be used for some considerations about the future final integration of the software in robot guidance systems for human-robot collaboration applications involving bin picking tasks.

2 The state of art

2.1 Introduction to Vision Sensors

This part of the thesis is intended to describe which are the main important technologies in the field of 3D metrology.

Nowadays there are a lot of methodologies for the automatic acquisition of an object shape: in the case of industrial field the main used and developed family of methods is the one of active optical systems.

An optical system is a technology that produces tridimensional informations from the combination of images, so the bidimensional projections of an object. These images should be acquired through one or more cameras. Advantages of these techniques are high execution velocity and no necessity of contact with the object to be measured. These features allow the extensions of this type of techniques to many industrial sectors affected by particular or dangerous conditions and environments.

Optical systems are divided in active and passive. Passive optical systems are based on image analysis and acquisition with no modifications of the light conditions by the user. In active optimal systems instead, the object to be measured is typically radiated by some specific light patterns.

This type of optical systems can be divided in other two families: active structured light optical systems and active non-structured light optical systems. The second type is made of all those acquisition methodologies in which some light source is used to improve the acquisition conditions. Some of these methodologies are retro-illumination and coaxial led illumination.

Structured light optical systems instead, have a different logic: the light source is a fundamental part of the acquisition process. For this reason, any type of image acquisition is characterized by a well determined light source.

One of the main technologies, which is part of this family, is active laser triangulation. In this methodology all geometric properties are estimated starting from the knowledge of the laser pattern. The most diffused approach is to use a laser plane. The intersection between the laser plane and the surface of the object to be measured, generates the so called profile, that has a deformed shape depending on the shape of the object. The reflected light is acquired by a camera. In order to perform the acquisition of an entire object is needed to create a relative movement between the laser source and the object. The movement can be a rotation or a roto-translation. The final acquisition will be the union of a lot of single profiles. The final device made of a moving light source and a camera is called laser scanner and is currently the main type of sensor developed by the ISS. Main advantages of this methodology are a very good collimation of the light emission, the monocromaticity and the low power required.

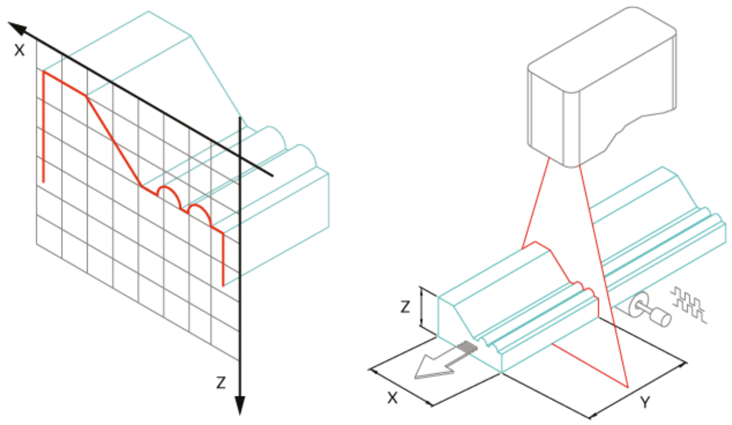


Figure 5: laser scanner

One of the main problems is that this type of devices is characterized by a high inertia, due to the weight of the laser source, the camera and of the elaboration unit. For this reason, the acquisition velocity is not so high as other methods.

Another approach is the stereoscopic interferometry. It is based on the Moirè effect: the basic idea is to project a light pattern on the object and to acquire the image trough another light pattern rotated respect to the first one. These two patterns create a grid made of light and shadows areas that can be used to perform a 3D reconstruction. This method is extremely accurate and cheap, but is computationally heavy and does not work very well with objects characterized by sudden changes of depth, that cause too big deformations of the grid, or with objects that are too far from the camera, so is not typically used in big bin picking applications but only in tray picking. Another advantage is that it does not require a relative movement between the object and the measurement device, so the acquisition is typically faster respect to another methodologies.

For its features the stereoscopic interferometry is the most suitable technology for applications with collaborative robots and will be used for the experimental implementations of the algorithms developed during this thesis.

2.2 INTRODUCTION TO MOTION PLANNING

In this part of the thesis the problem of trajectory planning will be introduced and discussed. First of all, will be presented the theoretical aspects of robotic manipulators trajectory planning. Then, will be introduced the current state of art methodologies in the field of automatic motion planning, that have of course important applications in industrial robotics, but, as it will be possible to understand, can be generalized to many problems and situations.

With the problem of "motion planning" in industrial robotics, is typically assigned the way the robot evolves from an initial posture to a final one. Motion planning is one of the essential problems in robotics and most of the success of a robot on the market depends on the quality of motion planning, that is typically demanded to the trajectory planner, one of the fundamental units of a robot controller. The result of the trajectory planning will be the reference input to the motion control system which ensures that the manipulator executes the planned trajectory. Planning consists of generating a sequence of values, attained by an interpolating function (typically a polynomial) of the desired trajectory. This interpolating function is usually called motion law and has to be designed in such a way that the transition from a planned point to the next one does not require to the actuators to exert joint forces over the saturation limits and does not excite the typical resonant modes of the structure. For this reason, planned trajectories are usually smooth trajectories.

First of all, is important to distinguish between the concepts of path and trajectory:

- Path: is a geometric entity, a line in a certain space (the cartesian space, the joint space …), to be followed by the object whose motion has to be planned. The path can be defined in different ways: it can be a point-point movement, a movement between intermediate points or a geometry path. The path is what is typically obtained from an automatic motion planning algorithm like the ones developed in this thesis.
- Trajectory: is the concept of movement over a time horizon. For this reason, it involves not only geometric quantities, but also kinematic ones, like velocities and accelerations. Trajectories are the typical results of the robot planner interpolation of the path provided by a guidance system.

Trajectories can be generated in the operational space, that is typically the cartesian space. In this way the path is a sequence of positions and orientations of the robot end-effector. The main features of this type of planning is that is easier to take into account the presence of obstacles, that the task can be described in a more natural way, but is not possible to take into account problems related to singularities and redundancies and kinematic inversion is still required in a successive phase to provide informations suitable for joints movement. The alternative is to plan directly in joint space. In this way problems of singularities and redundancies are intrinsically always monitored, online kinematic inversion is not necessary, but it's harder to manage the movement we want to obtain for the end-effector (follow a straight line, describing a circle ….).

From a theoretical point of view the joint space planning allow to manage each joint separately with a component-wise approach, without a loss of generality. For this reason, this type of planning will be discussed. Then, the common approach is to rescale the motion law of all joints on the motion time of the slower computed one.

A manipulator motion is typically assigned in the operational space in terms of trajectory parameters such as the initial and final end-effector pose, possible intermediate poses and travelling time along particular geometric paths. The values of the joint variables have to be determined first from the end-effector position and orientation specified by the user. It is then necessary to resort to an inverse kinematics algorithm if planning is done off-line, or to directly measure the above variables, if planning is done by teaching techniques.

In general, a joint space trajectory planning algorithm is required to have the following features:

- The generated trajectories should be not very demanding from a computational viewpoint;
- The joint positions and velocities should be continuous functions of time;
- Undesirable effects should be minimized.

The basic type of planning is the point-point motion, in which the manipulator has to move from an initial to a final joint configuration in a given time. The end effector path is no concern. From a mathematical point of view a polynomial function is used. Is important to notice that, the higher the degree of the polynomial, the higher is the number of conditions that can be satisfied and the smoother the trajectory will be. According to all the considerations done, the minimum function required is a cubic polynomial

$$q(t) = a + b(t - ti) + c(t - ti)^2 + d(t - ti)^3$$

Imposing boundary conditions, the typical planning result is of the type of the one depicted in figure 5.

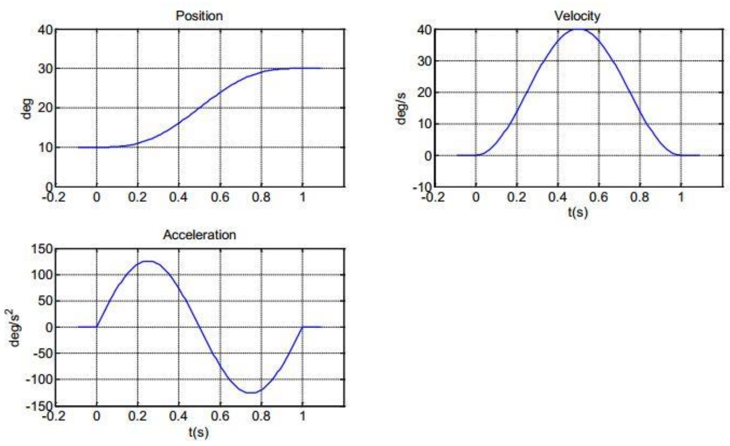


Figure 6: single joint motion planning

In industrial applications is a common practice to generate trajectories consisting of a linear profile adjusted at the beginning and at the end with parabolic bends. The resulting velocity profile has the typical trapezoidal shape (TVP) and the acceleration one is made of constant segments in the start and arrival phase.

In several applications, the path is described in terms of a number of points greater than two. For instance, even for the simple point-point motion of a pick and place task, it may be worth assigning two intermediate points between the initial and final ones, that can be representative of suitable positions for lifting off and setting down the object. For more complex applications it may be convenient to assign a sequence of points so as to guarantee better monitoring on the executed trajectories. The points are typically to be specified more densely in those segments of the path where obstacles have to be avoided or a high path curvature is expected. It should not be forgotten that the corresponding joint variables have to be computed from the operational space poses.

The problem of generating a path interpolating N points is usually accounted with a N-1 polynomial. This choice has some drawbacks:

- It is not possible to assign the initial and final velocities;
- The resulting system of equations to be solved for each constraint becomes computationally expensive;
- All the resulting coefficients are functions of all the defined points. If a point may be changed, all the polynomial coefficients must be re-computed.

The typical approach to overcome these drawbacks is to redefine the path in more low-order interpolating polynomials, taking into account the necessity to guarantee the continuity of the path over a point.

This type of interpolation can be performed with various algorithms such as polynomials, splines or lines & parabolas method.

Trajectory planning in the joint space yields unpredictable motions of the end-effector. When we want the motion to evolve along a predefined path in the operational space, it is necessary to plan the trajectory directly in this space. Trajectory planning in the operational space entails both a path planning problem and a timing law planning problem. The approach also changes in the case is important to consider only the position of the end-effector or also the orientation.

The most common types of paths required are linear paths, that are completely characterized once two points in cartesian space are given, and circular paths, that can be usually defined trough polar coordinates or trough the definition of three points in the cartesian space. These two basic types of path can be concatenated in order to obtain more elaborated paths. The concatenation is usually obtained trough a parametric parabolic path defined by a “via point”, a point that the path does not require to intersect. This behaviour is called over-fly. During the passage near a via point, the path remains always in the plane specified by the two lines intersecting in the via point. This means that the problem of planning the over-fly is planar.

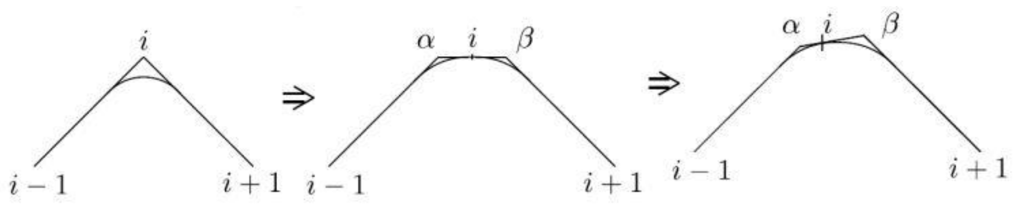


Figure 7: over-fly

Once the basic concepts of robot motion planning are introduced, is possible to start talking about automatic path planning.

As expressed by L. Larsen, from German aerospace centre, in "Automatic path planning of industrial robots comparing sampling-based and computational intelligence methods", "In times of industry 4.0 a production facility should be smart. One result of that property could be that it is easier to reconfigure plants for different products which is, in times of a high rate of variant diversity, a very important point. Nowadays in typical robot-based plants, a huge part of time from the commissioning process is needed for the programming of collision free paths. […] To speed up this process significantly, an automatic and intelligent planning system is necessary […]" [3].

Automatic path planning is required when obstacles are present and is necessary to plan motions that enable the robot to execute the assigned task without colliding with them. This means to generate a path that is collision free. The classical approach to this problem is to manually program a sequence of movement taking into account that the presence of obstacles in the working environment of the robot is known and usually fixed during the robot task execution. In this way the programmer is usually able to predict the possibility of collisions and write a sequence of commands in order to avoid that possibility.

Although the robot environment is usually known a priori in most of the industrial fields, the pose of the object to be manipulated can be random, such as in bin picking applications. In this case, a classical programming approach is useless and is necessary to find an algorithm that autonomously generate a geometric path, given the pose of the target, the kinematics of the robot and the known positions and shapes of the obstacles.

The algorithms capable to generate such results work typically at a high level of abstraction and, for this reason, they have implementations in different fields such as virtual reality or protein docking.

The classical planning problem is to find a continuous path from a starting to a final configuration without collision with the environment. The most famous challenge in this category is the "Piano Mover's problem": is required to find a path to move a piano from a starting to a final point of a room in which many objects are randomly placed.

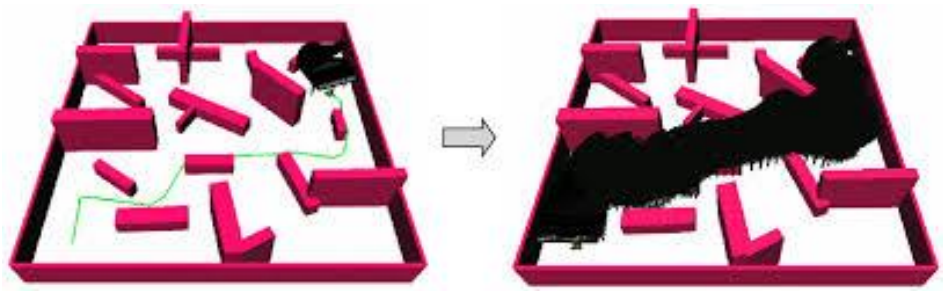


Figure 8: piano-mover problem

The main problem is to find a way of representing the position of each point of the object to move and of the obstacles; this representation is typically called "configuration" and all the feasible configurations are grouped in the "configuration space". The C-space typically is not a Euclidean space and its dimension is equal to the number of degrees of freedom of the object to move. Planning means to find a path in the C-space. In the C-space is possible to distinguish between portions of the space that are occupied by obstacles (C_{obs}) and portions useful for the path generation (C_{free}). Usually, for a small number of dofs, the path in the C-space can be found trough discrete approaches: discretization means to divide the C-space in small portions and verify if these portions are free of obstacles. This means that the algorithm tries to move from a configuration to the next one until it is totally part of C_{free}.

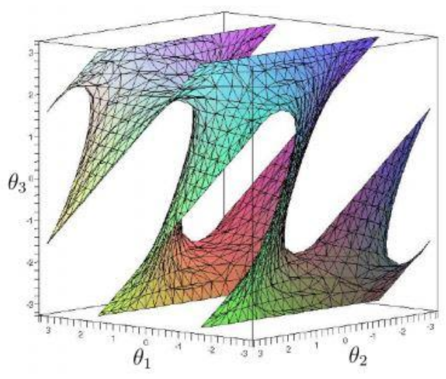


Figure 9: the C-space

If the configuration space has a high dimension the difficulty in the implementation of an analytical or discrete approach increases suddenly. Another category of algorithms is growing in the last few years to face this problem. These algorithms are called "sampling-based motion planning algorithms" and are nowadays the most used in industrial applications. The most important example is for sure the RRT (Rapidly-explored Random Trees). The basic idea is to sample the C-space, checking for a feasible, free configuration only after the sampling, instead of discretize and analyse all the C-space a priori, looking for a free path only when the entire space is known. From this basic idea, different versions of the algorithm were born and will be analysed in the thesis in order to find the most suitable for the studied application, based on planning time and percentage of success.

The basic idea behind all versions of the RRT algorithm is the following one but will be seen more in detail in the following chapters: the complete mapping net is not built. The algorithm, trough the sampling of the C-space, searches directly for a sequence of configurations capable to connect the initial and final configurations. The query is intrinsic in the sampling procedure. For this reason, at the end of the construction phase there will be only one completed path, the faster one, and the others remain uncompleted.

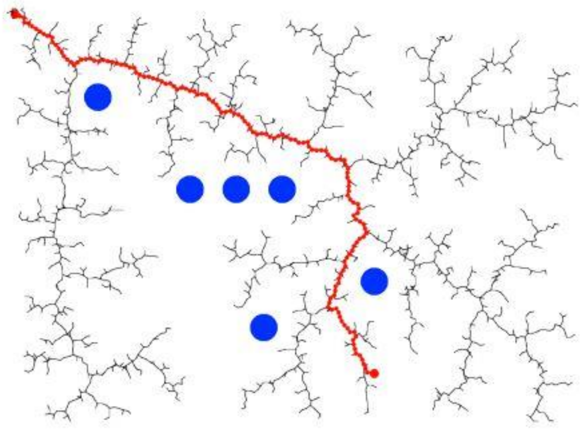


Figure 10: RRT mapping

The idea behind this thesis is to combine all these technologies, expanding motion planning features to the collaborative robotics capabilities provided by the Yumi robot. As already said in fact, motion planning algorithms just introduced are not directly studied for coordinated path planning of more then one moving entity. A smart approach for a correct movements management of Yumi must be developed, in order to help and speed up path computations, optimizing the whole procedure from a cycle-time point of view.

The result will be a smart, optimized, automatic bin picking procedure for collaborative assembling and manipulating procedures.

In this thesis, different algorithms will be tested and compared from some points of view. First of all, the planning time that is the most important parameter in order to ensure a high productivity for the typical applications of bin picking required by the companies. In fact, in most of the cases, the robot responsible of the bin picking function has to feed some other machines, responsible of one or more operations on the workpieces, that work with a predefined machining time. For this reason, is important to guarantee that, in any case, the planning time does not exceed a predefined threshold, or is preferable to start a new planning operation on the next detected workpiece. As we are going to see, the planning phase will depend on some configurable parameters and the variability on the results, for each algorithm, will be investigated.

Some other important aspects are, for example, the percentage of success as function of the position of the workpieces to grasp and the length of the computed trajectories that have an influence on the final robot actuation time.

3 ROBOT KINEMATICS

3.1 STATIC AND KINEMATIC OF INDUSTRIAL ROBOTS

This chapter will be used to introduce and clarify most of the theoretical aspects related to industrial manipulator's kinematics, starting from the basic algebra concepts, till the description of the files used to model robot kinematic chains in the present work of thesis.

As already stated, the position and orientation of a robot's end-effector can be described either in cartesian (workspace) or joint variables. Usually, these two spaces are related by a nonlinear system of equations that can be summarized by the expression below:

$$S = F(Q)$$

Where S is the vector of cartesian variables and Q is the vector of joint variables.

$$S = [\ s_1, s_2, \dots., s_n\ ]$$

$$Q = [\ q_1, q_2, \dots., q_m\ ]$$

When the relation is written in that way, so Q is given and S is the unknown, the problem is of "forward kinematics". In most of the industrial applications instead, the cartesian variables are given, because the final position of the end-effector defines the requirement of the task and the corresponding joint variables must be computed to guarantee the performing of the task. In this case is possible to talk about "inverse kinematics", that can be written in the following way:

$$Q = F^{-1}(S)$$

For open kinematic chains, that are the most diffused in the field of industrial manipulators for bin picking applications, this problem is usually more complicated respect to forward kinematics. The reason is in the difficulty in the inversion of the nonlinear relationships valid for the forward kinematics. As will be explained in this chapter, this problem is typically solved using numeric algorithms.

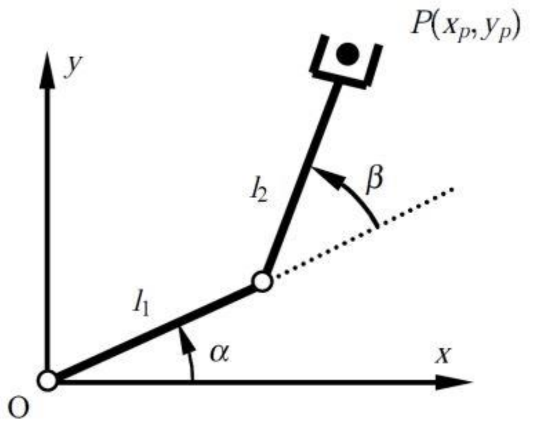


Figure 11: an open kinematic chain

The implementation of those algorithms requires the definition of the complete kinematics, so the evaluation of velocity and acceleration relationships, written both in direct and inverse form. The main advantage is that velocity and acceleration are defined by the differentiation of the position equation. This means that the following relationships will be linear and can be inverted in with less difficulties. Starting from the velocity equation, it can be defined as follows:

$$\dot{S} = \frac{\partial F}{\partial Q}\,\dot{Q} = J(Q)\,\dot{Q}$$

Where J is the jacobian matrix. It is the matrix expressing the linear relation between cartesian and joint velocities and it may be seen as the variable transmission ratio for complex, multi-degree-of-freedom mechanical systems. As is possible to understand, this relation is easier to invert, even if there is the drawback that the jacobian matrix is still position dependent; Anyway, in general, inverse kinematics, from the velocity point of view, is computationally not expensive as the position relation:

$$\dot{Q} = J^{-1}(Q)\,\dot{S}$$

Deriving the velocity equation is possible to obtain the acceleration relation for forward kinematics:

$$\ddot{S} = \dot{J}\dot{Q} + J\,\ddot{Q}$$

That can be inverted, obtaining the expression of joint accelerations:

$$\ddot{Q} = J^{-1}(\ddot{S} - \dot{J}\dot{Q})$$

Once the complete kinematics is explained, is possible to introduce the main method used for inverse kinematics of serial manipulators. It is called Newton-Rhapson Method. It is a general algorithm used to find solution of no linear systems of equations. In case of systems made of one variable, it is called Gradient Method. It consists in an iterative procedure, based on the series expansion of the kinematic relation:

$$S = F(Q_0) + \frac{\partial F}{\partial Q}(Q - Q_0)$$

$$S = F(Q_0) + J(Q - Q_0)$$

$$\Big\downarrow$$

$$Q = Q_0 + J^{-1}(S - F(Q_0))$$

These passages can be generalized at the i-th iteration, obtaining the flow chart depicted in figure 12.

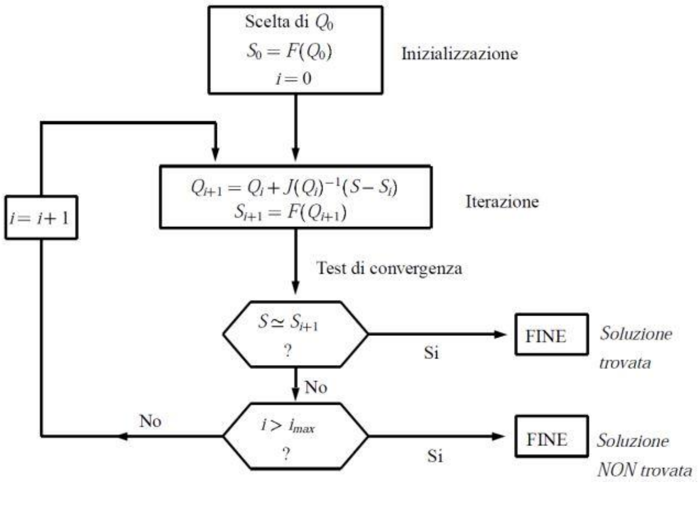


Figure 12: Newton-Rhapson algorithm

This algorithm, with a numeric approach, allows to find the joint variables, corresponding to a given cartesian pose, with a specified tolerance, after a number of iterations influenced by the initial guess of joint variables that must be performed to start the procedure.

3.2 STATICS OF THE RIGID BODY AND KINEMATIC CHAINS

Once the kinematics of a serial manipulator is known, is important to face with the problem of describing the position and orientation of each link of a robot respect to the previous one. The sum of all these informations allows to describe and model any type of manipulator, writing what is called “kinematic chain” of a robot. Having the knowledge of these concepts is of fundamental importance to manage the model of a robot from a mathematical point of view, without the necessity of strong and expensive graphical tools as, for example, CAD systems.

First of all, is important to clarify the basic mathematical representation of a rigid body in the space. The description of a kinematic chain will be simply the composition of a series of rigid body connected by joints. These joints can be revolute (the relative movement between the connected links will be a rotation) or prismatic (the relative movement will be a translation). The overall motion of the robot is achieved by the composition of elementary movements of each arm with respect to the previous one. In most of the application what is important to find is the final pose of the robot’s end effector respect to a known reference frame. This direct kinematics problem can be solved with a systematic matrix approach.

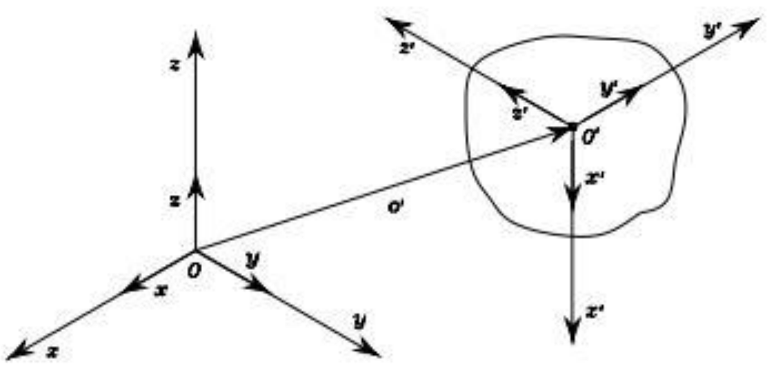


Figure 13: 3D object relative position

Considering a rigid body in a 3D space, described by its reference frame x’-y’-z’ with centre O’, its position respect to the world reference frame x-y-z is:

$$o' = [\, o'_x\,,\ o'_y,\ o'_z]$$

The description of the relative rotation between the body reference frame and the world one is more complex and requires three unitary vectors, each of them describing the orientation of a body axes respect to all the world frame:

$$\boldsymbol{x}' = x'_x\boldsymbol{x} + x'_y\boldsymbol{y} + x'_z\boldsymbol{z}$$

$$\boldsymbol{y}' = y'_x\boldsymbol{x} + y'_y\boldsymbol{y} + y'_z\boldsymbol{z}$$

$$\boldsymbol{x}' = z'_x\boldsymbol{x} + z'_y\boldsymbol{y} + z'_z\boldsymbol{z}$$

All the coefficients allowing the linear transformation from world to body reference frames are called direction cosines and can be summarized in the 3x3 rotation matrix R:

$$R = \begin{bmatrix} x'_x & y'_x & z'_x \\ x'_y & y'_y & z'_y \\ x'_z & y'_z & z'_z \end{bmatrix}$$

Each column of R corresponds to the unit vector of the orthonormal rigid body frame. This matrix, if well determined, has always the property to be orthogonal. The most important consequence is that:

$$R^{-1} = R^T$$

Rotation matrices are typically defined respect to the world frame. For this reason, is possible to define three matrices, one for each rotation respect to a fundamental axis.

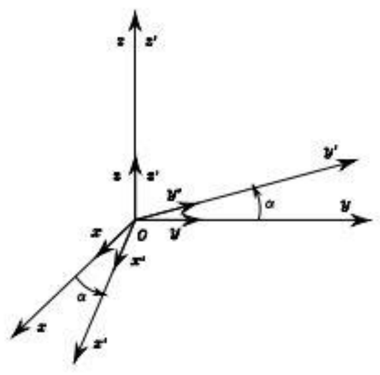


Figure 14: frame rotations

The three fundamental rotation matrices can be defined as follow:

$$Rz = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$Ry = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

$$Rx = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}$$

As is possible to see, although each matrix is composed of 9 elements, only three independent parameters are required to completely describe the orientation of a body in a 3D space. This is called minimal representation of the orientation.

An arbitrary orientation in the space can always be defined as the combination of these fundamental rotations. From a mathematical point of view, this is obtained simply multiplying these three matrices. It’s important to underline that the final orientation of a body changes according to the specific sequence of rotations chosen, or, mathematically, by the sequence of multiplication of the rotation matrices. For this reason, some standard conventions were defined. The one chosen for this work is based on ZYX Cardan angles that can be summarized as follow:

- Rotate the reference frame by an angle α about axis z;
- Rotate the current reference frame by an angle β about the new y’ axis;
- Rotate the current reference frame by an angle γ about the new x’’ axis.

The vector defining the position of a rigid body and the matrix defining its orientation can be combined in the so called Homogeneus transformation. It is usually a 4x4 matrix defined as follows:

$$T = \begin{bmatrix} R & o' \\ \mathbf{0} & \mathbf{1} \end{bmatrix}$$

Matrix T provides a compact, complete representation of the pose a rigid body in a 3D space. As for rotation matrices, also homogeneus transformation matrices can be combined, in order to describe the final pose of a body expressed as a sequence of given coordinate frames. This concept is of fundamental importance for describing the kinematic chain of an industrial manipulator.

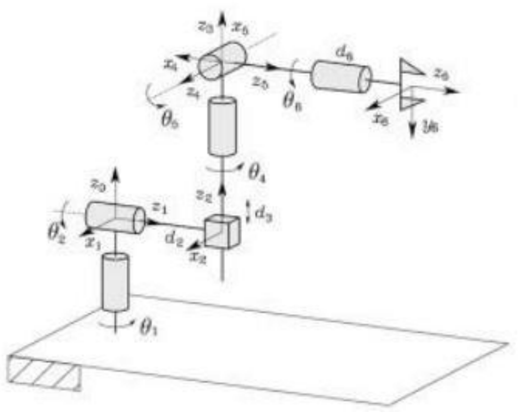


Figure 15: 3D kinematic chain

$$T_{06} = T_{01}T_{12}T_{23}\ldots T_{56}$$

As is possible to understand, although the T matrix have 16 values, it depends on only 6 independent parameters and being a rigid body in the space provided with six degrees of freedom, T is capable of describing its position exactly.

Homogeneus transformations are not the only way to describe the relative position and orientation of two rigid bodies. In the robotic field, the most important method is represented by the Denavit-Hatenberg parameters. This is a classical method, developed in the 60’s, used to formalize and standardize the way in which to choose and place the reference systems of a rigid body respect to the previous one, allowing an easy definition of an open kinematic chain. Following a method has the purpose of “automating” equation’s writing and simplifying some mathematical steps.

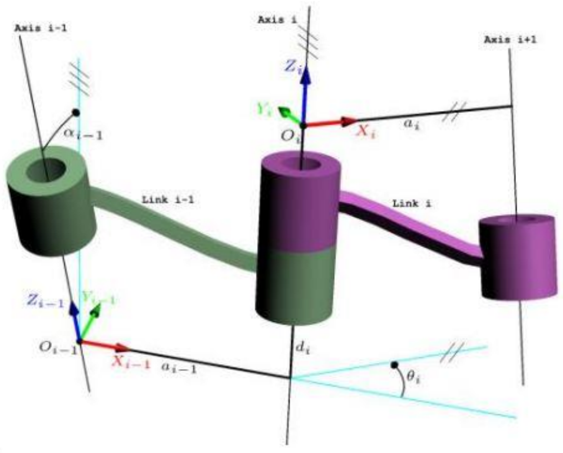


Figure 16: D-H parameters

As it’s possible to see in figure 16, in the D-H convention, axis z is always the axis of movement of the link I respect to the link i-1. Based on this convention, the four independent parameters, needed to completely describe the relative pose of the link I respect to the link i-1 are the following:

- Distance a along axis x of link i-1;
- Distance d along axis z of link i;
- Relative angle α respect to axis x of link i-1;
- Relative angle θ respect to axis z of link i.

The developed software is able to accept both types description of a kinematic chain, but, in the specific case of the implementation on the Yumi robot, homogeneus transformations have been used, simply because these informations are already provided by the ABB company.

3.3 INVERSE KINEMATICS FOR REDUNDANT MANIPULATORS

In subsection 5.1 the direct and inverse kinematics of standard industrial manipulators has been presented. In that case, all the computations are based on the assumption that the joint and cartesian spaces have the same dimension. Under this assumption, all matrices are squared and it’s easier to solve all the differential kinematics, involving velocities and accelerations, that is based on the inversion of the jacobian matrix.

In the specific case of this thesis instead, is required to face with the motion planning problem of two robotic arms, both of them redundant.

A robot is kinematically redundant if the joint space dimension is bigger of the cartesian one, or, in other words, if it has more degrees of freedom than those strictly necessary to perform a task. In fact, from a more general point of view, redundancy is a relative concept: the number of task variables may be less than the dimension of the cartesian space.

Kinematic redundancy can be exploited to:

- increase dexterity and manipulability;
- avoid obstacles;
- avoid kinematic singularities;
- minimize energy consumption;
- increase safety.

For all these reasons, the majority of collaborative robots have redundant structures, because they are studied to mimic human capabilities and/or cooperate with human beings. Human arm in fact is a seven d.o.f system, without considering the degrees of freedom of the fingers. That’s because, also if we fully constrain the hand, there is still a possibility of movement (typically the elbow rotation)

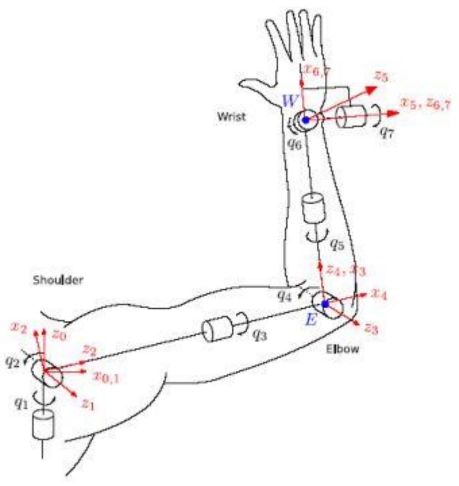


Figure 17: human arm kinematics

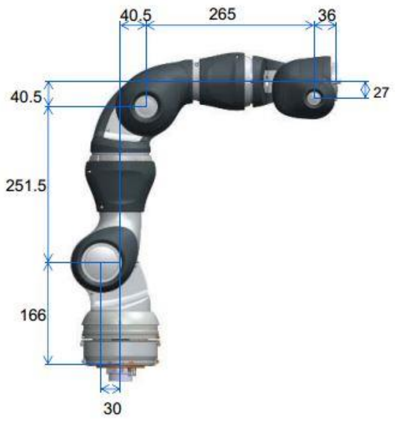


Figure 18: Yumi's arm

Problems related to the inverse kinematics of a redundant robot are the existence of infinite solutions and the problem of self-motions: the robot may have internal undesired motions in the joint space which do not affect the pose of the end-effector in the tsk space.

The problem is to select a solution of the inverse kinematic procedure. This problem is typically addressed at velocity level:

$$\dot{S} = J(Q)\dot{Q}$$

Where J is a mxn matrix, in which m is the dimension of the task space and n is the dimension of the joint space. For redundancy m<n, so the Jacobian matrix is a lower rectangular matrix.

Most of the method used to find the joint configuration are based on the pseudo-inverse matrix $J^{\dagger}$ defined as:

$$J^{\dagger} = J^T (JJ^T)^{-1}$$

The pseudo-inverse jacobian is obtained by an optimal minimization approach based on a quadratic cost function on the joint variables, modified, according to the lagrangian multipliers method, taking into account the constraint of the exact solution. In the case of redundancy there is the possibility that the jacobian matrix is not full rank. It corresponds to the condition of null velocities, so the condition in which some joint velocities do not produce task velocities. In this case it's possible to compute the pseudo-inverse numerically trough a method named singular value decomposition.

The ability of the pseudo-inverse to provide a meaningful solution in the least-square sense regardless that the initial problem is under-defined, makes it the most attractive technique in redundancy resolution. However, there are some drawbacks associated with this solution. First of all, the solution does not guarantee generations of joint motions which avoid singular configurations, or, from a mathematical point of view, configurations in which J is no longer full rank. Another problem is that the joint motions generated by this approach do not preserve the repeatability and cyclicity conditions. This means that a closed path in the cartesian space may not result in a closed path for the joint space. Non repeatability of kinematic inversion is a source of problem. The task trajectory in common industrial application is usually cyclic: that's because a cyclic joint motion is more predictable and simplifies the programming. This allows an easier simulation and verification of the tasks. The repeatability of the method has connections with the concept of holonomy of constraints. Basically, to enforce the repeatability of the cycle, some methods have been developed to enforce a holonomic constraint among the joint variables.

An alternative approach to dealing with this problem is to solve for an approximate solution. The idea is to replace the exact solution of a linear equation, with a solution which takes into account both the norm and the accuracy of the solution at the same time. This method, usually named "damped least-squares solution" has been used in different forms for redundancy resolution, but the most common formulation is the following one:

$$\left\| J\dot{Q} - \dot{S} \right\|^2 + \lambda^2 \left\| \dot{Q} \right\|^2$$

Where λ is the damping robustness factor, used to specify the relative importance of the norms of joint rates and tracking accuracy. This is equivalent to an augmented system whose solution is:

$$\dot{Q} = (J^T J + \lambda^2 I)^{-1} J^T \dot{S}$$

The practical significance of this solution is that it gives a unique solution which most closely approximates the desired task velocity among all possible joint velocities which do not exceed.

3.4 INPUT FILES

Once all the main concepts regarding robot kinematics are clarified it's possible to explain the structure of all the files make part of the input to the software. Trough these files is possible to describe both the kinematic structure of all the robots we want to control and all the static objects making part of the working environment.

The generation of this virtual world is of fundamental importance for a correct motion planning. For a correct behaviour of the real robot we must ensure that there is a perfect correspondence between real and virtual environments. Objects are, from the robot point of view, obstacles that algorithms must take into account in the configuration space in order to plan a correct, free of collision trajectory for the robot. For this reason, the geometry of all objects must be approximated in the best possible way, but also in a simple way, in order to facilitate computer calculations. The approach chosen to model all objects is to decompose their geometries into simple geometric entities called "primitives".

Now will be shortly described some concepts related to 2D and 3D primitives that will be used to approximate all geometries needed to the software.

In a 2D environment is always possible to delimit an area in which there is the object using lines and vertexes. In particular a closed shape is also called "convex" if, taking two points inside this closed shape, is also possible to connect these two points with a line and all points belonging to this line are part of the closed shape. Is important to notice that is always possible to decompose a non-convex shape into some convex shapes.

Figure 19: non-convex shape

In 3D primitive shapes are delimited not only by lines but also by planes. Also for 3D non-convex objects is always possible to reconstruct the shape trough convex decomposition.

The main advantage of the decomposition into convex shapes is that, for this kind of geometry, are already developed a lot of collision detection algorithms that use the already illustrated primitives to distinguish between free space and interdicted space.

The result of 3D decomposition of a virtual object is the .STL file, characterized by the typical triangular mesh. One important feature of this type of virtual geometric representation is that it keeps the native reference frame of the object, coming from the 3D cad software used to model it. This is of fundamental importance for the definition of the scene and of the kinematic chain of the robot

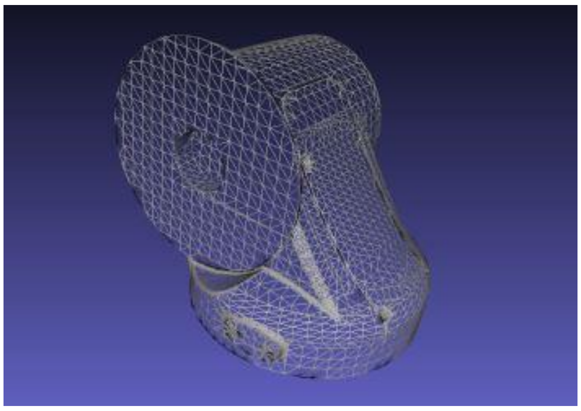


Figure 20: stl mesh

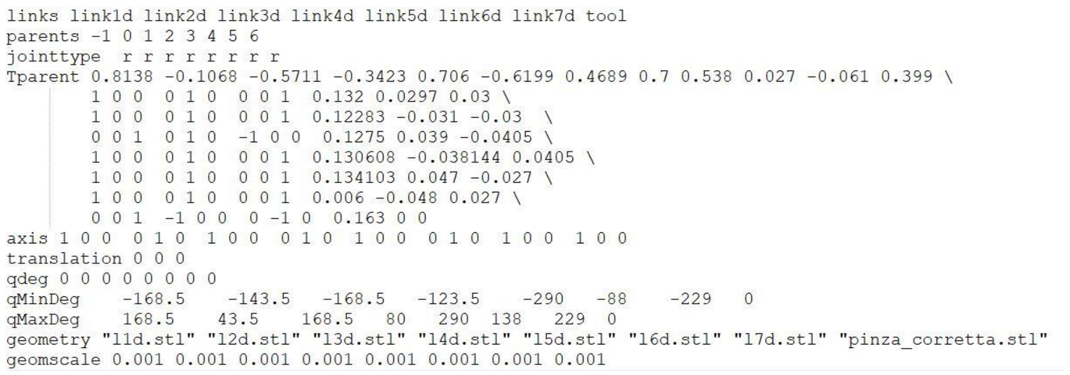
Once all virtual objects are exported as .STL files, is possible to assemble the kinematic chain of the robot and the entire scene. This is possible with other specific type of files.

.ROB file is the textual standard for the generation of virtual industrial manipulators. The typical sequence of instructions needed for the correct kinematic description of a robot is the following one:

- Links: on this line at each link is assigned a name;
- Parents: following the order of the previous line, at each link is assigned a parent link (by means of indexes). The first link will be of course associated to the ground, whose index is always -1. Is important to notice that more than a link can have the same parent allowing the possibility to assemble humanoid robots;
- Tparent: at each declared in the first line, is associated an homogeneus transformation respect to the parent link declared in line 2. Through this line is possible to define the kinematic chain of any robot;
- Axis: defines which axis (x, y, z) define the connection between link i and link i-1;
- Translation: defines the position of the first link respect to the ground reference frame. It is always (0, 0, 0) because it’s easier to define the translation in the overall scene file that will be explained later.

- Qdeg: defines the initial configuration of the robot;
- Qmin: defines the minimal values of each link;
- Qmax: defines the maximal values of each link;
- Geometry: associates at each link declared at line 1 a .STL file describing its geometry;
- Geomscale: defines the scale of the .STL files respect to the world. Useful when the robot and the other objects are exported with different unit of measurement.

The structure of the right arm of the Yumi robot is obtained trough the following .ROB file:

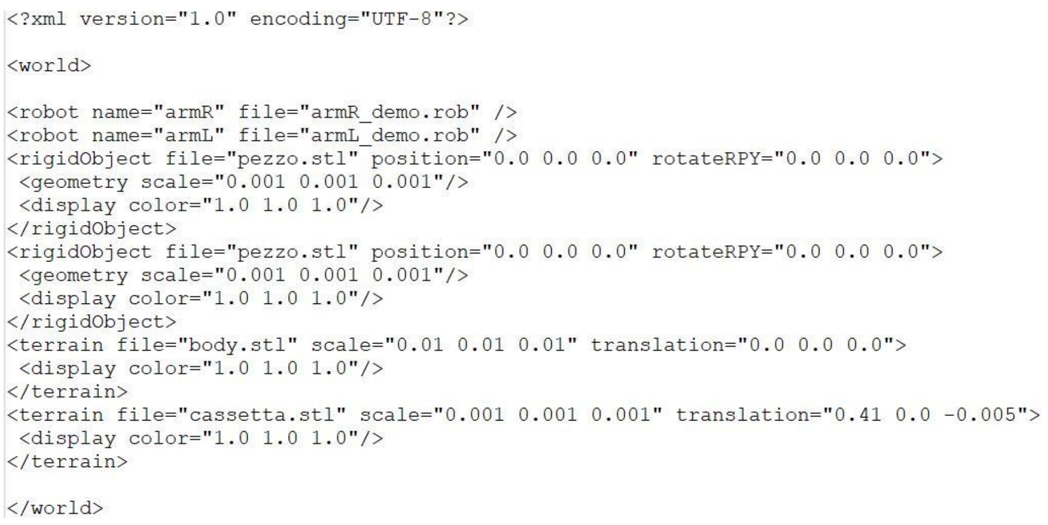


```
links link1d link2d link3d link4d link5d link6d link7d tool
parents -1 0 1 2 3 4 5 6
jointtype  r r r r r r r r
Tparent 0.8138 -0.1068 -0.5711 -0.3423 0.706 -0.6199 0.4689 0.7 0.538 0.027 -0.061 0.399 \
        1 0 0  0 1 0  0 0 1  0.132 0.0297 0.03 \
        1 0 0  0 1 0  0 0 1  0.12283 -0.031 -0.03  \
        0 0 1  0 1 0  -1 0 0  0.1275 0.039 -0.0405 \
        1 0 0  0 1 0  0 0 1  0.130608 -0.038144 0.0405 \
        1 0 0  0 1 0  0 0 1  0.134103 0.047 -0.027 \
        1 0 0  0 1 0  0 0 1  0.006 -0.048 0.027 \
        0 0 1  -1 0 0  0 -1 0  0.163 0 0
axis 1 0 0  0 1 0  1 0 0  0 1 0  1 0 0  0 1 0  1 0 0  1 0 0
translation 0 0 0
qdeg 0 0 0 0 0 0 0 0
qMinDeg    -168.5    -143.5   -168.5   -123.5    -290   -88    -229   0
qMaxDeg    168.5    43.5    168.5   80   290  138   229  0
geometry "l1d.stl" "l2d.stl" "l3d.stl" "l4d.stl" "l5d.stl" "l6d.stl" "l7d.stl" "pinza_corretta.stl"
geomscale 0.001 0.001 0.001 0.001 0.001 0.001 0.001 0.001
```

Figure 21: .ROB file

Once all robots are "assembled" is possible to create the overall scene. For this purpose, another standard file is used, which is the .XML file. It is widely used in various software for simulation purposes.

The typical structure contains robots, rigid objects and terrains, with various additional parameters. An example of an .XML file used for the development of the application, that will be illustrated in the next chapters is the following one:



```
<?xml version="1.0" encoding="UTF-8"?>

<world>

<robot name="armR" file="armR_demo.rob" />
<robot name="armL" file="armL_demo.rob" />
<rigidObject file="pezzo.stl" position="0.0 0.0 0.0" rotateRPY="0.0 0.0 0.0">
 <geometry scale="0.001 0.001 0.001"/>
 <display color="1.0 1.0 1.0"/>
</rigidObject>
<rigidObject file="pezzo.stl" position="0.0 0.0 0.0" rotateRPY="0.0 0.0 0.0">
 <geometry scale="0.001 0.001 0.001"/>
 <display color="1.0 1.0 1.0"/>
</rigidObject>
<terrain file="body.stl" scale="0.01 0.01 0.01" translation="0.0 0.0 0.0">
 <display color="1.0 1.0 1.0"/>
</terrain>
<terrain file="cassetta.stl" scale="0.001 0.001 0.001" translation="0.41 0.0 -0.005">
 <display color="1.0 1.0 1.0"/>
</terrain>

</world>
```

Figure 22: .XML file

The visualization of this file is depicted in figure 21.

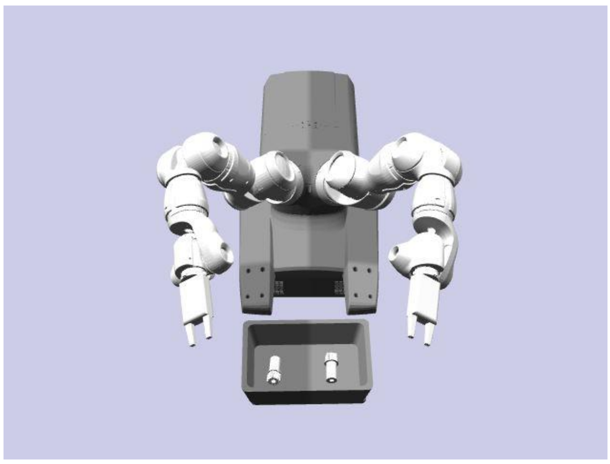


Figure 23: xml visualization

The structure of the file is quite intuitive: each object, robot or terrain is declared with a tag. Any additional information is contained between the initial and final tags. These informations are typically the absolute position and orientation of the considered entity, its color and its scale.

As it's possible to see, for the aim of this thesis we have decided to model each arm as a standalone robot and the Yumi's body is considered as a terrain entity. This choice allows to simplify the kinematic inversion for each arm and to implement in an easier way the collision detection of each arm with the body. The Klamp't library has also the possibility to manage multi-armed robots as a unique entity, but is quite complicated, the computational time increases suddenly and is substantially useless from a practical point of view in this type of applications, because .ROB files become really hard to write.

4 MOTION PLANNING ALGORITHMS

This chapter will be used to provide a deeper knowledge of the most important algorithm families for automatic motion planning, with a focus on the main theoretical aspects related to the definition of the space, the different planning strategies and collision detection approaches.

4.1 CONFIGURATION SPACE

The configuration space is one of the basic concepts needed to be introduced in any generic problem of automatic motion planning. If the configuration space (often called C-space) is well defined, most of the motion planning problems can be solved with some very common and general-purpose algorithms, regardless to the complexity of the robot to move or of the environment in which the robot should move.

Define a configuration means to generate a complete and specific representation of the position of any point of a system. This means that the C-space is made of all the possible configurations feasible for a given system. In the case of industrial manipulators, the C-space is always based on the joint space of the robot. That’s because a joint configuration fully defines the position and orientation of all points of a robot; the cartesian position of the end-effector instead, is an example of representation that does not clarify the geometric conditions of all points of the robot. This concept is easy to understand for a simple SCARA robot, as the one schematized in figure 22, but can be extended to any complex system.

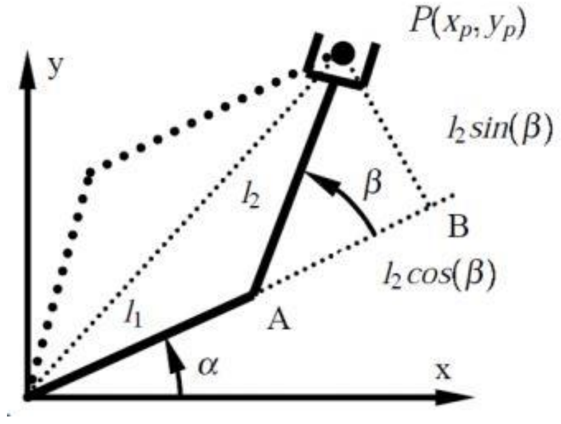


Figure 24: kinematic duality

From the considerations just explained, is possible to understand that, in robotics applications, the C-space is always equal to the joint space.

From a mathematical point of view, the C-space is a topological space. This means that it can be seen as a cloud of points with specific characteristic, such as the homeomorphism. Two topological spaces X and Y are homeomorphic if there exist a continuous function from X to Y, such as also the inverse function from Y to X is continuous. From a more practical point of view, this concept can be explained as follows: there exist an homeomorphic relation between two spaces if one can be transformed in the other one with "continuous" deformations, so without cutting or merging parts of the initial space.

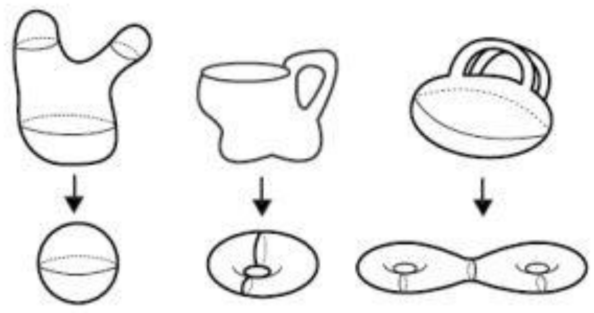


Figure 25: homeomorphic transformation

An important property of this type of spaces is that the topological space of a composed system (for example a kinematic chain) is the cartesian product of the topological spaces of each entity which is part of the system.

If the number of entities in the system increases, obtaining a graphical representation of the C-space becomes really complex, but is usually very easy from an analytical point of view: typical industrial manipulators have only rotational joints, whose C-space can be called S^1. For this reason, the C-space of each arm of the Yumi will be:

$$S^1 x S^1 x S^1 x \dots = S^7$$

We have defined a way of describing the space using a convention that is consistent with the mathematical description of a robot. The problem is that the space is usually occupied by other objects that cannot be modelled as kinematic chains, but must be considered in the C-space, because they represent obstacles for the robot motion planning. Is important to divide the space of all feasible configurations in two different parts:

- C_{free} is the portion of the C-space in which configurations of the system are not in collision with obstacles;

- C_{obs} is the portion of the feasible C-space in which configurations of the system generate collisions with the environment.

According with these definitions is possible to write:

$$C_{free} = C \setminus C_{obs}$$

Another important concept to introduce is the Manifold.

The concept of manifold is born from the necessity to manage all possible C-spaces, of any complexity, as Euclidean spaces, in order to have the possibility to apply all the mathematical concepts of the classical Euclidean math. A manifold is a topological space in which is always possible to define a limited set of points that behaves like an Euclidean space. The idea of manifold may appear quite complex, but in reality is applied in many common fields: the planisphere is the most famous example of a manifold.

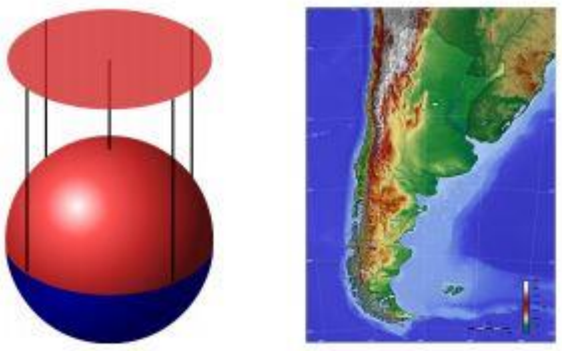


Figure 26: examples of manifold

Once a space is mathematically defined, is possible to implement a research algorithm. During this operation is of fundamental importance if a queried point in a space is attainable starting from another point. In the C-space in fact, a specific configuration becomes a point. Connect two configurations means finding trajectory from the starting to the final point in the C-space. Is important to underline that the trajectory is not a discrete sequence of other points, but a continuos function. Defining the starting point a q_{start}, the final one as q_{end} and the trajectory as τ, a C-space can be defined as path-connected if is always possible to find a trajectory τ such as $\tau(0) = q_{start}$ and $\tau(1) = q_{end}$. Obviously all manifolds treated in common motion planning problems are globally connected. The presence of obstacles can of course create local problems of connection.

4.2 GENERAL PLANNING FORMULATION

At this point, all the mathematical concepts, needed to model and integrate a system and its environment in the majority of motion planning algorithms, are known. Is now the moment to define a generic problem of automatic planning.

The formulation of a motion planning problem is composed of the following elements:

- A task space W;
- A subspace occupied by obstacles O;
- A robot model defined in W;
- A C-space defined on all feasible configurations for the robot;
- A starting configuration q_{start} and a final configuration q_{end};
- A planning algorithm.

All these elements have been already described, except for the logic behind the planning algorithms.

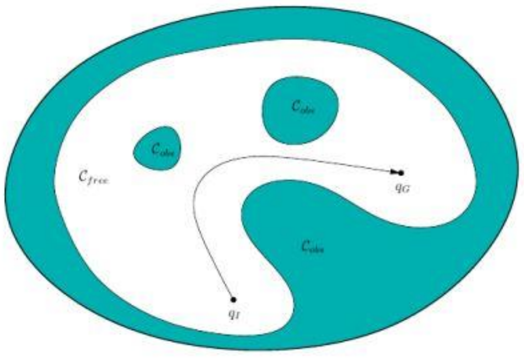


Figure 27: general planning problem

As explained in chapter 3, there are basically two main approaches to motion planning problem: the first one is the family of combinatorial motion planning, based on the completeness of its algorithms; the second and newest one is the family of sampling-based motion planning.

The first family contains algorithms that can produce only exact solutions. Results of these types of algorithms cannot be affected by errors or approximations, because they explore the C-space explicitly, finding a solution, if exist, in a finite time. Of course, they are a capable of providing a solution only for low-dimension C-spaces, because the computational times increases exponentially with the dimension of the space to be explored.

In order to solve this problem, algorithms based on a different approach were born. They are based on the generation of sampled collision-free configurations. Then, the algorithm tries to connect the sampled configurations. In this way, is possible to compose a collision-free trajectory step by step. The great difference is that C_{free} should not be completely computed, but the C-space is randomly sampled and only a successive module is responsible of verifying that the sampled configuration is part of C_{free} or C_{obs}.

Now is important to describe the main methods used to sample the C-space. First of all, is important to guarantee that, from an ideal point of view, if the algorithm should run for an infinite, it must be able to completely sample the C-space. An algorithm with this characteristic is called as "topological dense" or "topological complete". Generations of samples is typically demanded to pseudo-random algorithms. In fact, a calculator hasn't the possibility to generate real random numbers, but can only simulate a casual distribution. The most famous algorithm in this field is the Mersenne-Twister one.

The sampling of a manifold is typically represented on the so called "Voronoi diagram".

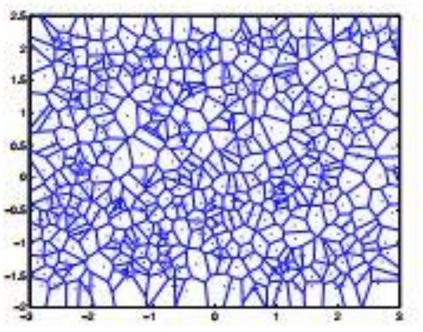


Figure 28: Voronoi diagram

During the sampling phase, also the collision checking phase is typically performed. This is the most computationally expensive operations for all motion planning algorithms. The check can be seen as a Boolean function that takes a configuration as input and gives as outputs TRUE if the configuration is in collision or FALSE if it is safe. The checking method is divided in two phases: a broad phase and a narrow one.

- Broad Phase: any object in the space is surrounded by an object bounding box (OBB). This entity is the smallest parallelepiped that completely contains the object. The algorithms in this phase simply verify that no intersections in two or more OBB are present at the considered configuration. If there is an intersection the next phase is required. This check is very easy from a mathematical point of view, but the generation of the OBBs is quite complex, so also this phase can a relevant computational time.

- Narrow Phase: This phase of the collision detection algorithm uses the geometric primitives in which any object of the space is decomposed during the upload of the scene (as explained in chapter 5). For every primitive identified, for example a vertex, a bounding box is generated and a collision checking with the overall OBB of the second object under analysis is done. If there is a collision, also the OBB of the second object starts to be decomposed around its geometric primitives. Increasing the number of primitives used, the OBBs become more and more near to the original shape of the objects and the collision checking becomes more and more refined, until a predefined threshold is exceeded. At that point the algorithm stops to iterate and the two objects are considered as not in collision.

Another fundamental check that all motion planning algorithms perform is the distance check in order to ensure a predefined level of safety for the final computed trajectory. This is often due to the possibility that virtual obstacles are only broad approximations of the objects present in the real robot environment. Is important to underline that the distance check is more restrictive respect to the collision one, and it’s usually performed before the collision checking operations. In this way in fact is possible to optimize the planning time, starting a new sampling just after having verified that for the considered configuration the distance check is not satisfied.

In order to implement distance calculation functions, called “metrics”, is important that the manifold is also a metric space, in other words, a space X in which a function g: X -> X is defined. Considering three configurations a, b, c, making part of X, g must have the following properties:

- Non-negativity: g(a, b) > 0;
- Reflexivity: g(a, b) = 0 if and only if a = b;
- Symmetry: g(a, b) = g(b, a)
- Triangular inequality: g(a, b) + g(b, c) > g(a, c).

The most important metric in this field is the L_p metric defined as:

$$g(q_0, q_1) = \sum_{i=1}^{n} (|q_1 - q_0|^p)^{\frac{1}{p}}$$

The parameter p can assume different values. In the case of Euclidean spaces, it is typically equal to 2.

Is important to underline that in robotics applications, the metric as previously defined can generates some problems, in particular when it is used to connect configurations trying to generate a trajectory. There are some particular cases in which two really near configurations from a geometric point of view, give distance values bigger respect to opposite configurations that are less different from numerical point of view. Is the case, for example, of two robots with exactly the same pose except for the wrist, rotated of 360°, and of other two robots, in which the only difference is in the joint one, rotated of 180°. From a geometric point of view, the first couple is completely overlapped, instead in the second couple, robots will result oriented in opposite directions, but from a mathematical point of view, the first couple is further than the second one.

In order to solve this problem, some weights are typically added, in order to give more importance to the first three joints of common industrial manipulators that are mainly responsible of the position in the space of robot's volumes. The resulting expression of the metric is the following:

$$g(q_0, q_1) = \sum_{i=1}^{n} (\omega_i |q_1 - q_0|^p)^{\frac{1}{p}}$$

4.3 RRT AND SBL ALGORITHMS

In this subchapter will be presented and explained the most important types of RRT algorithms that are, at the moment, the state of art in the field of sampling-based approaches to motion planning. Then a valid alternative, less known algorithm called SBL will be presented and analysed.

In the previous chapters, was already explained the main advantages of a sampling-based approach for motion planning. About this topic, is possible to introduce another important differentiation: in fact, is possible to distinguish between "single.query" algorithms and "multiple-query" algorithms. RRTs are part of the single-query sub-family. This type of approach is characterized by a unique planning phase. In multiple-query instead, the sampled space must be pre-processed iteratively until the planning is complete and it's possible to compose the final trajectory. Typically, single-query approaches are preferable when a lot of planning cycles are required by the specific application, because, from a general point of view, this type of approach should be quite faster respect to the other one. Depending on the environment to be sampled, this statement can loss its validity. In the usual tasks that the ISS must face, the C-space is relatively free of objects (the only big obstacle is usually the bin, that, however, is typically easy to be avoided for an industrial manipulator). This means that the sampling phase of the C-space will be quite faster and also a multiple-query approach may become valid from a time-saving point of view.

The general RRT procedure search for trajectory from the initial to the final configuration, trying to connect any intermediate configuration trough short, free of collisions paths. Any intermediate configuration is called "node", instead any path, generated from a node to the next one is called "branch". The reason of this nomenclature is due to the fact that increasing the number of iterations, the algorithm tends always to generate a net of nodes and branches similar to a tree.

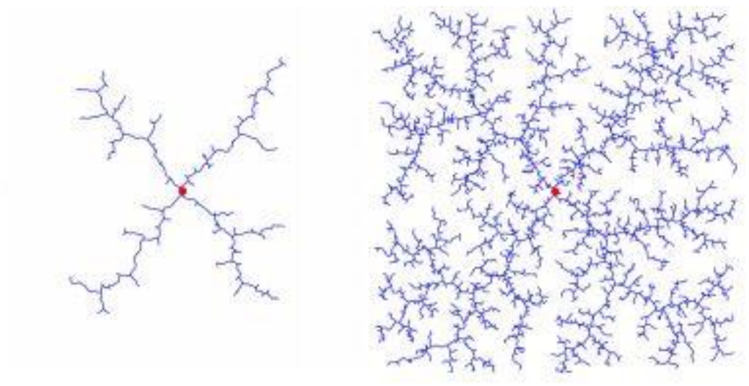


Figure 29: RRT expansion

The general sequence of fundamental operations of an RRT algorithm is the following one:

- initialization: defined V the vector of all valid configurations, q_{start} and q_{end} are added to V;
- sampling of a new configuration q_{new}
- local planning: attempt to connect q_{new} with the nearest valid node (at the first iteration it can be only q_{start} or q_{end}), with a free of collisions trajectory. If it's not possible, go back to point 2;
- if q_{new} is valid it is added to V;
- search for a valid sequence from q_{start} to q_{end}. If it does not exist go back to point 2.

Typically, the main differences in different forms of RRTs are in point 3. In fact, is possible to plan, expanding the tree only from the start configuration, but also trying from both the start and final ones (bidirectional RRT).

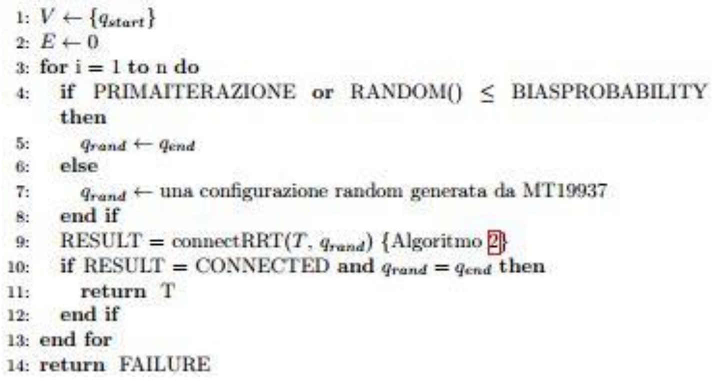
The main advantage of this algorithm is that the planning phase is only local: the planner must try to connect only two nodes that, depending on the sampling strategy, may be very near. This type of operation, that is one of the most computationally expensive, becomes in most of the applications very fast and the total planning time depends only on the total number of small paths that must be generated and checked.

Is important to notice that usually the first paths are shorter respect to the ones near to the final configuration. That's because most of the sampling algorithms generate a dense sampled space: first samplings will be very near one to each other, then the logic behind the code will tend to explore the space with more velocity.

The final goal of the planning phase is reached when it's possible to find connections between different trees able to generate a trajectory from the start to the final configuration.

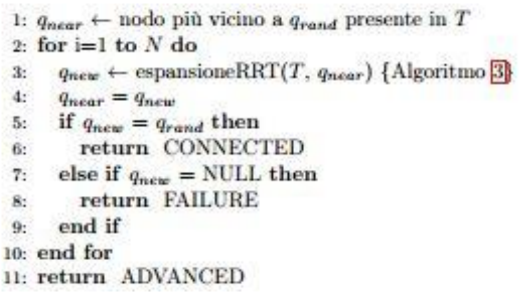
The first version of this planner is the basicRRT. It tries to generate the configurations tree starting always from the starting point. The final connection to the end configuration is attempted only when this configuration is closer to the last node of the tree respect to all others sampled possibilities.

In figure 28 and 29 are explained both the main logic and the connection procedure of the basicRRT.



```
$V \leftarrow \{q_{start}\}$
$E \leftarrow 0$
for i = 1 to n do
    if PRIMAITERAZIONE or RANDOM() $\leq$ BIASPROBABILITY then
        $q_{rand} \leftarrow q_{end}$
    else
        $q_{rand} \leftarrow$ una configurazione random generata da MT19937
    end if
    RESULT = connectRRT($T$, $q_{rand}$) {Algoritmo 2}
    if RESULT = CONNECTED and $q_{rand} = q_{end}$ then
        return T
    end if
end for
return FAILURE
```

Figure 30: RRT main

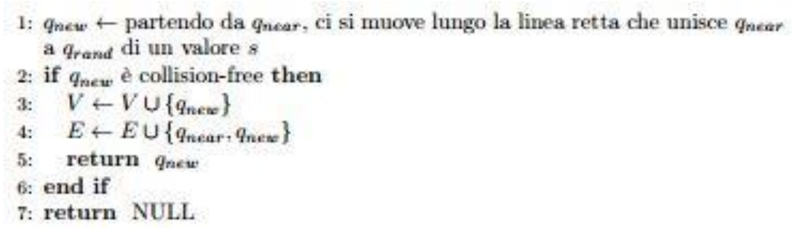


```
$q_{near} \leftarrow$ nodo più vicino a $q_{rand}$ presente in $T$
for i=1 to $N$ do
    $q_{new} \leftarrow$ espansioneRRT($T$, $q_{near}$) {Algoritmo 3}
    $q_{near} = q_{new}$
    if $q_{new} = q_{rand}$ then
        return CONNECTED
    else if $q_{new}$ = NULL then
        return FAILURE
    end if
end for
return ADVANCED
```

Figure 31: RRT expansion logic

V and E are respectively the vector of all valid configurations and the vector of all valid branches; they are both included in the vector T, that is the mathematical representation of the tree.

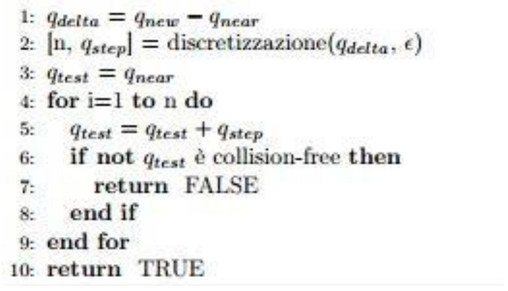
The algorithm 1 begins adding the start configuration to V. Then, the tree’s growing phase starts: a random configuration is generated; then the main algorithm requires to the algorithm 2 an attempt to connect the sampled configuration to the closer one already included in the tree T. inside the algorithm 2 is called the collision checking routine, that is a responsibility of the algorithm 3, depicted in figure 30.



```
$q_{new} \leftarrow$ partendo da $q_{near}$, ci si muove lungo la linea retta che unisce $q_{near}$
    a $q_{rand}$ di un valore $s$
if $q_{new}$ è collision-free then
    $V \leftarrow V \cup \{q_{new}\}$
    $E \leftarrow E \cup \{q_{near}, q_{new}\}$
    return $q_{new}$
end if
return NULL
```

Figure 32: RRT connection logic

The expansion check algorithm verifies that, along a line, connecting the random configuration sampled and the closer one, there is no collision with any other object in the scene. Obviously, this check is obtained discretizing the line in a finite number of intermediate configurations. This procedure is shown in algorithm in figure 31.



```
$q_{delta} = q_{new} - q_{near}$
$[n, q_{step}]$ = discretizzazione($q_{delta}, \epsilon$)
$q_{test} = q_{near}$
for i=1 to n do
    $q_{test} = q_{test} + q_{step}$
    if not $q_{test}$ è collision-free then
        return FALSE
    end if
end for
return TRUE
```

Figure 33: RRT collision checking

The basic idea is to divide the distance between the candidate configuration and the closer one in some steps and verify the collision for each of these steps.

If the check is satisfied, the random configuration becomes the new one and it's added to V. then, in algorithm 2, it becomes the new q_{near}, that will be used with the next sampled configuration for the tree expansion's attempt. These procedures are performed iteratively in algorithm 1, until the generated random configuration is equal to the final configuration. At this point the complete tree is generated and it's possible to select the feasible trajectory. Obviously for time-saving issues, iterations are limited to n: if the algorithm 1 reaches n configuration, without having found a feasible trajectory, the planning phase stops.

RRT is, actually, one of the best approaches for motion planning in high-dimensional C-spaces, but its capability to find a solution is really influenced by the complexity of the C-space. The typical example is the one depicted in figure 32: the sampling procedure is intrinsically not robust in case of the, so called, "trap obstacles".

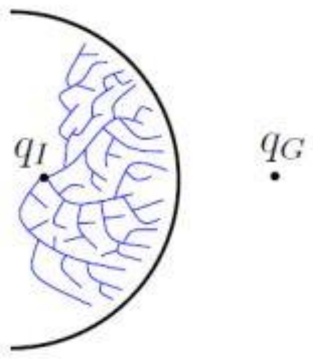
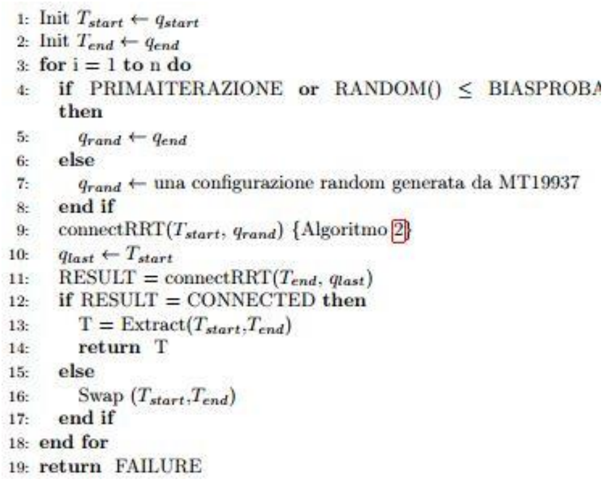


Figure 34: trap obstacle

In order to increase the performances of the RRT algorithm, the main solution found has been the BiDirectionalRRT. This type of approach starts to generate trees of configurations both from the start and the end node. This means that the propagation around an obstacle comes from two different directions and there will be high probability that, if a tree stops to propagate, the other one should find the way to keep its expansion. Typically, the expansion iterations are alternated in order to ensure a quite equal expansion rate of these two trees.



```
Init $T_{start} \leftarrow q_{start}$
Init $T_{end} \leftarrow q_{end}$
for i = 1 to n do
  if PRIMAITERAZIONE or RANDOM() $\leq$ BIASPROBA
  then
    $q_{rand} \leftarrow q_{end}$
  else
    $q_{rand} \leftarrow$ una configurazione random generata da MT19937
  end if
  connectRRT($T_{start}$, $q_{rand}$) {Algoritmo 2}
  $q_{last} \leftarrow T_{start}$
  RESULT = connectRRT($T_{end}$, $q_{last}$)
  if RESULT = CONNECTED then
    T = Extract($T_{start}$,$T_{end}$)
    return T
  else
    Swap ($T_{start}$,$T_{end}$)
  end if
end for
return FAILURE
```

Figure 35: BiRRT main

The main advantage of this algorithm is that the second tree tries to expand itself towards the last node added to the first tree. This means that the final configuration for each tree is always nearer at each iteration. In this way, in the case of simple environments like those for bin picking applications, the number of necessary nodes to find a complete trajectory is very low.

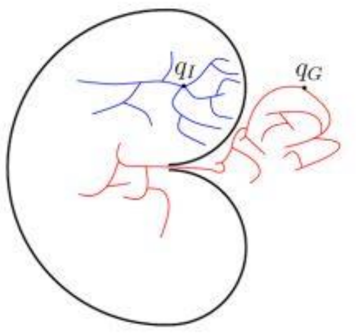


Figure 36: BiRRT behaviour

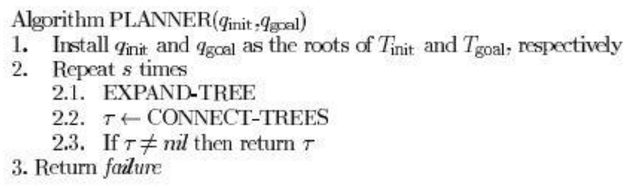
An alternative algorithm, suggested by the documentation of the Klamp't Library, is the SBL one (Single-query Bidirectional Lazy planner). SBL is a tree-based motion planner that attempts to grow two trees at once: one grows from the starting state and the other from the goal state. For this reason, the expansion philosophy is similar to the one of the BiRRT. Attempts are made to connect these trees at every step of the expansion. If they are connected, a solution is obtained. The main difference respect to RRT algorithm is that the solution path is not certain to be valid, or, in other words, free of collisions. If invalid parts of the path are found, they are removed from the tree and the exploration of the space continues from the last valid state until a solution is found. This is the "lazy" part of this algorithm. The validity of a solution is not checked in the same time of the planning phase, but only when a possible solution is guessed. This means that the SBL algorithm requires usually more iterations respect to the RRT, but each iteration is some order of magnitudes faster respect to the correspondent for the RRT. In this way, the SBL algorithm becomes competitive for bin picking applications, due to the relative simplicity of the space to explore.

SBL is based on two parameters:

- s: the maximum number of milestones that it is allowed to generate;
- ρ: the distance threshold.

Two configurations are considered close one to the other if their metric distance L is less than ρ.

As for RRT, the SBL algorithm is based on a main algorithm and on some fundamental functions. The main is the one depicted in fig. 36.

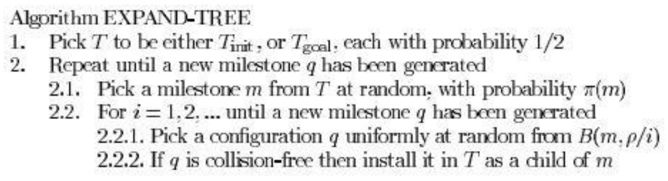


```
Algorithm PLANNER($q_{init}$, $q_{goal}$)
1.  Install $q_{init}$ and $q_{goal}$ as the roots of $T_{init}$ and $T_{goal}$, respectively
2.  Repeat $s$ times
    2.1.  EXPAND-TREE
    2.2.  $\tau \leftarrow$ CONNECT-TREES
    2.3.  If $\tau \neq$ *nil* then return $\tau$
3. Return *failure*
```

Figure 37: SBL main

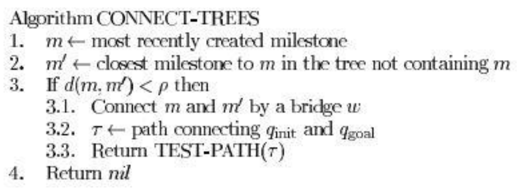
Basically, the planner builds two trees of milestones, Tinit and Tgoal, respectively rooted at q_{init} and q_{goal}. At each iteration of the step 2, EXPAND-TREE adds a milestone to one of the two trees, while CONNECT-TREES connects the two trees. If, after s iterations, a solution is not found the planner returns failure, so no collision-free path exist or the planner actually failed to find one.

The EXPAND-TREE and CONNECT-TREES functions are the ones described in fig. 37 and 38.



```
Algorithm EXPAND-TREE
1.  Pick $T$ to be either $T_{init}$, or $T_{goal}$, each with probability $1/2$
2.  Repeat until a new milestone $q$ has been generated
    2.1.  Pick a milestone $m$ from $T$ at random, with probability $\pi(m)$
    2.2.  For $i = 1, 2, ...$ until a new milestone $q$ has been generated
          2.2.1. Pick a configuration $q$ uniformly at random from $B(m, \rho/i)$
          2.2.2. If $q$ is collision-free then install it in $T$ as a child of $m$
```

Figure 38: SBL expansion logic



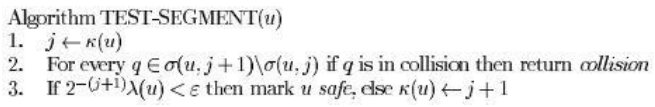
```
Algorithm CONNECT-TREES
1.  $m \leftarrow$ most recently created milestone
2.  $m' \leftarrow$ closest milestone to $m$ in the tree not containing $m$
3.  If $d(m, m') < \rho$ then
    3.1.  Connect $m$ and $m'$ by a bridge $w$
    3.2.  $\tau \leftarrow$ path connecting $q_{init}$ and $q_{goal}$
    3.3.  Return TEST-PATH($\tau$)
4.  Return *nil*
```

Figure 39: SBL connection logic

Each expansion of the roadmap consists of adding a milestone to one of the two trees. The algorithm first selects the tree T to expand, next a milestone m is picked from T with probability π. The use of the probability distribution is important to avoid over-sampled regions of the C-space. It guarantees that the distribution of sampled milestones eventually diffuses in the subset of the C-space reachable from q_{init} and q_{goal}. Finally, a collision-free configuration q is found, such that its distance from m is less than ρ. The alternation between the two trees prevents any tree from eventually growing much bigger than the other, as the advantages of bi-directional sampling would then be lost.

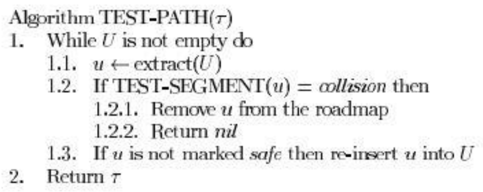
As its possible to see at step 2.2.1, at each iteration, the distance from m at which the new milestone is selected decreases of the factor i, in order to increase the probability of finding it. The third algorithm, simply select in the other tree the nearer milestone to m and tries to connect them, if their distance is less than ρ. If it's possible, the two trees are connected and the SBL will try to verify if the generated path between the initial and final configuration is collision-free.

This verification is responsibility of the fourth part of the algorithm, depicted in fig. 39 and 40.



```
Algorithm TEST-SEGMENT($u$)
1.  $j \leftarrow \kappa(u)$
2.  For every $q \in \sigma(u, j+1)\backslash\sigma(u, j)$ if $q$ is in collision then return *collision*
3.  If $2^{-(j+1)}\lambda(u) < \varepsilon$ then mark $u$ *safe*, else $\kappa(u) \leftarrow j+1$
```

Figure 40: SBL segment check



```
Algorithm TEST-PATH($\tau$)
1.  While $U$ is not empty do
    1.1.  $u \leftarrow$ extract($U$)
    1.2.  If TEST-SEGMENT($u$) = *collision* then
          1.2.1.  Remove $u$ from the roadmap
          1.2.2.  Return *nil*
    1.3.  If $u$ is not marked *safe* then re-insert $u$ into $U$
2.  Return $\tau$
```

Figure 41: SBL path check

In these two functions some new notations are present. The segments connecting two milestones are indicated with the letter u. At each segment is associated a collision-checker index, k(u). This index takes an integer values indicating the resolution at which u has already been tested to be collision-free. If k(u) = 0, only the two endpoints of the segment u have been tested. If k(u) = 1, then the two endpoints and the middle point of the segment have been verified. More in general the function TEST-SEGMENT states that, defining λ(u) as the length of u, if:

$$2^{-k(u)}\lambda(u) < \varepsilon$$

Then the segment is safe and can be included in the overall path. The function TEST-PATH is simply a managing function that removes from the trees the segments that don't pass the verification proposed in function 3.

These three path planning algorithms will be analysed in the next sub-section from a numerical point of view, with a practical example of path planning for a standard industrial manipulator, miming a bin picking application.

4.4 CHOICE OF THE PLANNER

In this section a preliminary and simplified version of the bin picking software has been used to choose the algorithm that will be implemented and used for the final application on the Yumi robot. The three algorithms will be compared from different point of views and the most performing for the specific task will be chosen. The model of the robot used for the test is the Staubli TX2-90, one of the most used medium-size manipulators in the industrial field.

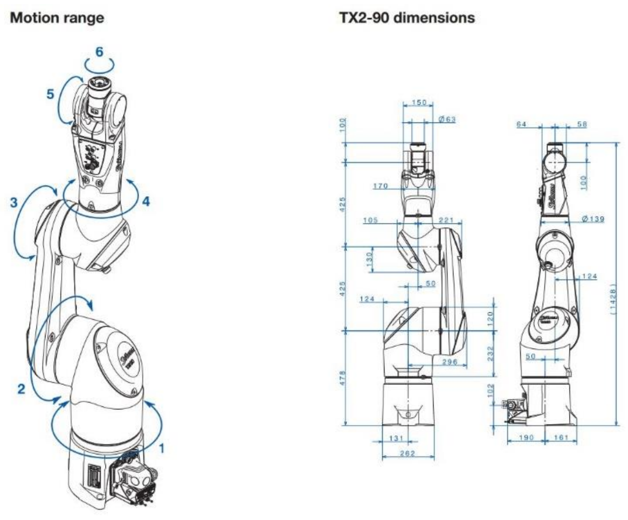
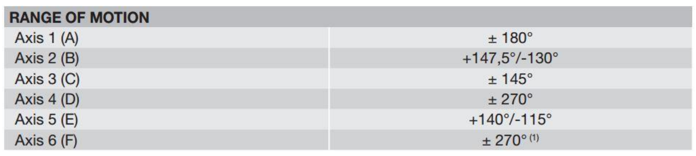


Figure 42: TX2 90 layout



RANGE OF MOTION	
Axis 1 (A)	± 180°
Axis 2 (B)	+147,5°/-130°
Axis 3 (C)	± 145°
Axis 4 (D)	± 270°
Axis 5 (E)	+140°/-115°
Axis 6 (F)	± 270° (1)

Figure 43: TX2 90 kinematic limits

Trough the images provided in figure 39, is possible to write the .ROB file, responsible of the kinematic representation of the robot. The virtual model is composed of the .STL files of each link, that are free to download from the website of the robot manufacturer. These files were properly re-meshed, trough a proper, open-source tool called Meshlab, in order to preserve their boundary region, necessary for a correct collision checking, but reducing the file's weight, in order to ensure a faster upload of the scene and visualization of the movements. For this reason, the virtual representation of the robot will appear quite transparent: this is the result of a loss of useless details caused by a broad re-meshing of the geometries. Finally, the robot was equipped by an industrial pneumatic tool, commonly used in industrial tasks. This type of device ensures less precision and gripping force respect to a mechanical gripper but allows to define many more grip points on the workpiece. Another good point is due to its shape that increases the possibility of the robot to reach also the deepest angles of the bin.

Ones the virtual model of the robot is done, it’s possible to generate the overall scene, that, in this simplified test is composed only of the robot and of a simple representation of an industrial bin of dimension 400x600x350 mm.

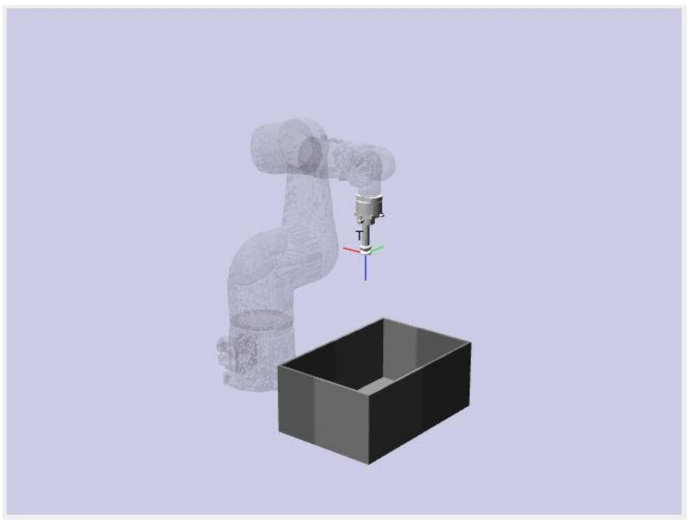


Figure 44: configuration of the test

As it’s possible to see the reference frame for the TCP of the robot has the z-axis aligned with the main direction of the tool and oriented in the opposite direction respect to the robot wrist. This is an important convention that must be always respected in the following examples and applications. That’s because in the virtual environment, in which the explicit dynamics is not included, the workpiece is reached when the TCP frame and the workpiece frame are coincident. Of course, as will be explained in the following chapter, the workpiece frame may not be the principal one, usually posed in the piece’s centre of mass, for the obvious reason that a tool like the one used in this example will be not able to reach it without a compenetration of the tool with the object. Another important aspect to underline, is that if the grip frame of the object has the z-axis oriented upward, the correspondent object will be impossible to grasp because the TCP frame cannot never be coincident with the one of the object, due to the obvious kinematic limitations of the robot. As we are going to see in the next chapter, this problem is quite always avoided thanks to the intrinsic logic behind the sensor conversion algorithm and thanks to some mathematical tricks introduced in the application developed for this thesis.

In this example no objects are present in order to speed up the test. The target that the robot should reach will be simply represented by a target frame randomly posed inside the bin.

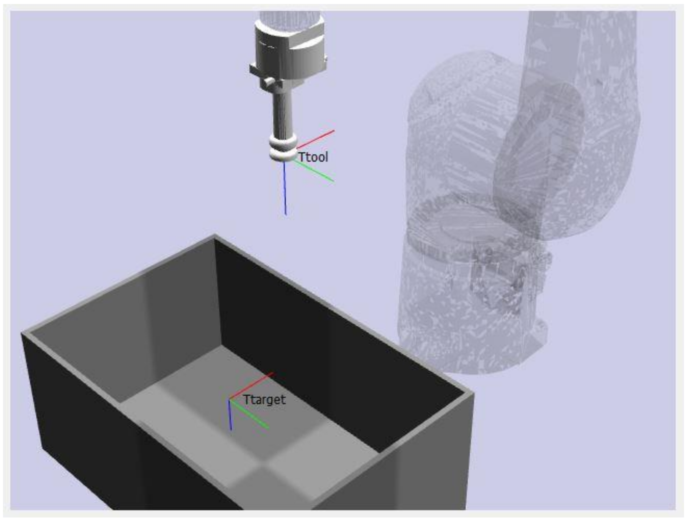


Figure 45: positioning of the target

Of course, in order to represent a realistic planning case, all the targets that will be generated, are going to have the z-axis oriented downward with a +/- 20° around the y-axis. The angle of rotation around the z-axis will be random. These two decisions allow to evaluate the possibilities of the algorithms to plan for the typical position of objects in a bin picking application.

The code that has been developed, generates randomly inside the bin 200 targets, all posed with the limitations just explained. The user is asked to choose the type of algorithm used to plan for these targets, between three possibilities: RRT, BiRRT, SBL. The user has also to choose the maximum number of iterations that the planner can perform for each target. The idea is to evaluate the potentiality of these three planners, already provided by the Klampt't library, looking at four fundamental parameters:

- percentage of planning success;
- mean required planning time;
- mean planned distance;
- mean planning number of iterations.

The important aspect will be how these performances change, varying the max number of iterations allowed. What we are looking for, is the planner algorithm that, from a statistical point of view, ensure a good percentage of success, with the minimum number of iterations required, minimizing the planned distance (that may be traduced in faster trajectories for the robot) and the planning time. The planner that will produce the best responses will be used for the final application on the collaborative robot subject of the thesis.

Is important to underline that the case proposed for this test (reaching a point inside a relatively big bin) is really easy for these types of planning algorithms. These codes are typically used for more complex planning problems. For this reason, is probable that each algorithm does not face with particular difficulties and each of them will have comparable performances. There aren't specific obstacles (such as "trap obstacles") capable of reducing planning potentialities of an algorithm respect to the others.

4.4.1 RRT planner

What is expected from the theoretical introduction already done in the previous pages, is that this algorithm should have the worst performances. As already said, the relative simplicity of the environment should not create falls of the percentage of success, but for sure the planning time should be statistically higher respect to the BiRRT algorithm.

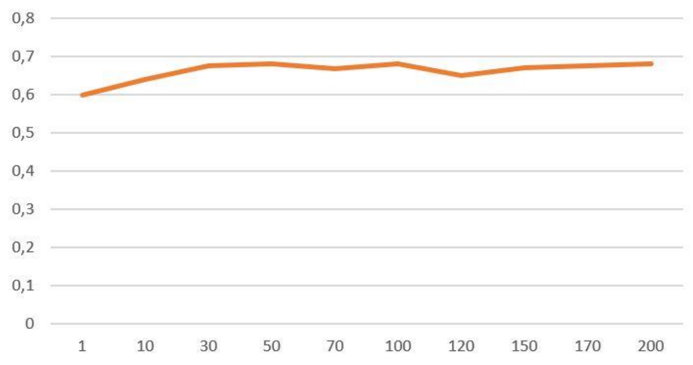


Figure 46: RRT percentage of success

As it's possible to see, the percentage of success of the RRT algorithm is not influenced by the number of maximum iterations allowed. Is important to underline how the planner tends to successfully solve always about the 65 - 68% of the planning problems proposed. This means that, how it was expected from the theoretical presentation of the planner, its solving capabilities, for relative free environments, are low-dependent to the number of iterations conceded. The next graph confirms this aspect: also conceding more possibilities to iterate a solution, the planner tends to find a trajectory, when it's possible, always with about the same number of iterations (1 – 2.2).

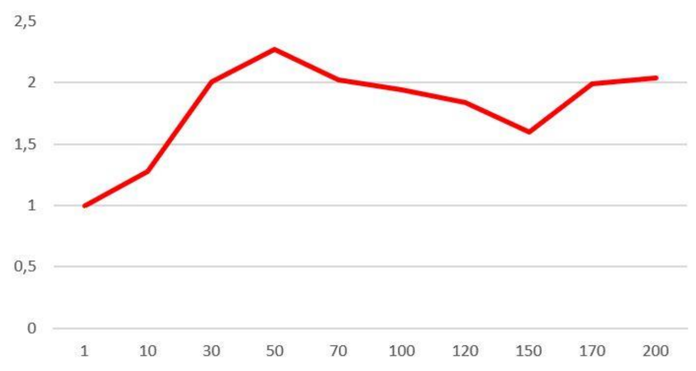


Figure 47: RRT mean number of iterations

As a direct consequence, the time required for a correct planning is quite constant. This is trivial considering the relative simplicity of the environment.

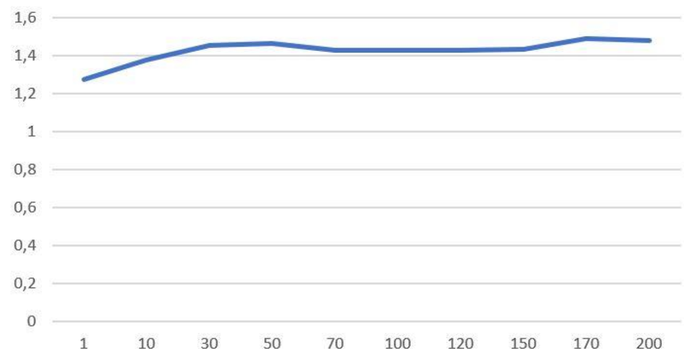


Figure 48: RRT mean planning time [s]

The computational time is always assessed around a value of 1,45 - 1,5 seconds: this will be the reference value for the planning algorithms provided by the Klamp't library. At this time in fact, is useless to compare these results with the ones obtained with the ISS softwares: these algorithms are part of a different library, compiled in a different programming language and running on more performing machines.

The final aspect to consider is the mean length of the trajectories planned by the algorithm. Also in this case, the capability of the algorithm to converge in a limited number of iterations, ensure a quite constant result respect to the number of iterations. As it's possible to see, the mean length of the computed trajectories is always about 0,76 meters.

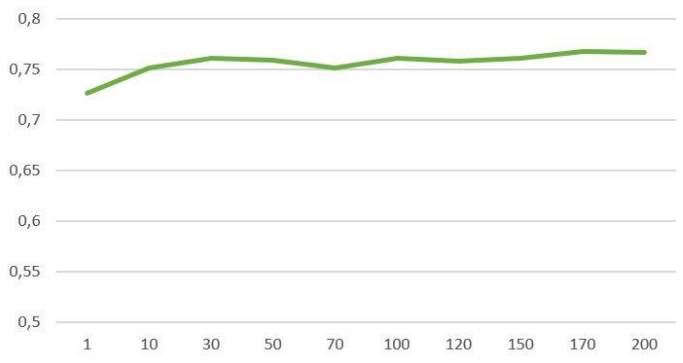


Figure 49: RRT mean path length [m]

4.4.2 BiRRT planner

The second algorithm provided by the library that we have chosen to evaluate is the BiRRT one. It has a structure similar to the RRT one, but improved capabilities due to the possibility of exploring the C-space along two independent trees. What is important to understand is if these capability produces an effective increase of performances in the case of an industrial bin picking task.

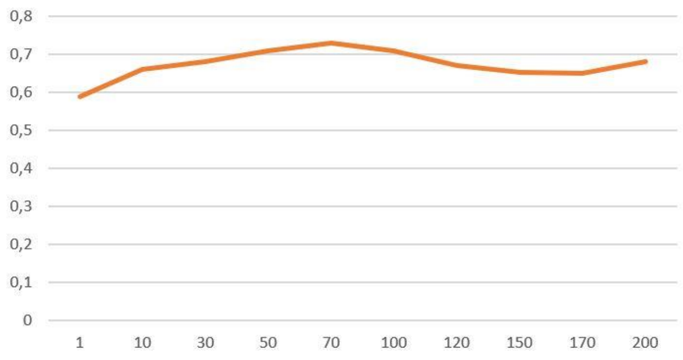


Figure 50: BiRRT percentage of success

The percentage of success of the BiRRT algorithm is, as expected, a bit higher respect to the RRT one, assessing around a value of 0,67 - 0,72. Also in this case this result is insensible to the maximum number of iterations allowed. That’s because also in this case the algorithm is always capable of finding a solution in very few iterations for the case proposed. The mean time of iterations required is reported in the next figure and it’s about 1 – 2,8.

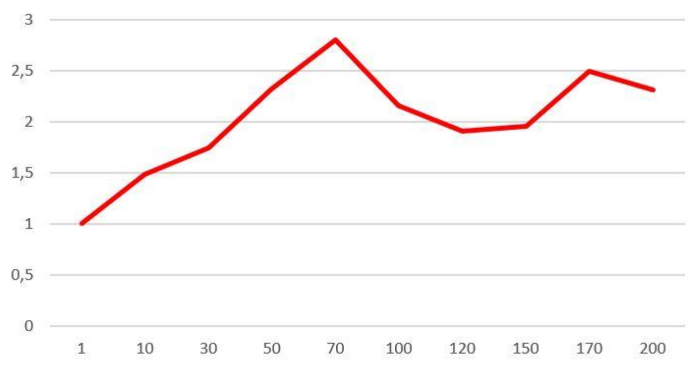


Figure 51: BiRRT mean number of iterations

One of the main interesting point in this case is the mean planning time. The structure of the BiRRT algorithm should provide a faster computational time.

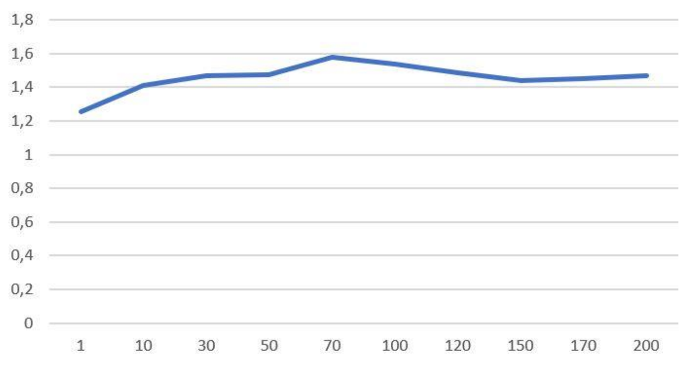


Figure 52: BiRRT mean planning time [s]

The graph instead shows a mean planning time totally comparable to the one of the RRT algorithm. This is a direct consequence of the simplicity of the application: The particular strategy of the BiRRT algorithm loses its utility in simple bin picking application without any kind of trap obstacle.

Also from the mean path length point of view, the BiRRT algorithm provides results comparable to the one of the basic version. At this point of the analysis the BiRRT algorithm is still preferable for its superior percentage of success.

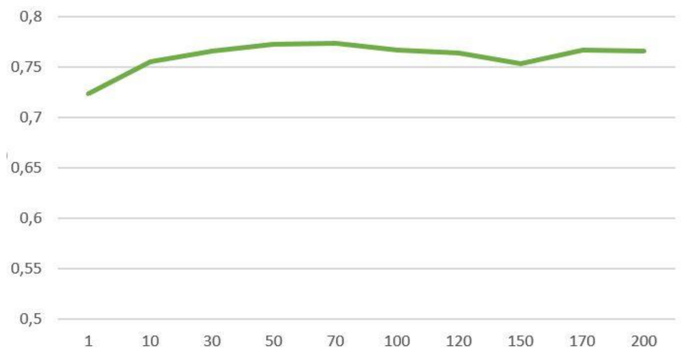


Figure 53: BiRRT mean path length [m]

4.4.3 SBL planner

This SBL algorithm has a quite different structure. It does not perform collision checking in between the tree expansion along the configuration space. For this reason, some differences are expected.

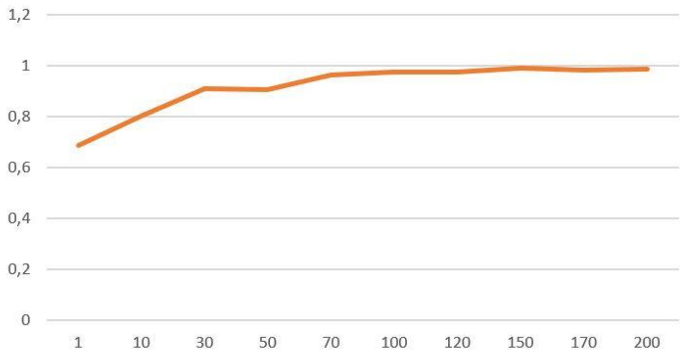


Figure 54: SBL percentage of success

The first important result is the percentage of success. It is higher respect to the previous ones for every value of iterations allowed. Another important difference is the tendence of the percentage of success of growing, increasing the maximum number of iterations conceded. After a certain value, the percentage of success tends to remain constant around the value of 98%.

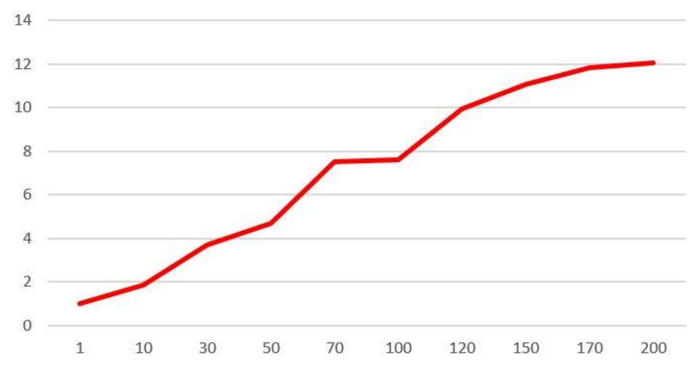


Figure 55: SBL mean number of iterations

This aspect is confirmed by the mean number of iterations graph. As it's possible to understand, in this case the overall planning procedure has a strong dependency on the number of iterations conceded. In particular, the SBL algorithm tends to successfully plan quite all the targets when has the possibility to perform at least 8 iterations, has its possible to see in the next graph.

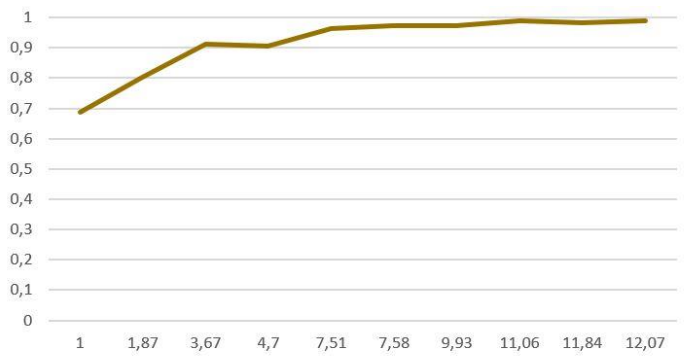


Figure 56: percentage of success vs number of iterations

This dependency is a direct consequence of the structure of the algorithm, that, due to the not-implicit collision checking, has to repeat the tree-expanding procedure several times in order to find the solution. However, is important to underline how, for the specific application, is never required a real big number of iterations. This is a prove of the goodness of the Klamp't library in optimizing this type of algorithm.

From the point of view of the mean planning time and of mean path length the SBL algorithm provides worse results respect to the previous ones.

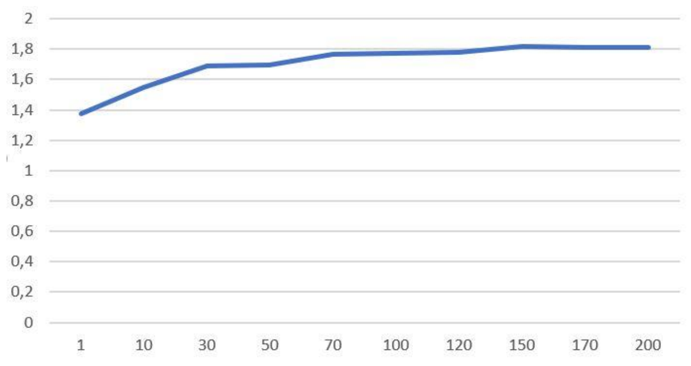


Figure 57: SBL mean planning time [s]

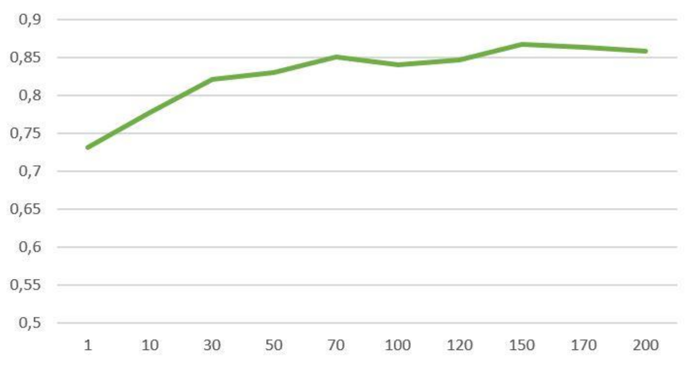


Figure 58: SBL mean path length [m]

The mean planning time is comparable to the RRT ones for low number of iterations conceded, then becomes quite constant around the value of 1,8 seconds. The mean path length is quite bigger respect to the RRT and BiRRT ones, assessing around a value of 0,85 meters.

4.4.4 Overall evaluation

In order to clarify the obtained results, two graphs showing a comparison in performances between the studied algorithms are depicted in the next figures. The orange colour is used for the SBL algorithm, the blue one for the BiRRT and the green one for the RRT.

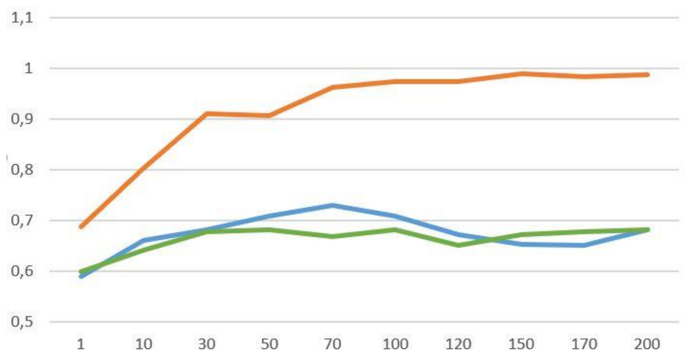


Figure 59: percentage of success comparison

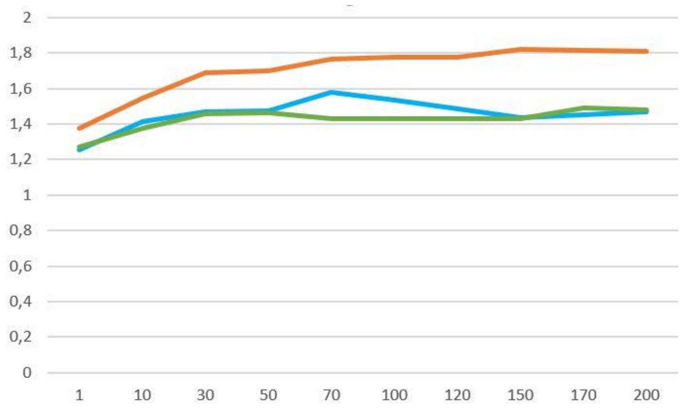


Figure 60: mean planning time comparison

As it's possible to see, although the SBL algorithm requires higher computational times, for an appropriate number of iterations conceded is possible to statistically complete quite always a standard planning procedure in a bin picking environment. Another important aspect to consider regards the computational time in case of fail. From the numerical simulation is possible to see how, for all the analysed planners, the time required to finish all the iterations, in the case it's not possible to successfully plan for a specific target, is always less of the time required for a successful plan.

For these reasons, regardless to the higher computational time (about 25 – 30 % more respect to the RRT and BiRRT algorithms) and length of the generated trajectory (about 10 – 15% more respect to the RRT and BiRRT ones), the SBL algorithm has been chosen as the planner for the final application on the Yumi Robot with a number of conceded iterations equal to 100.

5 THE DEVELOPED SOFTWARE

5.1 STRUCTURE OF THE APPLICATION

Once the main planner has been chosen, is possible to start describing the overall architecture of the software developed for the automated bin picking application with the Yumi Robot.

As already introduced, from a hardware point of view, the application is composed of the following objects:

- The Robot;
- The 3Dcps Fast scanner;
- A personal computer;
- A small bin that will be filled with some objects.

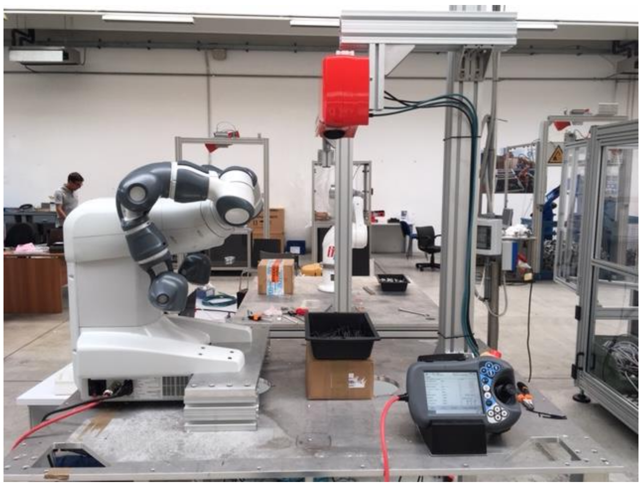


Figure 61: layout of the test case

Is important to underline that all codes, except for the communication protocols have developed in the simulation phase, in which all the computed trajectories were animated in a simulated environment, the common called “digital twin” of the real application. Disactivating the visualization of the results and adding the communication protocol, the following codes have become part of the software architecture for the experimental demo.

The overall demo works in “manual mode”. This means that the communication between the scanner and the robot is not fully automatic as in usual industrial applications but is governed by some commands provided by the user. This choice is due to the impossibility of introducing the developed code in the overall software architecture of the scanner, without consistent modifications that weren’t introducible in short times.

The communication is managed by the user, from his personal computer. The PC is connected via ethernet both with the sensor and the robot. The application requires the following manual and automatic passages:

- The user, from its personal computer, starts the scanning phase of the bin trough a software specifically developed for interacting with the 3Dcps; The scanner generates a points cloud from the scan. The scan is saved on the elaboration unit of the scanner;
- The user has the possibility to access to the memory of the scanner via ethernet connection (trough, for example, Teamviewer) and provides the points cloud to the code responsible of the 3D pattern matching and pose estimation. This small code runs on the PC of the user, is written in Halcon and its structure will be described in the following subsections;
- Once the pose of the recognized object is computed, it is saved on a .txt file. The user starts on his PC the main algorithm, written in Python, that is responsible of the planning phase. This code reads the .txt file and starts to plan a trajectory for the robot. This is the most important and time expensive phase;
- Once the trajectory is computed, the software opens a communication protocol with the robot that is already waiting for a signal. On the elaboration unit of the robot, is already uploaded a protocol that elaborates the received trajectory and provides it to the arms; this protocol is written in Rapid, the programming language commercialized by ABB. When this phase is finished, the task starts.

5.1.1 The scanning phase:

The 3Dcps Fast is the device, provided by the ISS company, for the acquisition of the scene and the proper generation of the points cloud. For a correct use, it must be registered, this means that is necessary to reconstruct its pose respect to the robot frame, knowing the position of a reference object respect to the sensor and to the robot.

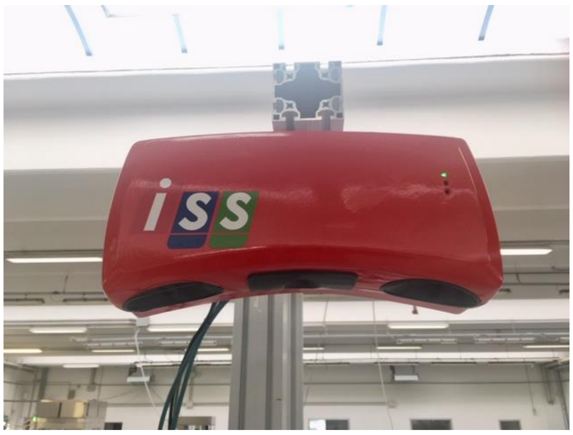


Figure 62: 3Dcps fast

In its industrial applications it is a complete robot guidance system able to automatically realize all the passages described above, but for the aim of this thesis it is only manually used as an acquisition device.

The object chosen for the tests is the one depicted in the following figure.

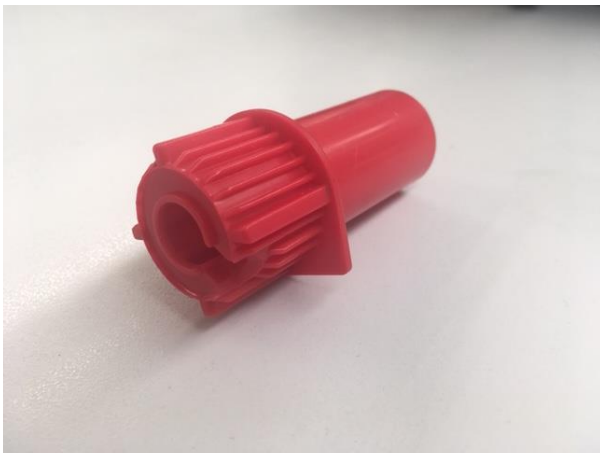


Figure 63: the workpiece

It is a simple component, made of plastic, characterized by a quite cilindrical shape but with some details that avoid the full axial symmetricity. These aspects have the following consequences: this object is quite simple to grasp, or, in other words, is quite simple to define different grasp points on its surfaces; due to the non-symmetricity, it's always possible to identify the full pose of the object, in order to test planning capabilities of the software. The pose of a standard "perfect" cilindrical object is insensible to the rotation around the main axis. This facilitates the kinematic inversion and the planning phase because it's always possible to reconstruct a frame aligned with the tool one. In this case, the potentialities of the software are not fully tested and, for this reason, this type of object has been chosen.

The result of the scanning phase on two objects placed inside the bin is the following:

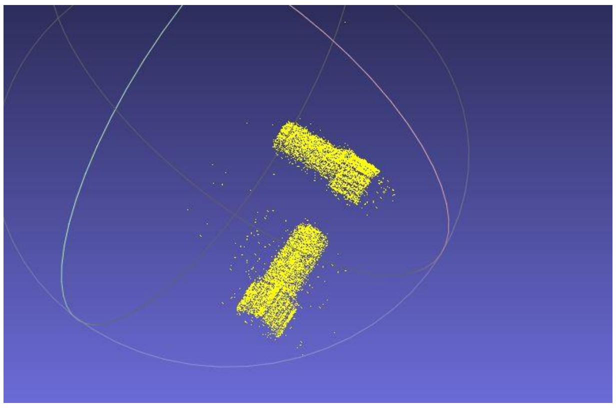


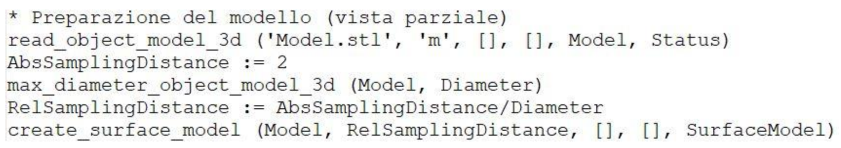
Figure 64: points cloud

As its possible to see, the scanner is able to reconstruct only the visible part of the objects but this will be sufficient to estimate their pose in the workspace.

5.1.2 The pattern matching and pose computation

The simplified procedure of pose estimation is written in Halcon. This is one of the most diffused software for machine vision applications. It is a quite powerful tool for bin picking applications, in fact also the industrial version, commercialized by the ISS, of this part of the software is written in Halcon.

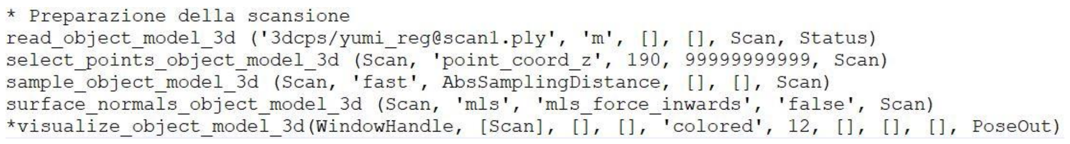
The code is composed of three main parts. In the first one the virtual model of the object is uploaded and processed. This model is the reference used for pattern matching with the points cloud and is simply a cad representation of the object used for the tests.



```
* Preparazione del modello (vista parziale)
read_object_model_3d ('Model.stl', 'm', [], [], Model, Status)
AbsSamplingDistance := 2
max_diameter_object_model_3d (Model, Diameter)
RelSamplingDistance := AbsSamplingDistance/Diameter
create_surface_model (Model, RelSamplingDistance, [], [], SurfaceModel)
```

Figure 65: Halcon script part 1

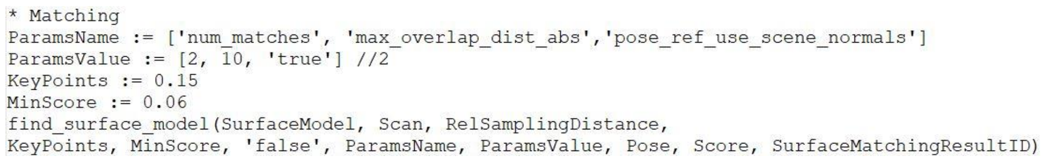
The second part of the code uploads and prepares the points cloud. In particular most of the useless points are removed, in order to simplify and speed up the computations.



```
* Preparazione della scansione
read_object_model_3d ('3dcps/yumi_reg@scan1.ply', 'm', [], [], Scan, Status)
select_points_object_model_3d (Scan, 'point_coord_z', 190, 99999999999, Scan)
sample_object_model_3d (Scan, 'fast', AbsSamplingDistance, [], [], Scan)
surface_normals_object_model_3d (Scan, 'mls', 'mls_force_inwards', 'false', Scan)
*visualize_object_model_3d(WindowHandle, [Scan], [], [], 'colored', 12, [], [], [], PoseOut)
```

Figure 66: Halcon script part 2

Once both the model and the point clouds are ready is possible to start the pattern matching

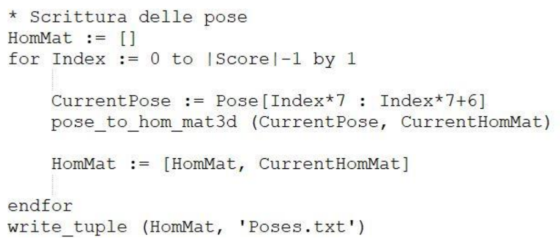


```
* Matching
ParamsName := ['num_matches', 'max_overlap_dist_abs','pose_ref_use_scene_normals']
ParamsValue := [2, 10, 'true'] //2
KeyPoints := 0.15
MinScore := 0.06
find_surface_model(SurfaceModel, Scan, RelSamplingDistance,
KeyPoints, MinScore, 'false', ParamsName, ParamsValue, Pose, Score, SurfaceMatchingResultID)
```

Figure 67: Halcon script part 3

This procedure depends on some parameters, such as MinScore that allows to obtain more precise results. Typically increasing these parameters causes also an important increase of the computation time. For this reason, and due to the fact that the application does not require a very high level of precision, typically the value of MinScore is very low, between 0,05 and 0,1.

After the pattern matching, the pose is derived and written on a .txt file.



```
* Scrittura delle pose
HomMat := []
for Index := 0 to |Score|-1 by 1

    CurrentPose := Pose[Index*7 : Index*7+6]
    pose_to_hom_mat3d (CurrentPose, CurrentHomMat)

    HomMat := [HomMat, CurrentHomMat]

endfor
write_tuple (HomMat, 'Poses.txt')
```

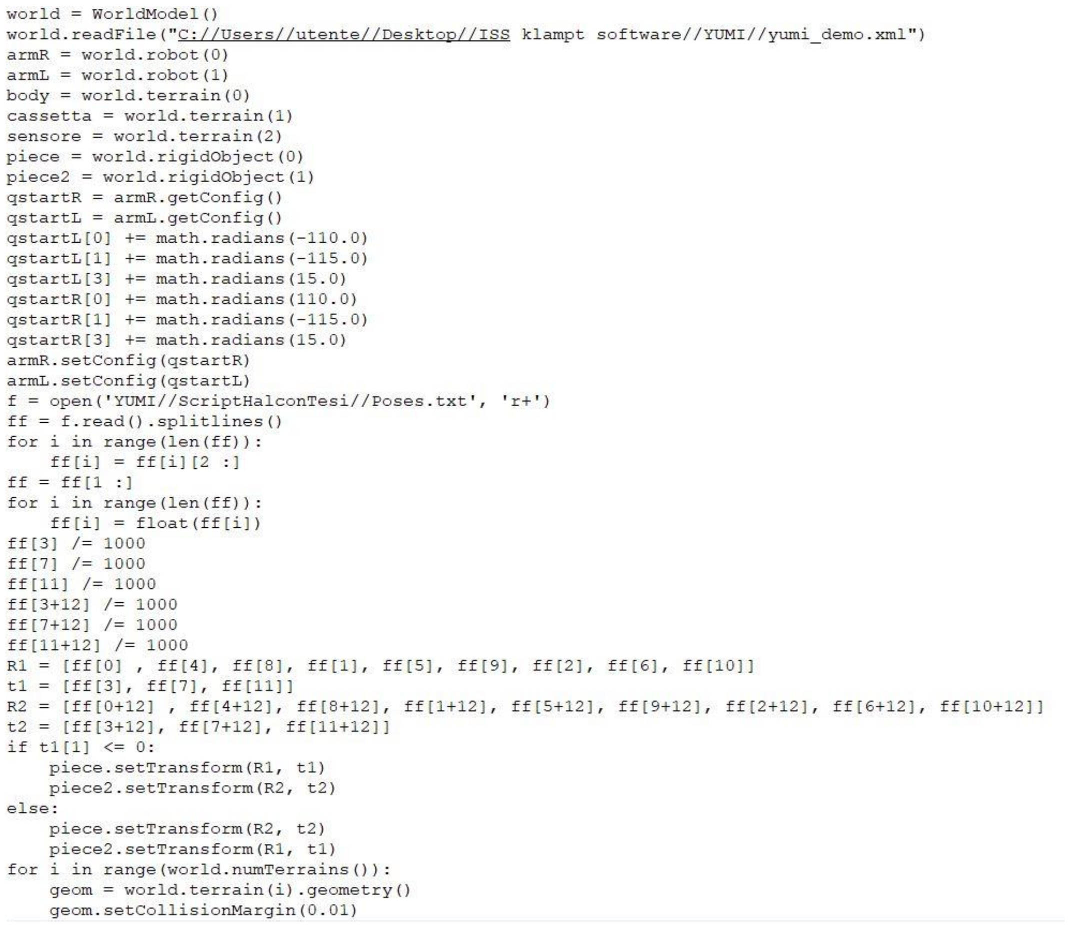
Figure 68: Halcon script part 4

5.1.3 The planning code:

After the pose estimation the user can start the main part of the software architecture, that extensively uses Klamp't library functionalities, combined with custom functions. This code as to manage all the procedures needed to ensure a correct bin picking application. These procedures have been studied and developed during the years by the ISS's engineers and are important to respect also for this thesis in order to guarantee future compatibility with the company's requirements.

Basically, the code starts with an initialization phase in which all input files are read and some virtual environments are created and configured. These environments are called "worlds" and are necessary for a correct description of the real world. They must be more than one. That's because each of them describes a different phase of the bin picking application: there will be a virtual world in which both Yumi's grippers are free and both workpieces are placed in the bin; another world describes the condition in which the right arm grasps the workpiece and the left arm is still free; there will be of course the opposite case and, finally, the case in which both the grippers grasp the objects. In this way is possible to plan in a realistic way both the approach trajectories and return ones. The alternative should have been to manage the object's picking introducing the dynamic module of the Klamp't library. This module is really heavy from a computational point of view, so its use increases the planning time and the application will become useless for real industrial tasks. Managing the problem only from a kinematic point of view instead, helps to speed up all the planning phases ensuring that time requirements are satisfied. Introducing different worlds for each possible situation in which the robot could work, can be done only one time for each bin picking task, so the time for worlds generation must be taken into account only at the first cycle.

Practically, the code reads the .XML files generating an object for each geometrical entity of the scene. Some of these objects, such as terrains, are already placed in their position respect to the world reference frame and can't be moved, some others, like robots and rigid objects, can be moved in order to place them in the position required by the application. In this case for example, each arm of the Yumi robot is placed in its home position and the two workpieces are positioned inside the bin, using informations coming from the .TXT file generated by the Halcon procedure.



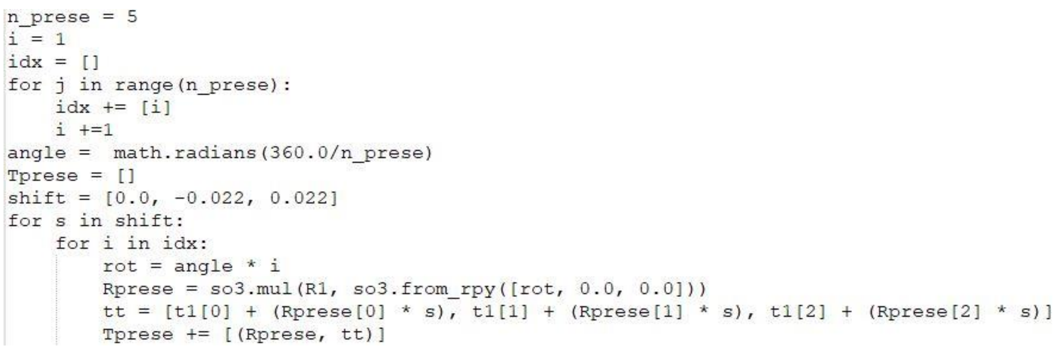
```
world = WorldModel()
world.readFile("C://Users//utente//Desktop//ISS klampt software//YUMI//yumi_demo.xml")
armR = world.robot(0)
armL = world.robot(1)
body = world.terrain(0)
cassetta = world.terrain(1)
sensore = world.terrain(2)
piece = world.rigidObject(0)
piece2 = world.rigidObject(1)
qstartR = armR.getConfig()
qstartL = armL.getConfig()
qstartL[0] += math.radians(-110.0)
qstartL[1] += math.radians(-115.0)
qstartL[3] += math.radians(15.0)
qstartR[0] += math.radians(110.0)
qstartR[1] += math.radians(-115.0)
qstartR[3] += math.radians(15.0)
armR.setConfig(qstartR)
armL.setConfig(qstartL)
f = open('YUMI//ScriptHalconTesi//Poses.txt', 'r+')
ff = f.read().splitlines()
for i in range(len(ff)):
    ff[i] = ff[i][2 :]
ff = ff[1 :]
for i in range(len(ff)):
    ff[i] = float(ff[i])
ff[3] /= 1000
ff[7] /= 1000
ff[11] /= 1000
ff[3+12] /= 1000
ff[7+12] /= 1000
ff[11+12] /= 1000
R1 = [ff[0] , ff[4], ff[8], ff[1], ff[5], ff[9], ff[2], ff[6], ff[10]]
t1 = [ff[3], ff[7], ff[11]]
R2 = [ff[0+12] , ff[4+12], ff[8+12], ff[1+12], ff[5+12], ff[9+12], ff[2+12], ff[6+12], ff[10+12]]
t2 = [ff[3+12], ff[7+12], ff[11+12]]
if t1[1] <= 0:
    piece.setTransform(R1, t1)
    piece2.setTransform(R2, t2)
else:
    piece.setTransform(R2, t2)
    piece2.setTransform(R1, t1)
for i in range(world.numTerrains()):
    geom = world.terrain(i).geometry()
    geom.setCollisionMargin(0.01)
```

Figure 69: planner part 1 (scene generation)

The last lines of this part of the code are important to assign the name “piece” always at the workpiece that in the real scene is placed more on the left. That’s because, at this stage of the development, each object is assigned as target for a specific arm of the robot, depending on its position respect to the arm itself. In the final version proposed there will be more flexibility, but this will be showed in the following subsections. After the assignment, the collision margin is initialized: for this application the threshold is set to 0,01 m.

These passages require typically a time comprised between 0.1 and 0.3 seconds for each world. In the final Yumi application, four worlds are needed. This means that the inizialization phase requires a bit more than a second, but, as already said, it has to be done only one time.

Once the world is initialized and the objects are placed in the same position of the real scene, starts one of the most important procedure for a bin picking application with a high percentage of success: the definition of the grasping points. This passage requires the proper definition of some reference frames, rotated and shifted respect to the object principal one, that correspond to the grasp points desired by the user. Defining different grasp points has two consequences: increases the possibility of a successful planning, because the principal frame of an object can be not reachable due to the presence of obstacles or to the kinematic limitations of the robot; allows to define grasping configuration that are important for the future manipulation of the object (or example, for paletizing or assembly operations). In the software commercialized by the ISS, this procedure is realized through a graphical interface that allows to define, in an easy way, frames for any kind of objects. In this thesis the definition of the grasp points is realized mathematically, knowing the shape of the object in advance. This is due to the complexity of realizing an interactive tool for the graphical definition of these points. The strategy chosen for the object used in this application, knowing the shape of Yumi's grippers is the following: define some grasping points shifted along the external circumference of the cylinder and repeat this sequence along the main axis of the cylinder itself. This approach allows to manage the entire grasp points definition with only two parameters: the number of frames around a circumference and the shift along the axis of the cylinder. In the example below the number of points along a circumference is 5 and 2 shifts of 0,022 m and -0,022 m are defined. The final number of grasp points is 15.



```
n_prese = 5
i = 1
idx = []
for j in range(n_prese):
    idx += [i]
    i +=1
angle =  math.radians(360.0/n_prese)
Tprese = []
shift = [0.0, -0.022, 0.022]
for s in shift:
    for i in idx:
        rot = angle * i
        Rprese = so3.mul(R1, so3.from_rpy([rot, 0.0, 0.0]))
        tt = [t1[0] + (Rprese[0] * s), t1[1] + (Rprese[1] * s), t1[2] + (Rprese[2] * s)]
        Tprese += [(Rprese, tt)]
```

Figure 70: planner part 2 (grasp points definition)

The result is showed in the next picture. As already said, the grasp direction is always along the z-axis, the blue one.

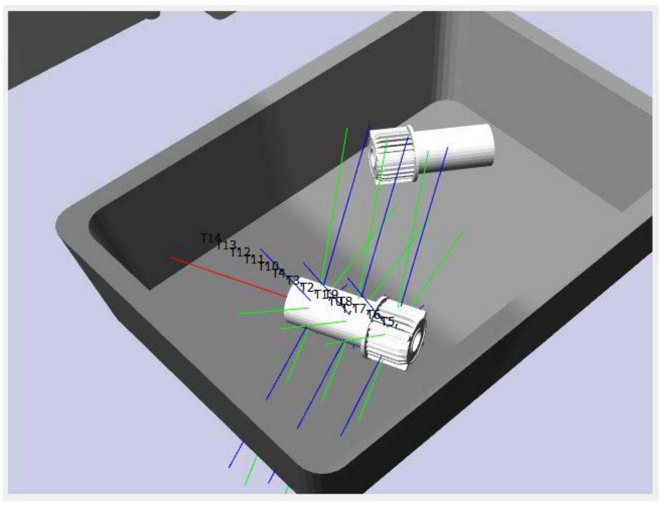


Figure 71: grasp points

After the definition of these frames is of fundamental importance to find the reachable one in the faster way possible, in order to speed up the passage to the planning phase. For this reason, these points are ordered using an index of priority that takes into account the "equivalent distance" of each frame respect to the tool one. This index is expressed as the cartesian distance between two points but takes into account also the rotations. This means that, although the loss of physical meaning (unit of measurements are not consistent), the more two frames are far and relatively rotated the one respect to the other, the more this index will be high. The grasp point with the lower index will be the first attempt for the planning phase.

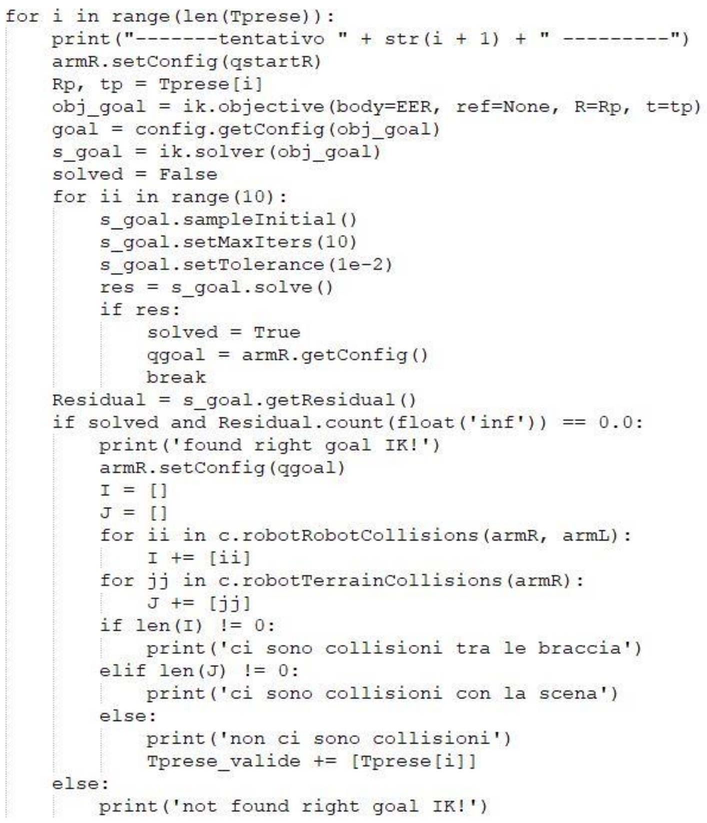
$$i = \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 + (rx - rx_0)^2 + (ry - ry_0)^2 + (rz - rz_0)^2}$$

```
distT = []
for i in range(len(Tprese)):
    distT += [se3.distance(Ttool, Tprese[i])]
Tprese_ordered = []
while Tprese != []:
    minT = min(distT)
    idx = distT.index(minT)
    Tprese_ordered += [Tprese[idx]]
    distT.remove(distT[idx])
    Tprese.remove(Tprese[idx])
```

Figure 72: planner part 3 (priorization procedure)

The equation above is introduced in the function "se3.distance" called in the procedure just showed.

Once the order of priority is defined, is possible to start one of the main procedures needed for an industrial robot motion planning: the kinematic inversion. In the application described in this thesis, the kinematic inversion is obtained through the ik module of the Klamp't library that, for redundant robots, uses a modified version of the pseudo-inverse approach, in which some jacobian inversions are by-passed trough mathematical transformations, in order to speed-up the procedure. Is important to underline that the kinematic inversion is still a numerical approximation procedure based on the Newton-Rhapson minimization approach. For this reason, is important to define a sufficient number of iterations needed to guarantee a correct inversion. The function in fact, does not produce errors if the iterations are ended without an exact solution, so it's important to verify that, at the end of the procedure, the residual between the desired final pose of the tool and the obtained one are under a certain threshold or the frame has to be discarded. Another important aspect to consider is that the kinematic inversion implemented in the Klamp't library is only a mathematical and graphical procedure, this means that the kinematic solver has no idea of the environment and does not take into account possible collision of the robot with objects, when the final configuration is computed. This means that a collision check must be performed, using a separate module of the library and, if it is not satisfied, the selected frame has to be discarded. The procedure just explained is realized with the following code.



```
for i in range(len(Tprese)):
    print("-------tentativo " + str(i + 1) + " ---------")
    armR.setConfig(qstartR)
    Rp, tp = Tprese[i]
    obj_goal = ik.objective(body=EER, ref=None, R=Rp, t=tp)
    goal = config.getConfig(obj_goal)
    s_goal = ik.solver(obj_goal)
    solved = False
    for ii in range(10):
        s_goal.sampleInitial()
        s_goal.setMaxIters(10)
        s_goal.setTolerance(1e-2)
        res = s_goal.solve()
        if res:
            solved = True
            qgoal = armR.getConfig()
            break
    Residual = s_goal.getResidual()
    if solved and Residual.count(float('inf')) == 0.0:
        print('found right goal IK!')
        armR.setConfig(qgoal)
        I = []
        J = []
        for ii in c.robotRobotCollisions(armR, armL):
            I += [ii]
        for jj in c.robotTerrainCollisions(armR):
            J += [jj]
        if len(I) != 0:
            print('ci sono collisioni tra le braccia')
        elif len(J) != 0:
            print('ci sono collisioni con la scena')
        else:
            print('non ci sono collisioni')
            Tprese_valide += [Tprese[i]]
    else:
        print('not found right goal IK!')
```

Figure 73: planner part 4 (inverse kinematics and collision checking)

In this case the maximum number of iterations is set to 10. This value provides a correct solution in most of the feasible kinematic inversion problems. Increasing the number of iterations becomes useless and introduces a lot of time losses, in which the solver tries to find a configuration for a frame that is over the joint limitations of the robot or behind an obstacle. Combining an appropriate number of iterations with the priority definition introduced before, allows to compute a feasible target configuration in about 0.005 seconds.

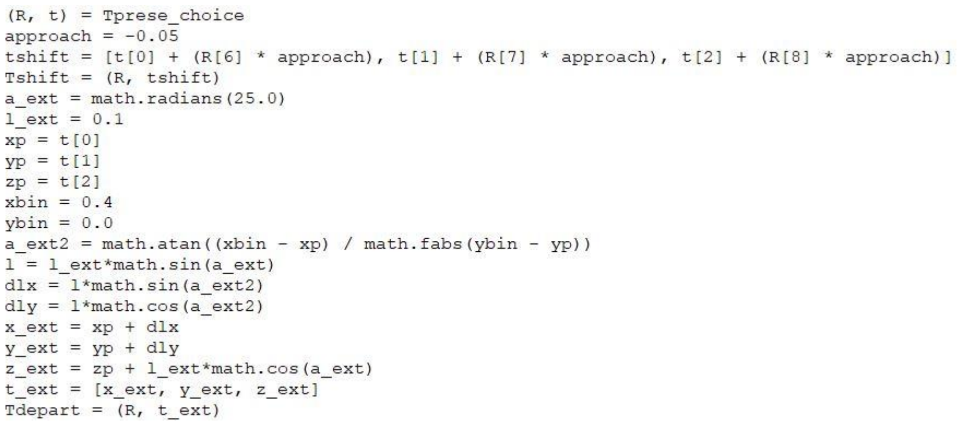
If the target configuration is valid, is possible to repeat the frame definition and the kinematic inversion for other two relevant points in a correct bin picking application. These two points are the approach point and the depart point.

The approach point is simply shifted along the z-axis of the selected grasp point, of a quantity defined by the user. This point is very important: it will be the target of the planning using the automatic planning algorithms introduced before. That’s because the successive trajectory between the approach point and the grasp one will be a simple linear, low velocity path. This strategy is important for helping the planner to compute the first trajectory. The approach point in fact, is usually farther from obstacles respect to the grasp one and allows to realize the free trajectory faster without risks of collisions with objects that can be modify the scene, invaliding the scan previously obtained from the 3Dcps.

The depart point is a safe point, usually placed around the centre of the bin, that the robot reaches after a linear path from the grasp point. In this phase the robot has already grasped the object; for this, reason this point has usually the same rotation matrix of the grasp point. In this way the extraction of the object from the bin happens keeping constant the object orientation. This strategy helps to extract objects that are near to the bin’s walls: plan a rotation of the robot’s wrist in those situations can generates collisions of the piece with the walls. This point is simply defined trough 3 parameters:

- The (x,y) coordinates of the centre of the bin;
- The angle of extraction (defined respect to the vertical axis);
- The length of extraction;

The code that computes these two points is the following one.



```
(R, t) = Tprese_choice
approach = -0.05
tshift = [t[0] + (R[6] * approach), t[1] + (R[7] * approach), t[2] + (R[8] * approach)]
Tshift = (R, tshift)
a_ext = math.radians(25.0)
l_ext = 0.1
xp = t[0]
yp = t[1]
zp = t[2]
xbin = 0.4
ybin = 0.0
a_ext2 = math.atan((xbin - xp) / math.fabs(ybin - yp))
l = l_ext*math.sin(a_ext)
dlx = l*math.sin(a_ext2)
dly = l*math.cos(a_ext2)
x_ext = xp + dlx
y_ext = yp + dly
z_ext = zp + l_ext*math.cos(a_ext)
t_ext = [x_ext, y_ext, z_ext]
Tdepart = (R, t_ext)
```

Figure 74: planner part 5 (approach and depart points definition)

In this example, the approach is shifted of 0,05 m and the depart is placed at 0,1 m along a line with an inclination of 25°.

Of course, also for these two points, a kinematic inversion is required, in order to find the correspondent, free of collision configuration to be used for the successive planning. If for one of these points, the kinematic solver fails to find a valid configuration, or the computed configuration is in collision with some obstacles, the correspondent grasp point must be discarded, passing to analyse the successive one in order of distance priority.

Typically, a complete set of points is computed and kinematically inverted in about 0,01-0,02 seconds. Of course, if a point is not feasible, the complete procedure must restart for the next set of points. If unluckily, in our example, the only feasible set corresponds to the grasp frame with the last priority index, the total computation time will be about 15 times the time for one set of points. This means about 0,3 seconds. This is the worst case and has a very low probability to happen, because usually points with high priority are very easy to invert because their position is very comfortable for the kinematics of a redundant robot. This time threshold is really appropriate for industrial applications. The points computation and the kinematic inversion phases aren't the critical from this point of view.

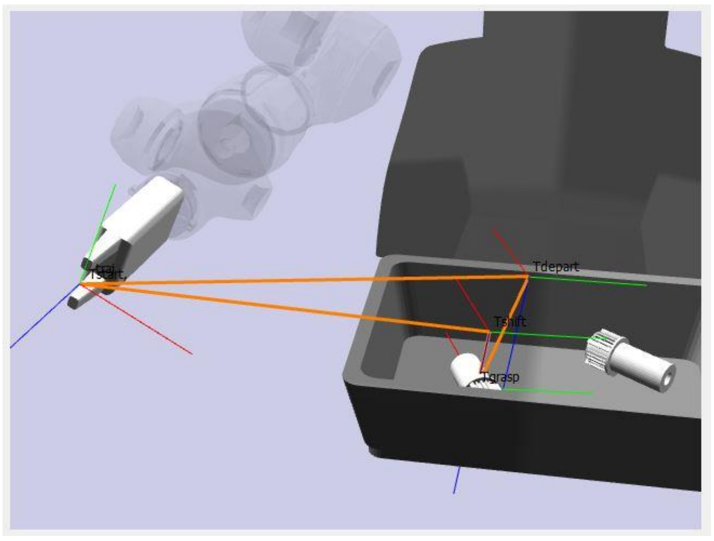
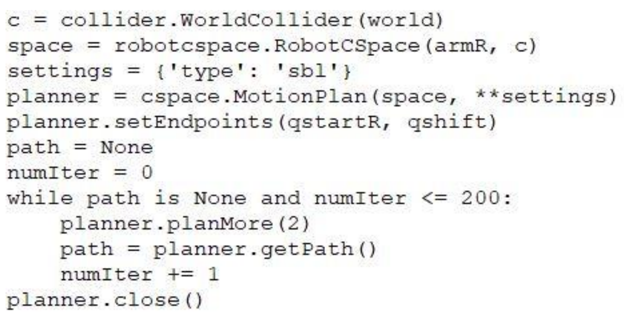


Figure 75: bin picking sequence

If the complete set of points is computed is possible to start the planning phase. At this step of the software development the points definitions and the motion planning for each arm is completely decoupled from the ones for the other. This means that, during the right planning, the planner and the kinematic collision checker see the left arm as an obstacle and for the left planning happens the opposite. Of course, the motion strategy of the real robot will be alternated: when an arm realizes the trajectories computed by the software, the other arm waits for the completed task. This is the simpler approach, in which each arm behaves like a single robot and the only difference respect to the already developed ISS software, is the managing of a scene occupied by more then a robot (In this case, two arms of the same robotic unit).

As already defined in the previous chapter, the free planning is demanded to the SBL algorithm. The procedure requires to define the collision space that the planner uses to validate the computed path through an automatic collision checking procedure. Then the world is converted into a C-space and the initial and final configuration are added to that space in order to provide the starting points for trees expansion. As already said, for the SBL algorithm, the choice of the maximum number of iterations is of fundamental importance in order to ensure a successful planning without exciding time limitations required by the application.

The Klamp't library allows to implement the automatic path planning with very few lines of code.



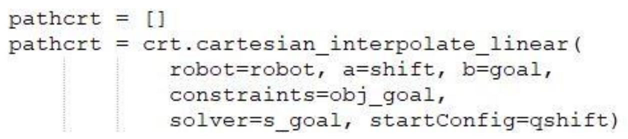
```
c = collider.WorldCollider(world)
space = robotcspace.RobotCSpace(armR, c)
settings = {'type': 'sbl'}
planner = cspace.MotionPlan(space, **settings)
planner.setEndpoints(qstartR, qshift)
path = None
numIter = 0
while path is None and numIter <= 200:
    planner.planMore(2)
    path = planner.getPath()
    numIter += 1
planner.close()
```

Figure 76: planner part 6 (planning procedure)

If the path is successfully computed it is, in any case re-arranged in order to be made of 10 configurations in the joint space. That's because all ISS applications manage 10 points for the free trajectories and 5 for the linear ones. The algorithm in most of the cases requires less configurations to find a valid free path. The additional configurations will interpolate the joint trajectories between the configurations computed by the planner, but their distribution is not equal spaced. The function tends to concentrate the additional points near the one automatically computed. This means that the real robot, that interpolates these points will tend to move slower in the regions with a higher concentration of points.

The computational time for each free path for arm of the Yumi robot is of about 1,5-1,7 seconds. This means that a complete alternated free planning requires about 6-7 seconds to be completed

The linear parts of the trajectory, the ones that connect the grasp point with the depart and approach ones, is computed calling a different procedure:



```
pathcrt = []
pathcrt = crt.cartesian_interpolate_linear(
            robot=robot, a=shift, b=goal,
            constraints=obj_goal,
            solver=s_goal, startConfig=qshift)
```

Figure 77: planner part 7 (linear path computation)

In this case the function tends to provide a trajectory made of a high number of points. That’s because the interpolator of the Klamp’t Library uses the joint movement also to realize a linear path. This means that to keep the robot tool along a straight line in the virtual environment, hundreds of very near points are required. The real robot does not have this problem because it’s sufficient to specify in the robot program to execute those points with a linear motion instead of a joint one. For this reason, the linear path is always re-arranged keeping only 5 points along the computed trajectory.

The computational time of a linear path is very low, typically about 0,003 seconds. The reason is that this part of the trajectory is not demanded to a planner algorithm so there is not automatic collision checking. Also the manual collision checking isn’t performed because, except for weird behaviours, the definition of safe approach and depart points allows to realize those small paths ensuring no collisions with the environment. Is also important to consider that some pieces require that the tool touches the bottom of the bin for a correct grasping procedure. This means that, if the linear paths would be collision-controlled, they weren’t feasible in most of the cases.

If any of the required parts of the total trajectory isn’t feasible, all the set of points is discarded and the procedure re-starts from the next point in order of priority. This isn’t a time-saving approach. It would be better from a computational time point of view to try another planning using the same set of points. The problem is that the configurations generator inside the Klamp’t planner is only pseudo-random, so if it receives the same inputs, it tends to recompute the same paths. In this case the code will never generate a safe path and it’s preferable to spend a fraction of second to compute a new set of points and try to plan for them.

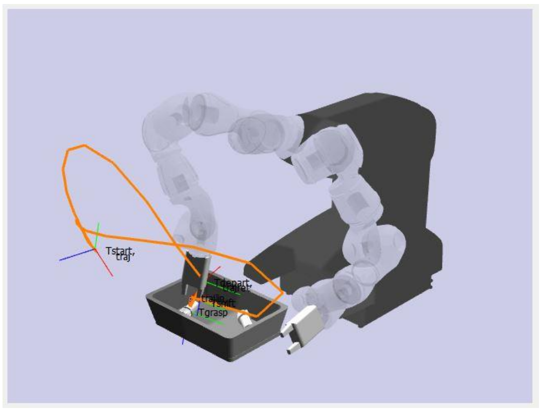
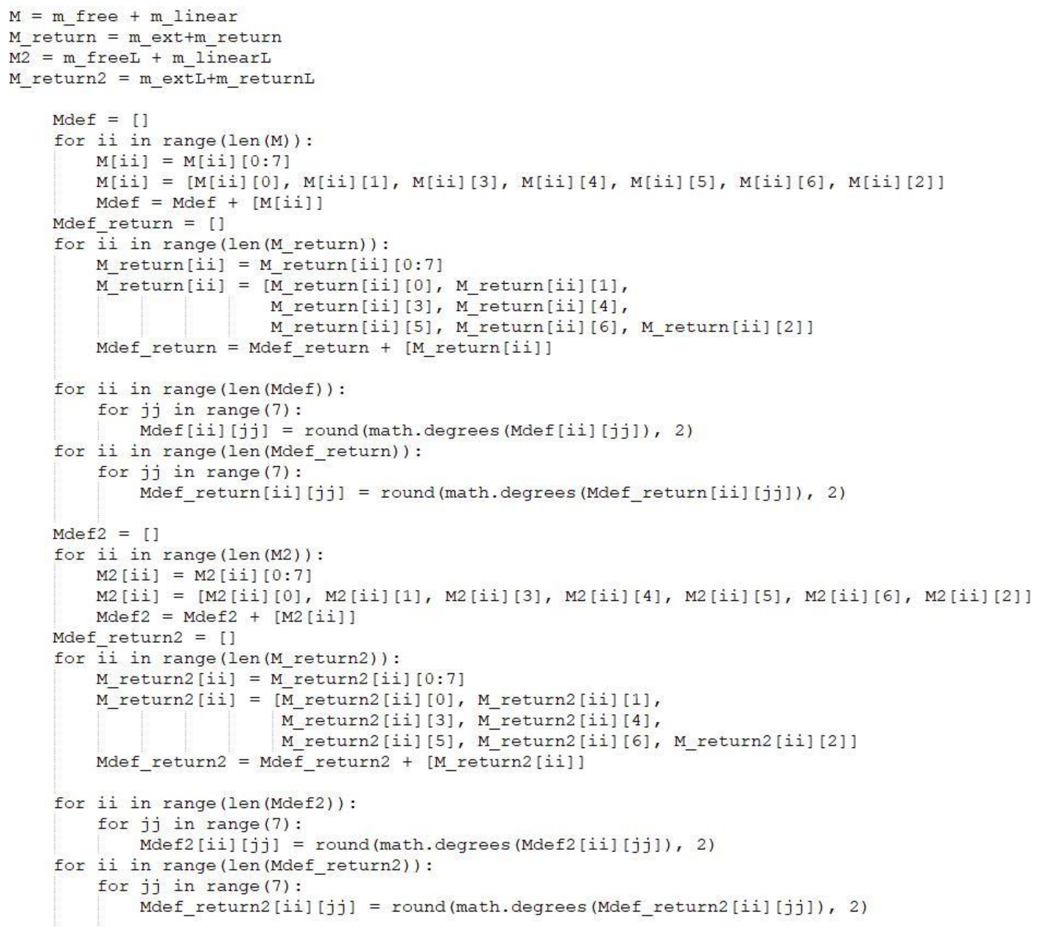


Figure 78: trajectory visualization

An important aspect to underline is the redefinition of the position of the object in the Yumi’s hand in the “return world”. When the world, describing the robot in one of the return phases, is uploaded, the piece has its main reference frame coincident with the TCP. Of course, the procedure of grasp point definition allows the possibility that in reality the robot holds the object in a different position and orientation. The algorithm is written in such way that when a complete set of points is defined, the two parameters (angle of rotation and shift along the axis) that define the position of the new grasp point respect to the principal one, are used to compute the new parent transform of the object respect to the tool. This procedure is actuated before the start of the extraction and return planning. Once all the trajectories for both arms are computed is time to start the communication with the real robot. The communication procedure is based on a TCP-IP standard protocol. First of all, the software re-elaborates the computed trajectories in order to provide them to the robot in right way. Each configuration is a list of 7 values. The redundant joint is the seventh but in the kinematic chain provided to the software it is the third. For this reason, is important to re-arrange each list, before the robot starts to analyse and interpolate them. The second aspects to consider is the conversion in degrees. The software based on the Klamp’t library generates all configurations in radians. The ABB programming language works with degrees. Finally, each value has to be rounded to the second decimal unit. In fact, there is a limitation on the number of characters that an ABB robot can receive trough ethernet connection and the software tends to generate joint values with a lot of decimal values. This re-arranging procedure is showed in picture 78.

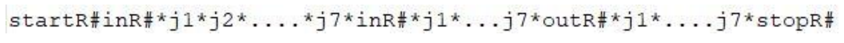


```
M = m_free + m_linear
M_return = m_ext+m_return
M2 = m_freeL + m_linearL
M_return2 = m_extL+m_returnL

    Mdef = []
    for ii in range(len(M)):
        M[ii] = M[ii][0:7]
        M[ii] = [M[ii][0], M[ii][1], M[ii][3], M[ii][4], M[ii][5], M[ii][6], M[ii][2]]
        Mdef = Mdef + [M[ii]]
    Mdef_return = []
    for ii in range(len(M_return)):
        M_return[ii] = M_return[ii][0:7]
        M_return[ii] = [M_return[ii][0], M_return[ii][1],
                        M_return[ii][3], M_return[ii][4],
                        M_return[ii][5], M_return[ii][6], M_return[ii][2]]
        Mdef_return = Mdef_return + [M_return[ii]]

    for ii in range(len(Mdef)):
        for jj in range(7):
            Mdef[ii][jj] = round(math.degrees(Mdef[ii][jj]), 2)
    for ii in range(len(Mdef_return)):
        for jj in range(7):
            Mdef_return[ii][jj] = round(math.degrees(Mdef_return[ii][jj]), 2)

    Mdef2 = []
    for ii in range(len(M2)):
        M2[ii] = M2[ii][0:7]
        M2[ii] = [M2[ii][0], M2[ii][1], M2[ii][3], M2[ii][4], M2[ii][5], M2[ii][6], M2[ii][2]]
        Mdef2 = Mdef2 + [M2[ii]]
    Mdef_return2 = []
    for ii in range(len(M_return2)):
        M_return2[ii] = M_return2[ii][0:7]
        M_return2[ii] = [M_return2[ii][0], M_return2[ii][1],
                         M_return2[ii][3], M_return2[ii][4],
                         M_return2[ii][5], M_return2[ii][6], M_return2[ii][2]]
        Mdef_return2 = Mdef_return2 + [M_return2[ii]]

    for ii in range(len(Mdef2)):
        for jj in range(7):
            Mdef2[ii][jj] = round(math.degrees(Mdef2[ii][jj]), 2)
    for ii in range(len(Mdef_return2)):
        for jj in range(7):
            Mdef_return2[ii][jj] = round(math.degrees(Mdef_return2[ii][jj]), 2)
```

Figure 79: planner part 8 (trajectory re-writing)

The sequence of messages established with the robot is very simple. The communication protocol of the robot continues to generate the string “send”, until it receives from the computer in which the software is running, the instruction “stop” for both the right and the left arms.

The overall string containing the total trajectory for each arm has the following syntax:

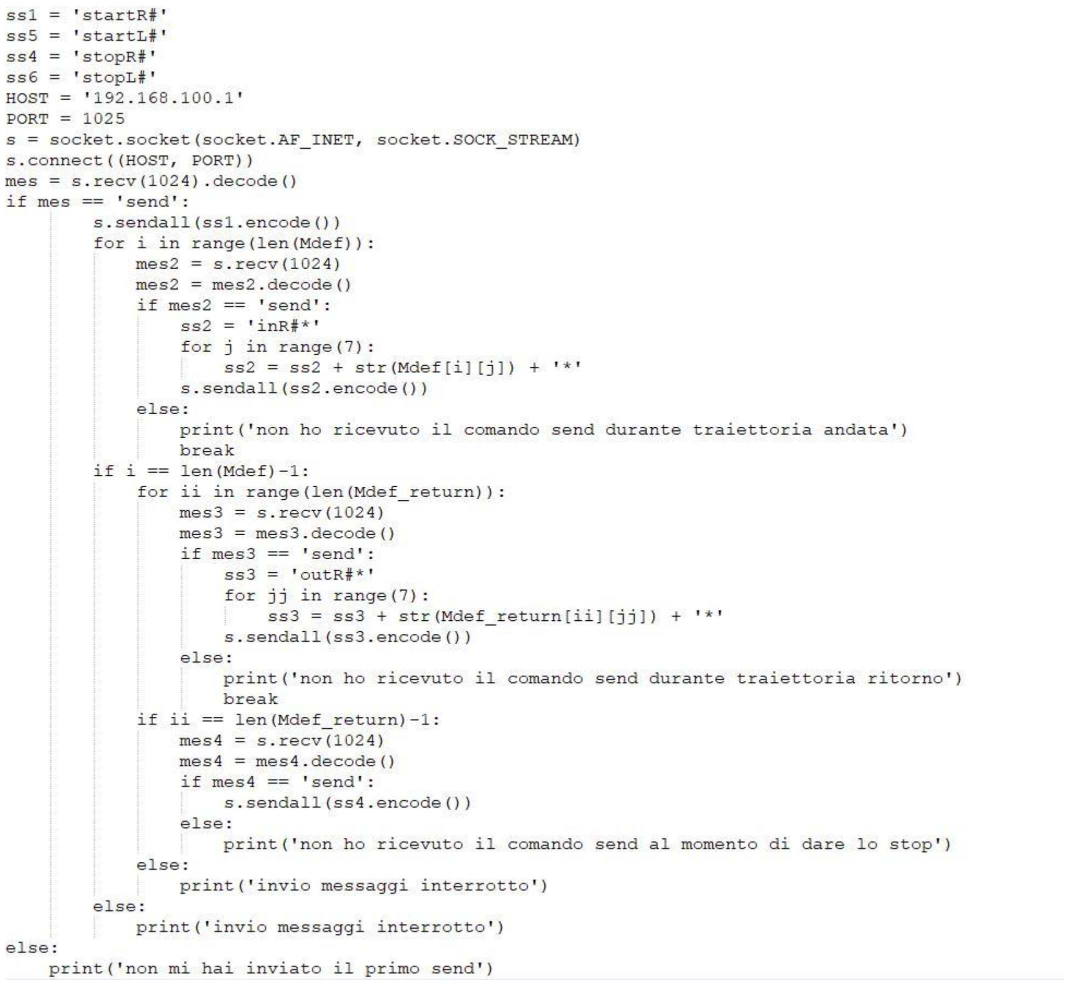


```
startR#inR#*j1*j2*....*j7*inR#*j1*...j7*outR#*j1*....j7*stopR#
```

Figure 80: example of string

In which the symbol “R” or “L” allows to distinguish between right or left trajectories.

The composition of this string is obtained trough a simple procedure, in which any possible error of communication is managed with an exit from the cycle and the interruption of the procedure itself.



```
ss1 = 'startR#'
ss5 = 'startL#'
ss4 = 'stopR#'
ss6 = 'stopL#'
HOST = '192.168.100.1'
PORT = 1025
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
mes = s.recv(1024).decode()
if mes == 'send':
        s.sendall(ss1.encode())
        for i in range(len(Mdef)):
            mes2 = s.recv(1024)
            mes2 = mes2.decode()
            if mes2 == 'send':
                ss2 = 'inR#*'
                for j in range(7):
                    ss2 = ss2 + str(Mdef[i][j]) + '*'
                s.sendall(ss2.encode())
            else:
                print('non ho ricevuto il comando send durante traiettoria andata')
                break
        if i == len(Mdef)-1:
            for ii in range(len(Mdef_return)):
                mes3 = s.recv(1024)
                mes3 = mes3.decode()
                if mes3 == 'send':
                    ss3 = 'outR#*'
                    for jj in range(7):
                        ss3 = ss3 + str(Mdef_return[ii][jj]) + '*'
                    s.sendall(ss3.encode())
                else:
                    print('non ho ricevuto il comando send durante traiettoria ritorno')
                    break
            if ii == len(Mdef_return)-1:
                mes4 = s.recv(1024)
                mes4 = mes4.decode()
                if mes4 == 'send':
                    s.sendall(ss4.encode())
                else:
                    print('non ho ricevuto il comando send al momento di dare lo stop')
            else:
                print('invio messaggi interrotto')
        else:
            print('invio messaggi interrotto')
else:
    print('non mi hai inviato il primo send')
```

Figure 81: planner part 9 (communication protocol)

The communication protocol spends about 2 seconds to send all the trajectories to the robot. This means that this basic version of the software requires about 7-8 seconds per cycle, from the elaboration of the informations coming from the scanner to the complete generation of the trajectories. This is a good result considering that the software is not optimized from a computational point of view. It is written in python that is a pre-compiled high-level programming language that is typically too slow for industrial applications. Another aspect is that the Python version of the Library is simply a wrapper of the basic functionalities written in C++. This approach allows an easier and more compact access to these functions, making easier the software prototyping, but the computer requires more time to execute each operation. The idea is that in the future, rewriting the software in a more efficient low-level programming language and optimizing computer performances will be possible to reduce the software computational time making possible an efficient industrial implementation.

5.1.4 The Yumi Program

The last part of the software architecture to describe is the one uploaded directly on the robot controller, responsible of the communication with the planning software and of the final execution of the movements.

The Yumi Robot is a double-arm collaborative robot, developed by the Swiss company ABB. Its main features are the high level of dexterity and its great flexibility, sensibility and precision. It was thought for assembly operations, but thanks to these characteristics it can be used also for particular bin picking applications: it has a working range comparable to the one of a human operator and, thanks to its intrinsic safeness, it can be used to feed a line, a conveyor, a machine or to cooperate with a human operator.

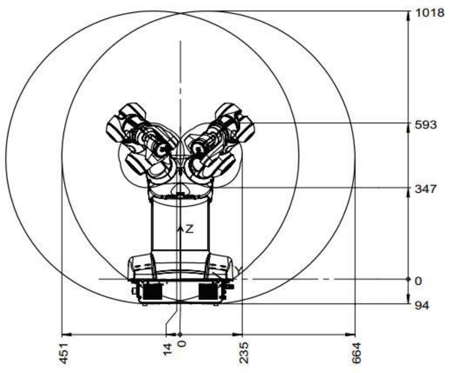


Figure 82: Yumi frontal ROI

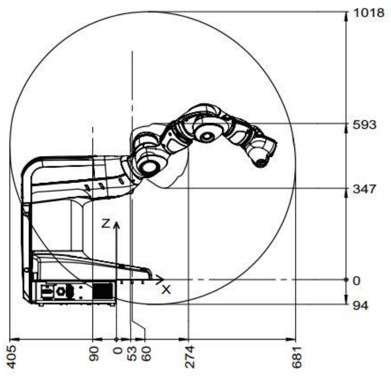


Figure 83: Yumi lateral ROI

From the hardware point of view, it has a unique control unit, but the motion control of each mechanical unity is managed independently. This means that is possible to upload on the controller a unique script with supervision and coordination functions but is of fundamental importance to define always two separate scripts for the motion generation of arms and grippers.

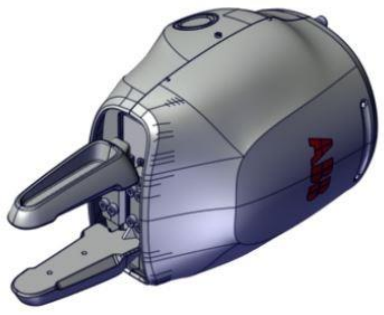


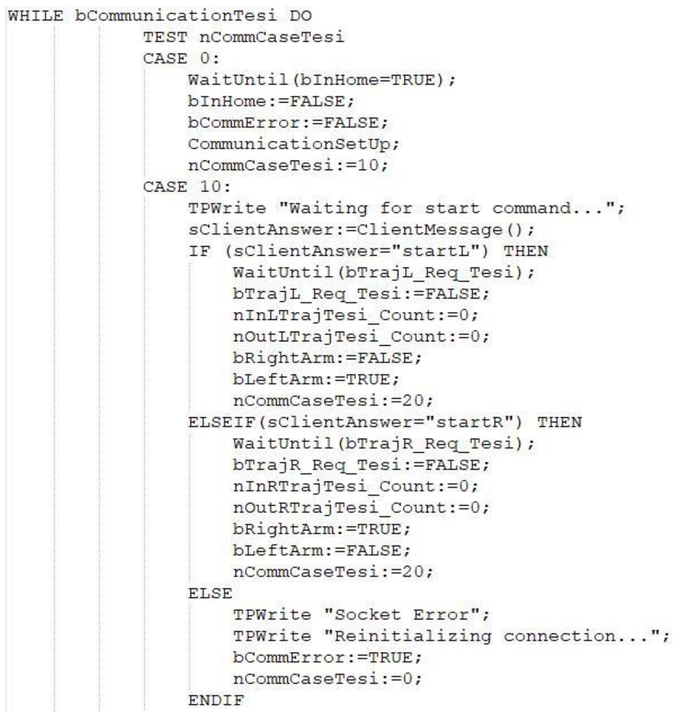
Figure 84: Yumi gripper

In the case of this thesis, three main scripts are defined:

- Tcom_tesi: opens the TCP-IP communication with the planner software, elaborates the messages received and sends the obtained trajectories to the appropriate motion code;
- T_robR: manage the movements of the right arm and gripper;
- T_robL: manage the movements of the left arm and gripper.

The communication on the robot side is based on a switch-case architecture. This strategy allows the manage the flow of the code with a sequence of bits ensuring no errors in the procedure of message analysis and trajectories generation.

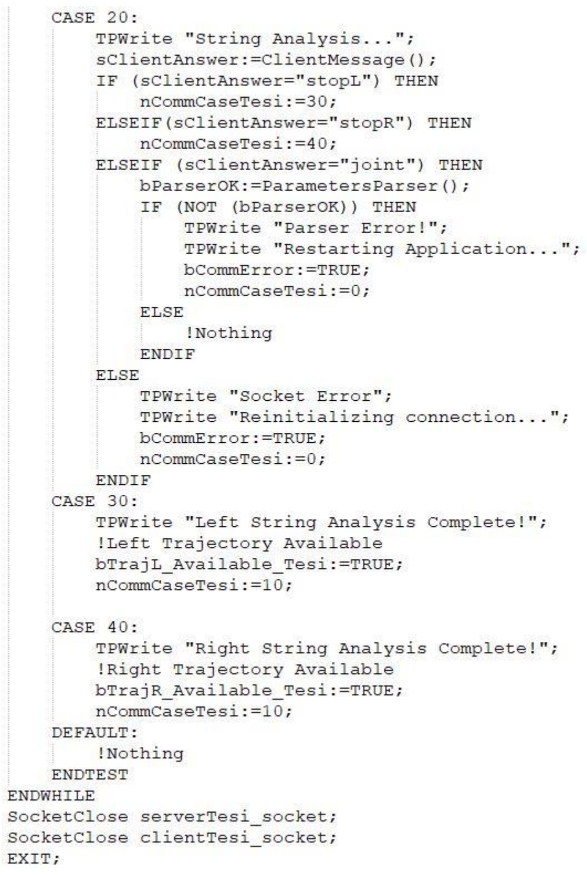
In the following lines of code, the main communication protocol is showed. Some custom functions are present that, for brevity will be only described without showing the code.



```
WHILE bCommunicationTesi DO
            TEST nCommCaseTesi
            CASE 0:
                WaitUntil(bInHome=TRUE);
                bInHome:=FALSE;
                bCommError:=FALSE;
                CommunicationSetUp;
                nCommCaseTesi:=10;
            CASE 10:
                TPWrite "Waiting for start command...";
                sClientAnswer:=ClientMessage();
                IF (sClientAnswer="startL") THEN
                    WaitUntil(bTrajL_Req_Tesi);
                    bTrajL_Req_Tesi:=FALSE;
                    nInLTrajTesi_Count:=0;
                    nOutLTrajTesi_Count:=0;
                    bRightArm:=FALSE;
                    bLeftArm:=TRUE;
                    nCommCaseTesi:=20;
                ELSEIF(sClientAnswer="startR") THEN
                    WaitUntil(bTrajR_Req_Tesi);
                    bTrajR_Req_Tesi:=FALSE;
                    nInRTrajTesi_Count:=0;
                    nOutRTrajTesi_Count:=0;
                    bRightArm:=TRUE;
                    bLeftArm:=FALSE;
                    nCommCaseTesi:=20;
                ELSE
                    TPWrite "Socket Error";
                    TPWrite "Reinitializing connection...";
                    bCommError:=TRUE;
                    nCommCaseTesi:=0;
                ENDIF
```

Figure 85: Rapid main communication protocol (part 1)

The first part of the protocol (case 0) waits for arms in home position, then opens the serial communication. When the operation is complete it goes to case 10. The case 10 uses the function ClientMessage() to analyse the first part of the message, containing the information about which mechanical unit should receive the trajectory. If there aren’t errors in the “start” message syntax, the protocol goes to case 20.



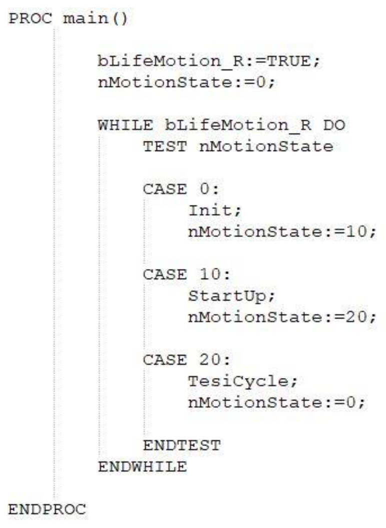
```
    CASE 20:
        TPWrite "String Analysis...";
        sClientAnswer:=ClientMessage();
        IF (sClientAnswer="stopL") THEN
            nCommCaseTesi:=30;
        ELSEIF(sClientAnswer="stopR") THEN
            nCommCaseTesi:=40;
        ELSEIF (sClientAnswer="joint") THEN
            bParserOK:=ParametersParser();
            IF (NOT (bParserOK)) THEN
                TPWrite "Parser Error!";
                TPWrite "Restarting Application...";
                bCommError:=TRUE;
                nCommCaseTesi:=0;
            ELSE
                !Nothing
            ENDIF
        ELSE
            TPWrite "Socket Error";
            TPWrite "Reinitializing connection...";
            bCommError:=TRUE;
            nCommCaseTesi:=0;
        ENDIF
    CASE 30:
        TPWrite "Left String Analysis Complete!";
        !Left Trajectory Available
        bTrajL_Available_Tesi:=TRUE;
        nCommCaseTesi:=10;

    CASE 40:
        TPWrite "Right String Analysis Complete!";
        !Right Trajectory Available
        bTrajR_Available_Tesi:=TRUE;
        nCommCaseTesi:=10;
    DEFAULT:
        !Nothing
    ENDTEST
ENDWHILE
SocketClose serverTesi_socket;
SocketClose clientTesi_socket;
EXIT;
```

Figure 86: Rapid communication protocol (part 2)

In case 20 the remaining part of the message is analysed. If it uses the keyword “joint”, the function ParametersParser() starts. It has the function of removing useless characters from the message and save all the obtained joint values in vectors available also for the scripts responsible of movements. If the message arrives to the keyword “stopR” or “stopL”, the script understands that the string analysis is successfully completed and gives to the other two scripts the possibility to start generating movements.

Codes related to the arm movement are equal both for right and left arm at this point of the development. For this reason, only one is showed and explained.

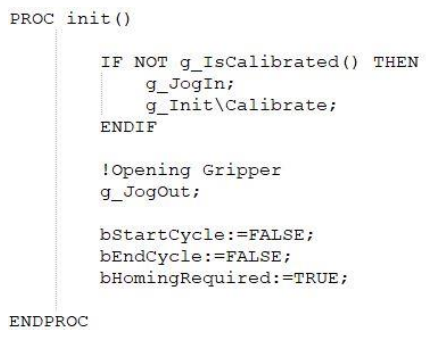
Also in this case the basic procedure involves a sequence of switch cases, governed by a series of bits that changes their status only when a sequence is totally completed. In this way Is possible to easily control the series of operation performed by the robot and avoid any possible error.



```
PROC main()

        bLifeMotion_R:=TRUE;
        nMotionState:=0;

        WHILE bLifeMotion_R DO
            TEST nMotionState

            CASE 0:
                Init;
                nMotionState:=10;

            CASE 10:
                StartUp;
                nMotionState:=20;

            CASE 20:
                TesiCycle;
                nMotionState:=0;

            ENDTEST
        ENDWHILE

ENDPROC
```

Figure 87: Rapid motion main protocol

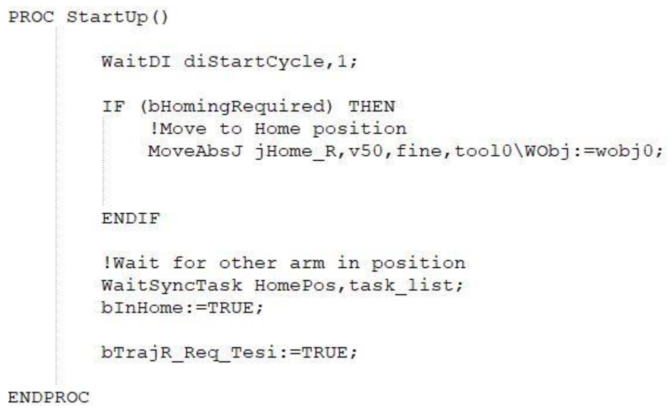
Basically, the code enters in case 0 that starts the “init” procedure. This procedure should be repeated at each cycle. It is necessary for the robot to understand in which position are their grippers and to open them. In order to perform a correct approach to the workpiece is important to ensure that the gripper is open before the start of the trajectory.



```
PROC init()

        IF NOT g_IsCalibrated() THEN
            g_JogIn;
            g_Init\Calibrate;
        ENDIF

        !Opening Gripper
        g_JogOut;

        bStartCycle:=FALSE;
        bEndCycle:=FALSE;
        bHomingRequired:=TRUE;

ENDPROC
```

Figure 88: Rapid "init" procedure

At the end of the “init” procedure, the code enters in the case 10, that is responsible of the homing procedure of both arms, named “StartUp”.

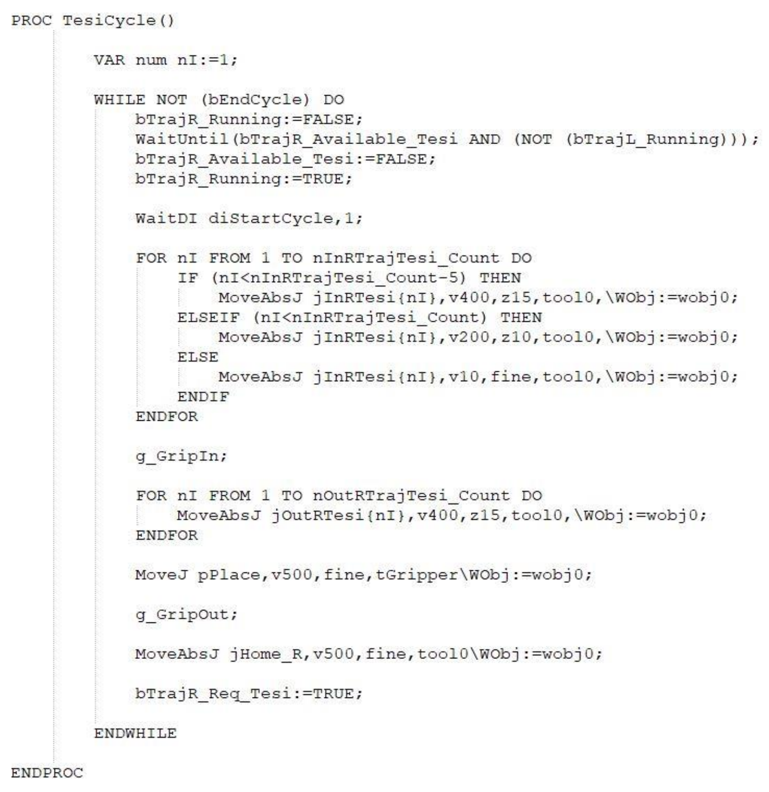


```
PROC StartUp()

        WaitDI diStartCycle,1;

        IF (bHomingRequired) THEN
            !Move to Home position
            MoveAbsJ jHome_R,v50,fine,tool0\WObj:=wobj0;


        ENDIF

        !Wait for other arm in position
        WaitSyncTask HomePos,task_list;
        bInHome:=TRUE;

        bTrajR_Req_Tesi:=TRUE;

ENDPROC
```

Figure 89: Rapid "StartUp" procedure

When both arms are in home position, the robot is in the same configuration used to start the automatic motion planning in the planner algorithm. In this configuration is possible to start the trajectory execution, entering the case 20 and starting the “TesiCycle” procedure. The sequence of operations is really simple:

- The robot starts executing the first part of the trajectory. As explained before, for consistency, each part of the trajectory is made of 15 points. The last 5 belong on the approach line. For this reason, they are executed at limited velocity. If some additional, weird point is provided to the robot and the communication protocol does not recognize the error, it is interpolated at very low velocity, in order to give the operator the possibility to manually stop the execution of the task.
- When the in trajectory is completed, the robot is expected to be in correspondence of the object, so the code closes robot grippers.
- At this point, the object is supposed to be grasped by the robot. The out trajectory can start. It is all interpolated at the same, intermediate velocity. At the end of this trajectory the arm is come to the home position. The robot has finished to interpolate points generated by the planning algorithm.
- The task finishes with a place operation, the opening of the gripper and a fast return to home position.



```
PROC TesiCycle()

        VAR num nI:=1;

        WHILE NOT (bEndCycle) DO
            bTrajR_Running:=FALSE;
            WaitUntil(bTrajR_Available_Tesi AND (NOT (bTrajL_Running)));
            bTrajR_Available_Tesi:=FALSE;
            bTrajR_Running:=TRUE;

            WaitDI diStartCycle,1;

            FOR nI FROM 1 TO nInRTrajTesi_Count DO
                IF (nI<nInRTrajTesi_Count-5) THEN
                    MoveAbsJ jInRTesi{nI},v400,z15,tool0,\WObj:=wobj0;
                ELSEIF (nI<nInRTrajTesi_Count) THEN
                    MoveAbsJ jInRTesi{nI},v200,z10,tool0,\WObj:=wobj0;
                ELSE
                    MoveAbsJ jInRTesi{nI},v10,fine,tool0,\WObj:=wobj0;
                ENDIF
            ENDFOR

            g_GripIn;

            FOR nI FROM 1 TO nOutRTrajTesi_Count DO
                MoveAbsJ jOutRTesi{nI},v400,z15,tool0,\WObj:=wobj0;
            ENDFOR

            MoveJ pPlace,v500,fine,tGripper\WObj:=wobj0;

            g_GripOut;

            MoveAbsJ jHome_R,v500,fine,tool0\WObj:=wobj0;

            bTrajR_Req_Tesi:=TRUE;

        ENDWHILE

ENDPROC
```

Figure 90: Rapid "TesiCycle" procedure

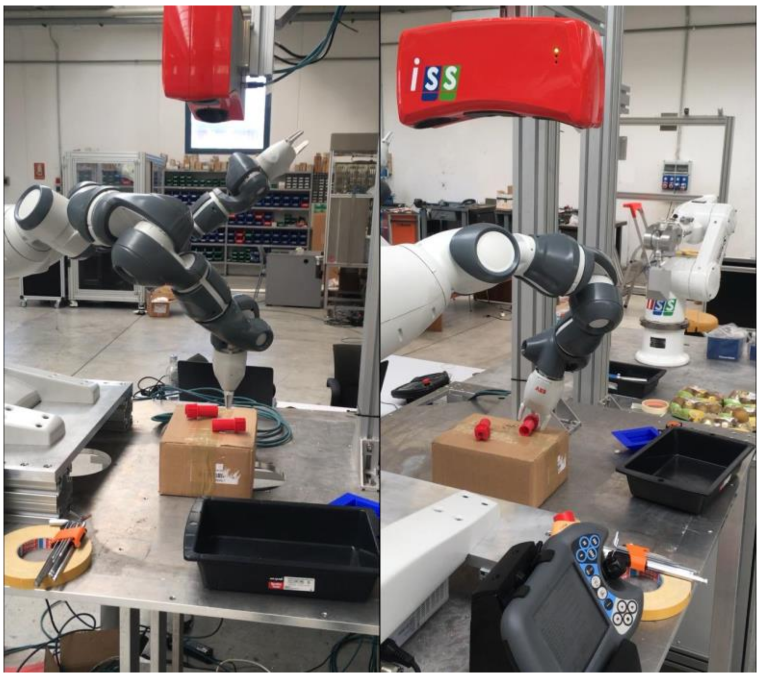


Figure 91: some images of the experimental tests

5.2 THE FINAL VERSION OF THE SOFTWARE

In the subsection above the basic version of the software has been described. As already said this version analyses the position of the scanned object separately and plan for each arm taking into account that the other arm is a static obstacle and does not interact with the scene until the other arm has completed its operation. This approach, called “alternated, multi-arm approach” is the easier and most intuitive one. It corresponds to manage the standard bin picking application with two independent robots. The only difference respect to the already commercialized version of ISS planner is the capability of managing a redundant kinematic chain. From this point of view the Klamp’t library has provided successful results, solving the redundant inverse kinematics in very short time.

The alternated multi-arm approach has two main drawbacks:

- First of all, there is an intrinsic rigidity in the assignment of the object more on the right to the right arm and of the object more on the left to the left to the left arm. This choice reduces the capability of the software of successfully complete the task. In some cases, in fact, is preferable to repeat the bin picking operation with the same arm for both the objects, that are easier to plan for an arm. That’s because the region of interest of each arm typically does not allow to reach the opposite part of the bin. This means that, in the case both objects are tendentially placed in the same part of the bin, is preferable to plan both times for the same arm.
  For this reason, the final version of the software proposed in this thesis introduces a simple pre-processing operation that analyses the position of the objects and chooses which strategy use to maximize the probability to complete the bin picking operation.
- The second, more important aspect is related to the possibility of the Yumi to perform coordinated movement with both arms, miming a human being. This capability should increase the overall productivity of the bin picking task, giving the possibility to feed two lines, machines or operators at the same time. Another possible future application is the possibility to pick different pieces from the same bin and perform an assembly operation. In this case the planner does not manage two standard robots for a repeated bin picking operation but plan a complex task managing a complete multi-arm robotic unit. This result should open new possibilities in this particular industrial filed, introducing functionalities never developed before by the company, useful for the final purpose of a bin picking application studied for collaborative assembling operations. For this reason, a new procedure, able to plan a coordinated, free of self-collisions trajectory for both arms, has been developed.

Both the problems have been already analysed in the technical literature and a lot of complex solutions have been proposed.

5.2.1 The theoretical background

From the point of view of the target selection, most of the theoretical procedures involve optimality functions, dynamic programming approaches or minimization algorithms. These approaches are typically complex from an implementation point of view. That’s because the theoretical and mathematical definition and solution to the problem are difficult to translate in an effective code written in a standard programming language. Another aspect to take into account is the computational time. Most of these methods provides very good results, allowing the coordinated motion planning of many objects in really complex configuration spaces, even more complex of the one of a double arm redundant robot, but the time required to find a solution is usually unfeasible for industrial applications.

In this work a real simple solution to selection strategy is proposed, that evaluates the relative and absolute position of the objects and chooses the correct bin picking procedure.

In the field of motion coordination instead, one of the main approaches, developed in different ways, starting from different planning algorithms, such as the already described RRT or the PRM one, is the roadmap composition or the super-graph construction.

All sampling-based planners create a tree or graph. This graph is a sampled mapping of the configuration space. Along this graph the collision checking is performed and the final trajectory is constructed. Some approaches to the coordinated motion planning create graphs for each robot, then a super-graph is constructed by the composition of the elementary trees. The composition procedure can be realized in different ways. The basic idea is to find a way to check the collision with the environment for each robot but also the mutual collisions between robots along the computed roadmaps. One of the main solutions to this problem is the evaluation of the cartesian product of the nodes belonging on the elementary graphs.

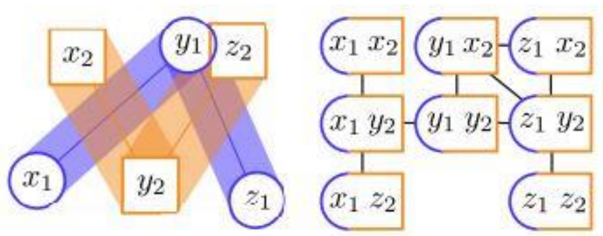


Figure 92: super-graph composition

In other cases, the coordination does not happen at the configuration level. Trajectories for each robot are computed completely independently without an iterative procedure that continues to expand trees until two independent, free-of-collisions trajectories are found. Some multi-robot planners try to solve the possibility of collisions with a complete control of the robot kinematics. When to robots are expected to collide due to the coincidence of their milestones along their separate graphs, the planner defines a master robot and modify the slave robot velocity in order to allow the passage of both robots in that configuration but at different times. This solution is way easier from the implementation point of view and is also faster because requires the restart of the cycle only in the case of planning failure due to the unfeasibility of the target or the presence of unavoidable obstacles. The main drawback is that the planning strategy of the ISS softwares are not based on a complete kinematic control of the robot: trajectories must be safe without regarding to the velocity used for their actuation.

The third and last approach to coordinated motion planning is based on a graphical reconstruction of the configuration space. Basically, as in the previous strategy, a robot is declared as the master one. The motion planning for this robot completely ignores the necessity to plan also for the slave robot and takes into account is presence only as a static obstacle. When the motion planning for the master robot is successfully complete, the obtained trajectory is used to reconfigure the configuration space seen by the slave robot. Each milestone of the master path is used to create a copy of the master robot; this is not another robot but just a geometrical entity with the same shape and dimension of the master robot, placed in all the configurations that the master robot assumes during is trajectory. All these copies will be used in the slave configuration space as obstacles. The result is that the slave path, if exist, will be always free of collision with the master one. This approach penalizes the slave robot, that, with a different coordination strategy, would have the possibility to use more space, increasing its possibilities of a successful planning. The idea is of sacrificing the flexibility and reliability of the algorithm, trying to realize a strategy that is easier to implement from a coding point of view and is potentially quite fast to be implemented in industrial applications.

Of course, this approach too coordinated motion planning cannot be used in any case. If the objects to grasp are too near the master trajectory will obstacle the slave one in most of the cases and a complete grasping task will become impossible.

For this reason, the idea is to demand to the selection strategy the responsibility to choose the right type of planning for the specific situation, choosing the coordinated one only if some distance thresholds are respected.

The final result is a software that analyses the position of targets and decides how to behave:

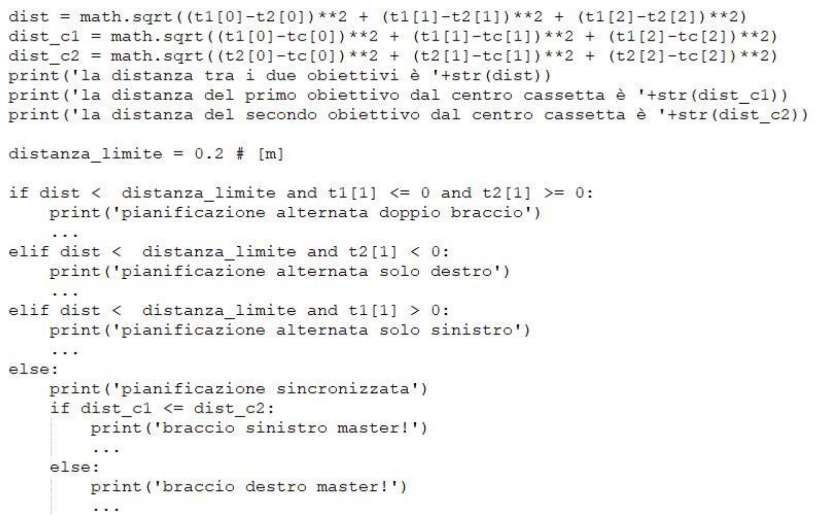
- grasps them using both arms alternatively;
- grasps using both arms at the same time;
- use only the right arm;
- use only the left arm.

The results in terms of elaboration time and percentage of success will be studied varying the distance threshold.

Is important to remind that the Klamp’t library does not provide built-in methods for coordinated motion planning of multiple robots. For this reason, the decision has been to use some of its features to implement a custom strategy, able to provide collision free trajectories for both arms, miming a coordination.

5.2.2 Proposed solutions

For the choice of the picking strategy, a simple if-else algorithm has been realized.



```
dist = math.sqrt((t1[0]-t2[0])**2 + (t1[1]-t2[1])**2 + (t1[2]-t2[2])**2)
dist_c1 = math.sqrt((t1[0]-tc[0])**2 + (t1[1]-tc[1])**2 + (t1[2]-tc[2])**2)
dist_c2 = math.sqrt((t2[0]-tc[0])**2 + (t2[1]-tc[1])**2 + (t2[2]-tc[2])**2)
print('la distanza tra i due obiettivi è '+str(dist))
print('la distanza del primo obiettivo dal centro cassetta è '+str(dist_c1))
print('la distanza del secondo obiettivo dal centro cassetta è '+str(dist_c2))

distanza_limite = 0.2 # [m]

if dist <  distanza_limite and t1[1] <= 0 and t2[1] >= 0:
    print('pianificazione alternata doppio braccio')
    ...
elif dist <  distanza_limite and t2[1] < 0:
    print('pianificazione alternata solo destro')
    ...
elif dist <  distanza_limite and t1[1] > 0:
    print('pianificazione alternata solo sinistro')
    ...
else:
    print('pianificazione sincronizzata')
    if dist_c1 <= dist_c2:
        print('braccio sinistro master!')
        ...
    else:
        print('braccio destro master!')
        ...
```

Figure 93: Planner v2 (strategy selection)

It is based basically on 3 parameters:

- The distance of the first object from the centre of the bin;
- The distance of the second object from the centre of the bin;
- The relative distance between the 2 objects;

If the relative distance is over a certain threshold, in this example 0,2 meters, the planner tries in any case to implement the coordinate planning with two possibilities:

- If the distance of the first object (always the one assigned to the right arm) from the centre of the bin is more than the distance of the second object, the right arm is the master arm for the planning procedure;
- If the second object is farther from the centre of the bin, the master arm will be the left one.

This strategy is based on the following assumption: if the target is farther from the centre of the bin the correspondent planned trajectory should remain far from the region of interest of the other target. In this way, for the slave arm, there will be more free space to try to plan a free-of-collision path.

If the distance between objects is under the threshold, coordinated planning is always avoided and the software tries to implement the alternated planning strategy with both arms or only one arm, depending on the absolute position of the objects in the bin.

For what concern the alternated strategies, there aren’t relevant variation in the code respect to the basic case introduced in the previous subsection.

The coordinated motion planning is based on the “update” of the scene seen by the slave arm. This is possible through a particular part of the Klamp’t library capable of managing .STL files directly from the script without manually updating the .XML file. These functions allow to import links and tools of the robot as they are rigid objects and, managing the trajectory milestones, define their position in the workspace.

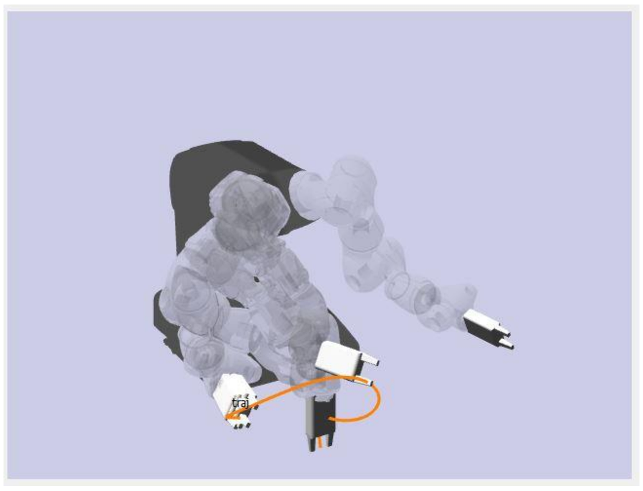
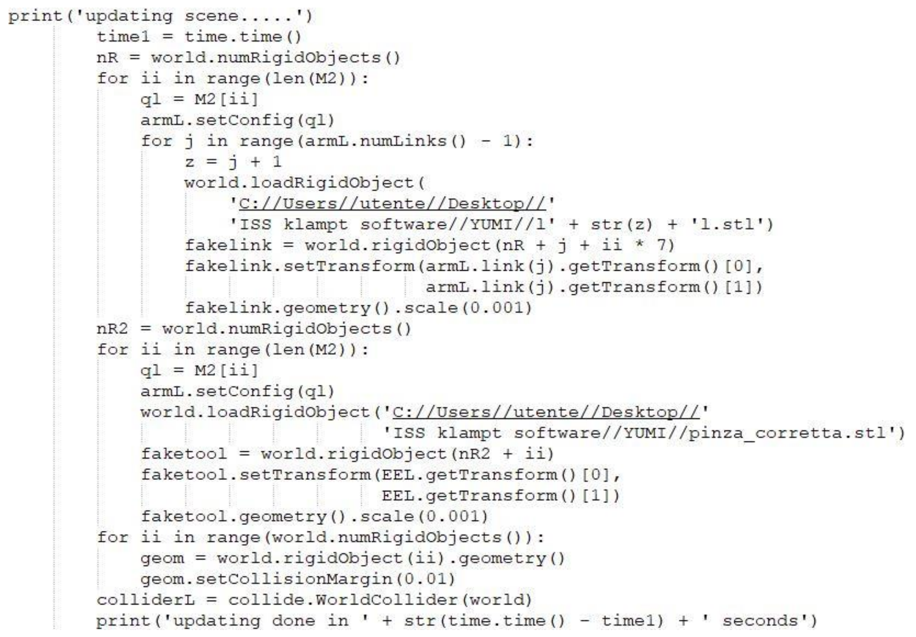


Figure 94: updating visualization

The updating procedure is computationally heavy and depends on the number of objects to upload and the number of copies to generate. In this test case is required to upload 8 links and copy them 15 times for each part of the trajectory. The time required for this operation is about 2,5 – 3 seconds both for the forward and the return trajectory.



```
print('updating scene.....')
        time1 = time.time()
        nR = world.numRigidObjects()
        for ii in range(len(M2)):
            ql = M2[ii]
            armL.setConfig(ql)
            for j in range(armL.numLinks() - 1):
                z = j + 1
                world.loadRigidObject(
                    'C://Users//utente//Desktop//'
                    'ISS klampt software//YUMI//l' + str(z) + 'l.stl')
                fakelink = world.rigidObject(nR + j + ii * 7)
                fakelink.setTransform(armL.link(j).getTransform()[0],
                                      armL.link(j).getTransform()[1])
                fakelink.geometry().scale(0.001)
        nR2 = world.numRigidObjects()
        for ii in range(len(M2)):
            ql = M2[ii]
            armL.setConfig(ql)
            world.loadRigidObject('C://Users//utente//Desktop//'
                                  'ISS klampt software//YUMI//pinza_corretta.stl')
            faketool = world.rigidObject(nR2 + ii)
            faketool.setTransform(EEL.getTransform()[0],
                                  EEL.getTransform()[1])
            faketool.geometry().scale(0.001)
        for ii in range(world.numRigidObjects()):
            geom = world.rigidObject(ii).geometry()
            geom.setCollisionMargin(0.01)
        colliderL = collide.WorldCollider(world)
        print('updating done in ' + str(time.time() - time1) + ' seconds')
```

Figure 95: updating procedure

Once the scene is updated, the planning procedure is repeated for the slave arm. At this point, the scene is more complex respect to the basic case and the required planning time increases respect to the example illustrated in chapter 6.

The sequence of operations in the final version of the software (including their typical time range) is the following one:

- Scene uploading (0,1 – 0,3 seconds; 4 scenes required);
- Distances definition and strategy selection (time negligible);

In case of coordinated motion planning:

- Grasp points definition master arm (time negligible);
- Grasp points definition slave arm (time negligible);
- Sequence of kinematic inversions master arm (time negligible);
- Sequence of kinematic inversions slave arm (time negligible);
- Forward planning master arm (1,5 – 2 seconds);
- Scene updating (2,5 – 3 seconds);
- Forward planning slave arm (6 – 7 seconds);
- Scene change (time negligible);
- Return planning master arm (1,5 – 2 seconds);
- Scene updating (2,5 – 3 seconds);
- Return planning slave arm (6 – 7 seconds);
- Communication protocol (2 – 3 seconds);

The resulting time for an overall coordinated planning procedure is about 26 – 29 seconds. This time is quite bigger respect to the standard planning time with alternated strategy. For this reason, the coordinated strategy may seem always disadvantageous for an industrial application but is important to consider that, using this strategy, the Yumi robot is capable of completing the task in half the time respect to the standard alternated approach. Another aspect to consider is that the overall planning procedure is actually implemented in Python. This programming language is usually slower respect to the C++ that has many more practical industrial applications.

Another important aspect to consider is that the planning time for the slave robot is always influenced by the increased complexity of the space, caused by the updating procedure responsible of the generation of some copies of the master robot. This means that, managing correctly the distance threshold in the strategy selection procedure, is possible to vary the behaviour of the planification:

- Increasing the threshold, the number of coordinated planning approaches should decrease but each planning procedure should be faster because of the increased distance between the targets. Of course, there will be a lot of couples planned with alternated trajectories; this will result in a higher time required to the robot to complete the task; Increasing the threshold also the probability of success of the coordinated approach should increase;
- Decreasing the threshold, the number of coordinated approaches increases; this means a reduction of time required to the robot to complete the task. The drawbacks are the increase of the planning time and the decrease of the percentage of success.

In order to evaluate these aspects an experimental test is performed, simulating 100 times the presence of two objects with random pose inside the bin. The test is performed first with a threshold of 0,1 meters, then with a threshold of 0,2 meters. Results are depicted in the following tab.

Threshold	Coordinated	Success coordinate	Mean plan time coordinate	alternated	Succ alternate	Mean plan time alternate
0,1 [m]	46 %	58,3 %	28,85 [s]	54 %	98,2 %	8,3 [s]
0,2 [m]	13 %	86,4 %	22,41 [s]	87 %	97,9 %	8,37 [s]

As it’s possible to see imposing a distance threshold of 0,1 meters, considering the dimension of the bin, about half of the planning attempts are performed with a coordinated approach. Due to the relatively small distance between the objects the mean planning time is quite higher and the percentage of success is not so high. In this case, considering also couples of objects planned with an alternate strategy the overall percentage of success of the planner is 79,8 %. Increasing thee distance threshold the number of coordinated planning approaches decreases but their percentage of success is higher, due to the more freedom in the configuration space conceded to the planner. In this case the overall percentage of success is 96,37 %.

6 CONCLUSIONS AND FUTURE DEVELOPMENTS

The obtained results have showed clearly how the software have the potentialities to generate successful planning operations for a redundant double-armed collaborative robot, with application in the bin picking field. The way in which this operation was performed has never tried before: generate valid, free-of-collisions, coordinated trajectories in order to allow the Yumi robot of grasping two random-oriented objects from a bin at the same time, opens new possibilities, both in the field of vision systems for automatic motion planning and in the sector of collaborative robotics. A collaborative robot that should not be programmed to execute always the same repetitive movements and does not need to be supplied with workpieces always positioned and oriented in the same way, is the real future of the human-robot collaboration we have mentioned at the beginning of this thesis. The potential future of the manufacturing industry is a robot capable of safely interacting with human beings and capable of "choosing" how to move in order to assist the labourer in assembling and manufacturing tasks, taking charge of the heaviest and tiring operations , optimizing productivity and efficiency and reducing possibility of errors and repeatability of the human operations, increasing the goodness of the working conditions.

Of course, the main problem at this point of the development is the computational time. The software is not yet implementable in real industrial applications due to the slowness of the planning phase. In our opinion, the main reason of this high computational time is the programming language in which the code is written at this point of the development. Python is easy to learn and perfect for software prototyping due to its easy accessibility, syntax and possibility to integrate additional functionalities without complex set-upping and debugging procedures.

Some easy tests, based on standard industrial robots, are already in progress, in which the basic functionalities of the C++ version of the klamp't library, such as the kinematic inversion module and the planning one, are implemented for a simplified path planning procedure in a bin picking-like environment.

These tests are confirming our ideas: functions written and debugged in C++ are way faster than correspondent Python scripts. The result is a complete approach and extraction trajectories computation in less than 1 second.

The expectation is of introducing all the developed functionalities for Yumi applications in the next months, including:

- managing the double-armed redundant structure;
- introducing an automatic communication with the scanner;
- generate a graphical tool for composing a set of grasp points on the CAD model of the workpiece and iterate a sequence of valid approach and return points until a full valid trajectory is found;
- introducing functionalities for automatic planning strategy choice;
- developing the procedure for the coordinated motion plan introduced in the last chapter, exploring some possible, time-saving alternatives;
- introducing the TCP-IP communication with the real robot.

The hope is of decreasing the time required for a coordinated double-armed bin picking planning below 5 seconds and time for alternated bin picking planning below 2 seconds. Under these times, the software can become really competitive from an industrial point of view, allowing the implementation of Yumi's bin picking operation in several industrial context: in fact, the planning time will be hidden by the time required to the human labourer or to the robot for performing the assembling or manufacturing operations; In this way, when one operation is finished, the next trajectories will be already computed and the overall result will be a continuous sequence of real-time planned movements for the robot and the overall task will never suffer of idle times due to planning computations.

7 BIBLIOGRAPHY

- “Robotic systems design”, H. Giberti, course slides, 2018;
- “Robotics, Modelling, planning and control”, B. Siciliano, L. Sciavicco, L.Villani, book, 2009;
- “Planning Algorithms”, S. LaValle, book, 2006;
- “Robot motion planning”, J. Latombe, book, 1991;
- “Sviluppo di algoritmi innovative e soluzioni flessibili per la generazione automatica di traiettorie robot per applicazioni industriali”, A. Quattrini, master thesis, 2013;
- “A single query Bi-Directional Probabilistic roadmap planner with lazy collision checking”, J. Latombe, paper;
- “Roadmap composition for multi-arm systems path planning”, J. Cortes, T. Simeon, paper, 2009;
- “Centralized path planning for multiple robots: optimal decoupling into sequential plans”, J. Van den Berg, J. Snoeyink, paper;
- “Sampling-based, multi-robot motion planning”, Z. Yan, N. Jouandeau, paper 2013;
- “On delaying collision checking in PRM planning”, G. Sanchez, J. Latombe, paper;
- “The use of collaborative robots in manufacturing”, A. Calitz, P. Poisat, M. Cullen, in SA Journal of Human Resource Management, 2017 [1].
- “Human-robot collaboration in industry”, A. Vysocky, P. Novak, paper, 2016 [2];
- “Automatic path planning of industrial robots comparing sampling-based and computational intelligence methods”, L. Larsen, J. Kim, M. Kupke, A. Schuster, in 27th international conference on flexible automation and intelligent manufacturing, 2017 [3];
- “Recent progress on programming methods for industrial robots”, Z. Pan, J. Polden, N. Larkin, S. van Duin, J. Norrish, paper, 2012;