University of Padova

Department of Information Engineering

Master's Degree in Control System Engineering

Master's Degree Thesis

TCP optimization methods for industrial robots

Supervisor: Prof. Ruggero Carli
Company's Supervisor: Ing. Luca Benvegnù

Author: Edoardo Pastorello
2021437

October 17, 2022

A.Y. 2021/22

Abstract

During the last decades, robot manipulators usage in industrial context has grown exponentially. On the other hand, research has tried to improve their geometric parameters in order to minimize energy requirements for motions, or generally adapt robots' characteristics to the specific task. In this work, a new aspect, rarely found in literature, is analyzed: tool center point optimization (i.e. the final tool pose). Thanks to the resources granted by the hosting company Euclid Labs, an optimal position seeking system have been implemented, working with both brand new algorithms and more common ones, already known in research fields. The purposes of this work are numerous: from the minimization of the time required to perform the motion, to the management of the working range of the axes in order to make the robot operate far from its limits and from singularity, besides the maximization of the number of picked objects in a random bin picking scenario. After the first part dealing with the theoretical methodology used, the new functionality implemented in the company's proprietary simulator will be illustrated, developed in C#, with discussion of the results.

Sommario

L'utilizzo di robot manipolatori in contesti industriali è cresciuto esponenzialmente negli ultimi decenni. Parallelamente, la ricerca ha sempre tentato di migliorarne i parametri geometrici e costruttivi al fine di minimizzare l'energia spesa nel movimento, o adattare al meglio le capacità del robot al task. In questo lavoro, viene analizzato un aspetto che raramente è presente in letteratura: l'ottimizzazione del tool center point (ovvero del posizionamento del tool terminale). Grazie alle risorse messe a disposizione dall'azienda Euclid Labs, è stato possibile implementare un sistema di ricerca dell'ottimo sia tramite algoritmi di nuova concezione, sia con altri più noti. Gli scopi sono molteplici: dalla minimizzazione del tempo necessario al movimento, alla gestione del range di lavoro degli assi in modo che il robot lavori lontano dai suoi limiti e dalla singolarità, oltre che alla massimizzazione del numero di oggetti prendibili in scenari quali il random bin picking. Dopo la parte votata all'illustrazione della metodologia teorica, verrà illustrato il simulatore proprietario dell'azienda con la nuova funzionalità sviluppata in C#, con discussione dei risultati.

Contents

Chapter 1

Introduction

In industrial environments, robot manipulators are becoming more and more spread. One of the main advantages for factories is the ability to perform highly repetitive tasks with particular accuracy or velocity requirements, even in dangerous situations.

When deploying a cell with an industrial robot, the main effort is to design the overall layout in order to optimize some metrics, for instance energy requirements. Indeed, robots are energy-intensive devices, hence building more efficient ones, as well as reducing components wear may be the key to lower the impact of industries on the planet and the expenses of companies (for further reading on the topic, refer to [1, 2]).

But minimization of robots' energy consumption is not the only struggle of researchers: working in industrial environments like assembly lines, it is necessary to guarantee the lowest cycle time in order not to constitute a bottleneck for production.

The aspects that can be analyzed when optimizing a robot for a given task are many: in [3], for instance, an algorithm able to properly design the links and the joints of the manipulator is developed. It considers as main target a manipulability measure based on the determinant of the Jacobian matrix of the robot. Indeed, the Jacobian stores important information about the robot current configuration, like the closeness to singularity. Other techniques like the Grid Method proposed in [4] can be used to design a minimal complexity structure while keeping high dexterity and satisfying dimensional and joint angles constraints.

Supposing to have a robot that must track a trajectory, once designed the mechanical structure and the controller, a good way to minimize energy consumption is to act on the trajectory itself by increasing or decreasing the velocity of the tool during some parts of the motion. In [5], these

observations are exploited to analyze how thermal energy dissipation decreases when trajectory duration increases. Another paper that also keeps into account the losses of electric motors and inverters in designing the best trajectory is [6], that shows a reduction up to 10% in electrical energy consumption by solving an optimization problem. In addition, being manipulators often redundant with respect to the task, another option would be to choose the proper joints' configuration to set the link in the best possible pose to minimize power consumption [7] or, acting directly to the planning level, the trajectory itself can be designed with these aims in mind [8].

Another parameter to modify to obtain better performance is base location: robots are typically fixed in a particular pose (but there is the possibility to install it on specific sliding guides, like fig. 1.1) and, given that their task is always the same, it makes sense to compute the optimal relative position and orientation of robot and task. Actually, a good base placement acts on many aspects of the task: for instance, reachability and maximum dexterity within the task area, as illustrated by [9, 10], or the stiffness of the robot arm in case of machining tasks, since higher rigidity of the arm implies better quality products [11]. Another target of base placement may be the maximization of the manipulator velocity ratio (MVR), that is the square root of the ratio between the norm of the end-effector velocity and the norm of joints' velocity. The bigger the MVR, the slower the joints have to move to achieve an instantaneous motion of the tool. A good dissertation on this is given by [12, 13].

Figure 1.1: A slide for robot manipulators: the KUKA KL 4000

1.1 Tool Center Point optimization

Before investigating the real aim of this work, it is necessary to provide a definition of the Tool Center Point. Yaskawa official site [14] states that "*The Tool Center Point (TCP) defines the tip of the current tool as defined relative to the tool flange. For example, for a welding robot, the TCP will generally be defined at the tip of the welding gun*". See fig. 1.2.

A bare 6R PUMA-like arm manipulator is just a sequence of revolute joints connected by rigid links: the last joint terminates with a "flange", that is nothing but a mounting point for the tool that, being a rigid body, implies a fixed rototranslation to pass from the flange reference frame to the TCP frame, i.e., its tip.

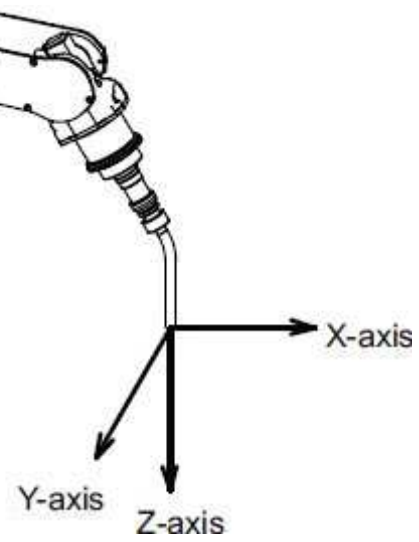


Figure 1.2: A representation of the TCP reference frame, drawn from Yaskawa official site

As stated in [15], that is the real starting point of this work, a TCP pose optimization algorithm has not been proposed in the literature yet. Indeed, by considering for instance the higher torque produced by the last joints to lift an object that is grabbed far from the flange, it may look counterintuitive to modify the TCP pose to optimize velocity, energy, or other measures. On the other hand, with respect to base placement that is most of the times limited to a translation, a tool can be rotated in almost any pose just by changing a support, and this may positively modify the reachable and dexterous workspace of the robot, and improve many other metrics. To justify the importance of a well-placed TCP, arc welding robots may be considered: typically, these welding torches are curved and,

in many applications, the nozzle is put exactly on the line exiting the flange perpendicularly. This configuration allows the robot to weld a sharp angle without moving nothing but the last joint, that is typically the fastest one. So, it is immediate to conclude that, at least in such points, motion velocity is optimized. As well as base placement, a better TCP pose can be computed with different aims in mind, and for various tasks. This is the purpose of this work: to provide a set of functionalities to retrieve an optimal pose of the tooltip (where "optimal" depends on the current application).

1.2 Report structure

This report is organized as follows: In chapter 2 the current solutions already available to optimize the TCP pose are reviewed, the target of the hosting company is presented, as well as requirements and equipment for the thesis. In chapter 3 some preliminary mathematical and physical considerations regarding joints range and velocity are reported, and genetic and pattern search algorithms are introduced. Chapter 4 presents and comments the actual implementation of the algorithms in the software, followed by the overview of the entire application with all the details that the user may set. Chapter 5 reports a thorough discussion of the results on many sample trajectories and scenarios. In chapter 6, final considerations about the effectiveness of the work are summarized.

Chapter 2

State of the art and problem formulation

2.1 The problem of changing the TCP

As already stated, the reviewed literature does not present any solution that is meant to find an optimal placement of the tool, and hence a good pose of the TCP, but one [15]. In this paper, a preliminar optimal base placement is solved for a robot that has to track a set of continuous paths.

Then, optimal TCP placement is found, based on an objective function made of multiple terms: the number of feasible inverse kinematics solutions for every path point, the number of feasible *continuous* solutions for each tool path, a term that minimizes the joint motion between consecutive path points and, finally, the distance of each joint angle from its mechanical limits. This function is not continuously differentiable, hence the author makes use of a MATLAB optimization algorithm that relies on an approximation of this function. After the optimal pose of the tool is found, the base placement is updated in order to have an optimized base-TCP pair.

The focus of that work is not specifically on the results, since the validation of the proposed method is qualitative, but on the fact that it is now sufficient to adapt the TCP of the robot to optimize the robot's performance, instead of readapting the robot work cell. Indeed, without an opportune robot slide, repositioning the robot may be time-consuming and expensive, while now a simple mechanical adapter may be designed to accomodate the new TCP.

Of course, designing a device to be placed between flange and tool that may continuously modify the tool pose is far from being physically feasible.

Indeed, this would increase the weight that the robot has to move, and for particularly heavy objects this device would be very bulky and hefty, basically frustrating all the attempts to find a good TCP. Moreover, the farther the TCP from the flange, the higher the torque produced by the robot joints to move the same object.

Hence, the target of TCP optimization is to find and design a proper tool that optimizes some metrics with respect to a repeated task. An old tool can be then adapted whenever the robotic cell layout changes. Since robots are mainly used to perform repetitive tasks with high precision and low cycle-time, the motion to be performed may be the same for a long time, and hence there is no reason to change an optimal TCP until the task changes (because of product redesign, change or update). Therefore, a solution to this problem may still be applicable to many scenarios.

2.2 Random bin picking in Euclid Labs

Euclid Labs, the hosting company, is concerned with robotics and computer vision-based automation solutions. One of its main products is MoonFlower Blue, a software for random bin picking that, if provided with adequate scanning sensors, is able to find randomly placed objects, design a collision-free trajectory for a specific robot, and make it pick the item.

Typically, bin picking-adapted tools are like the one depicted in fig. 2.1: in this case, the tool is a magnet, but it is not directly installed on the flange. Indeed, it is located far from the axis of the flange. The reason for this is that bins and pallets are typically close to the robot itself. Thus, to pick an object, it is very likely that the robot would assume a configuration very close to singularity. Using this kind of offset for the TCP, instead, even if there is no guarantee, helps to avoid this.

2.3 Requirements

The solution to be developed must satisfy the following requirements:

- A C# program must be developed, that extends the existing random bin picking software provided by the company.
- The optimization variable is the TCP pose (or the tool base pose) with respect to the flange of the robot. The admissible range can be set by the user.

- The target is either to minimize the time required to perform a motion, or to make the joints of the robot work as far as possible from joints limits or singular configurations.
- Also, the problem of maximizing the number of pickable objects by changing the TCP should be addressed.
- Reaching the optimal pose is not necessary: a sub-optimal solution is sufficient, given that some of the metrics above improve.

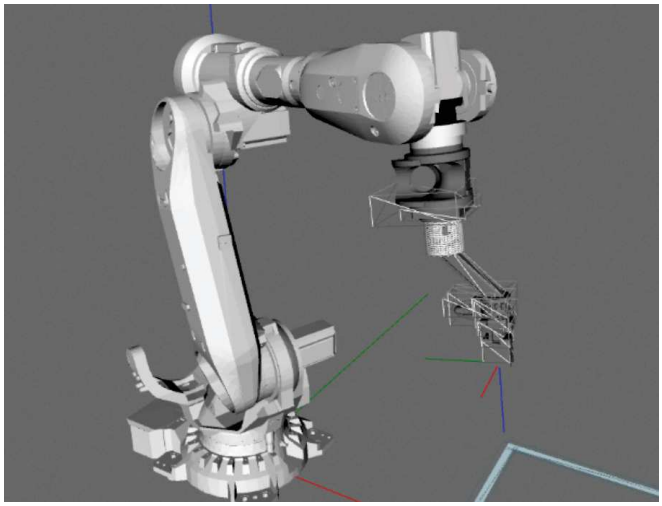
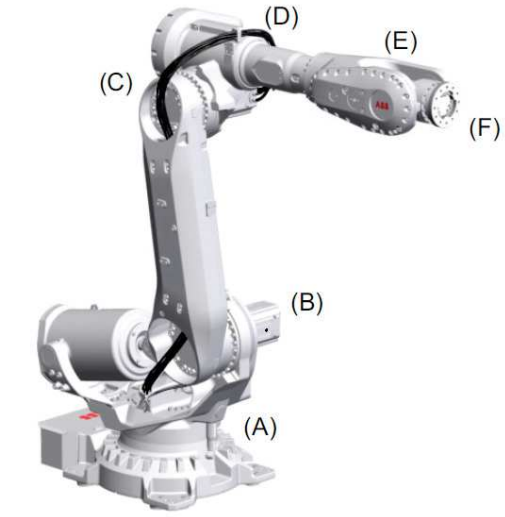


Figure 2.1: A screenshot of MoonFlower Blue simulator: an ABB IRB6700 robot is shown, with a bin picking tool

2.4 Test equipment

To develop and debug the code to solve the problem, beside the company's simulator, a complete model of the robot IRB6700-235/2.65 produced by ABB have been used. In fig. 2.2 the robot's model is shown.



xx1300000244

Pos	Description	Pos	Description
A	Axis 1	B	Axis 2
C	Axis 3	D	Axis 4
E	Axis 5	F	Axis 6

Figure 2.2: An ABB IRB6700 robot with the 6 joints highlighted. Image taken from the product specification manual

Chapter 3

Theory and intuitions

3.1 The manipulator velocity ratio

As already stated while reviewing the literature [12, 13], when it comes to optimizing the velocity of a robot, one of the first thoughts must be the manipulator velocity ratio (MVR). To understand what it is, how it works and why using it or not, it is necessary to recall some basic concepts of robotics.

A 6R PUMA-like robot manipulator is a sequence of six open-chain joints connecting links. A manipulator is meant to drive the single joints all together in order to produce a specific motion of the final tool, with its own velocity and acceleration profile. Some useful indices to assess the performance of a manipulator during a motion are given by velocity and force manipulability ellipsoids. In the following, only the former is going to be considered.

Consider the joints configuration vector $q = [q_1 \quad q_2 \quad q_3 \quad q_4 \quad q_5 \quad q_6]^\top$ and its first derivative $\dot{q}$. Introducing the Jacobian $J(q)$ and the fundamental inverse kinematics relation $\dot{q} = J^\dagger(q)v_e$, where v_e is the velocity of the tool (i.e., the TCP), it is possible to compute how a change is the joints configuration vector impacts the produced velocity. Starting from the 6-dimensional sphere $\dot{q}^\top\dot{q} = 1$ the following derivation holds:

$$\begin{aligned} \dot{q}^\top\dot{q} &= 1 \\ v_e^\top(J^{\dagger^\top}(q)J^\dagger(q))v_e &= 1 \\ v_e^\top(J(q)J^\top(q))^{-1}v_e &= 1 \end{aligned} \tag{3.1}$$

The last equation in (3.1) encodes the set of points lying on an ellipsoid in the end-effector velocity space. Hence:

$$\dot{q}^\top \dot{q} = v_e^\top (J(q)J^\top(q))^{-1} v_e \tag{3.2}$$

And in turn, we can define the transformation ratio for the velocity manipulability ellipsoid:

$$\alpha(q) = \frac{1}{\sqrt{u^\top (J(q)J^\top(q))^{-1} u}} \tag{3.3}$$

Where u is a unit vector in the direction of the velocity to be evaluated. The higher this ratio, also called manipulator velocity ratio, the lower the required joints velocities to produce the motion.

An intuitive idea would be to optimize the pose of the tool by performing gradient ascent on this ratio. This method requires to compute the gradient of the MVR ($\alpha(q)$) with respect to the six variables of the pose of the TCP ($x_{tool} = [x \quad y \quad z \quad \phi \quad \theta \quad \psi]^\top$, where the first entries are the offset with respect to the flange frame, and the last ones the rotation with ZYX convention). This procedure comes with a series of issues to be dealt with:

- The robot may pass through singularity. In such configuration, the Jacobian matrix is not invertible, and the required joints velocity in some directions may go to ∞ even for very small end-effector velocity. This makes the MVR not continuous nor differentiable.

- The gradient of the MVR depends on the gradient of the Jacobian with respect to the optimization variable: $\frac{\partial J(q)}{\partial x_{tool}}$. However, also the configuration depends on the TCP pose, since changing the offset and/or the rotation of the TCP affects the final link: this implies that, whenever a change of x_{tool} occurs, some Inverse Kinematics algorithm must be run to retrieve q, that changes also the Jacobian $J(q)$. Moreover, one has to study how the IK algorithm is affected by a change of the TCP, since, by the chain rule:

  $$\frac{\partial J(q)}{\partial x_{tool}} = \frac{\partial J(q)}{\partial q} \frac{\partial q}{\partial x_{tool}}$$

  Indeed, it is not possible to change the TCP without reconfiguring the robot's joints, otherwise the final pose of the TCP would not follow the right trajectory.

- Gradient Ascent method is just a steepest-direction following method. Being the structure of a manipulator highly non-linear and subject to singularity it's easy to conclude that many starting points must

be selected and, since x_{tool} has 6 entries, it is necessary to instantiate a high number of evenly spread starts.

In order to develop a fast search algorithm without complex computations and without dealing with the particular implementation of IK algorithms, an easier and faster way to evaluate the cycle time of a given pose of the TCP has been developed. In particular, the trajectory is split in multiple sub-trajectories that, in principle, should require the same time to be tracked. Then, the TCP is moved and the minimum time required to perform each piece of trajectory is recorded (that is determined by the joint that, considering its maximum velocity, needs the greater amount of time). The tool is finally modified trying to lower this time.

The final algorithm with all details will be shown and discussed later on (section 4.1).

3.2 How to evaluate the joints range

When the task to be performed is random bin picking, velocity is not the first optimization target to consider. As the name suggests, the environment is randomic, and hence there is no fixed trajectory. Conversely, the robot has a sequence of points to be reached: there, it has to pick an item, find a way out and successfully remove the piece. Intuitively, if the object is inside the reachable and dexterous workspace of the manipulator, no problem arises. However, placing it in this range is just a matter of optimal base placement, that is preliminar to optimal TCP placement.

The second metrics to consider are instead joints ranges limits: the purpose of this optimization is to make all the joints work as close as possible to the center of their range, except for the links that may cause singularity.

Still, the question to be answered is: how is it possible to weigh the distance of a joint from its limit? The solution is quite obvious and consists in assigning a score in the interval $[0; 1]$ to the angle of each single robot axis. Suppose an axis can sweep an angle of 300°, and its central position is 0°. Some possibilities to assign a score to the joint configuration are depicted in fig. 3.1. The picture shows three possible scoring functions: the red one is linear, the orange one gaussian (where $\mu = 0$ and $\sigma = \frac{range}{7}$), and the blue one is a generalized normal distribution (with $\alpha = \frac{\sqrt{2} \cdot range}{6}$ and $\beta = 2.7$. For further details please refer to appendix A).

It is possible to notice that the linear one is not the best choice. Indeed, when the joint angle moves away from the center, its score begins

immediately to fall proportionally to the distance. In addition, a shift of the same amount in the angle produces the same "fall" in the score, both if the angle is close to 0° and also if it is further away. A good alternative may be the gaussian scoring function (with peak = 1): in this case, if the joint is close to its center, then the score is higher than before, and it falls steeply if it gets farther. Values close to the limits are assigned a score that is just a little greater than 0. This function has a big advantage: a worsening in the angle produces a change in the score that is not constant, but it depends on where the joint is: if close to the center or to the limits, it does not change evidently, otherwise it falls rapidly. Obviously, one may choose a different standard deviation for the gaussian to make it thinner or not.

Another interesting possibility is instead a generalization of the gaussian scoring function: the generalized normal distribution. It forms a wider peak in the center that allows to extend the range in which the score is close to 1, and to make the falling section even steeper, while keeping the score of the range limits equal to 0. This was the chosen function to implement the algorithm, given also the flexibility that its parameters α and β provide.

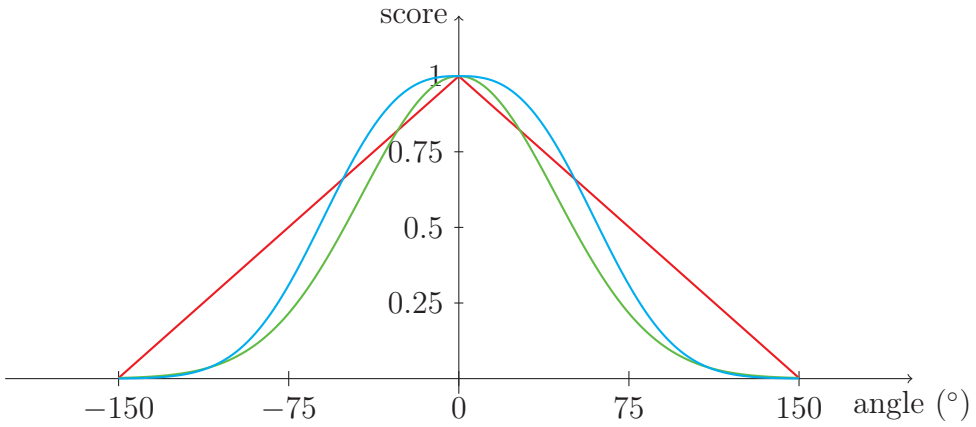


Figure 3.1: Some possible ways to assign a score to the joint range. The red function is linear, the green one a gaussian and the cyan one a generalized gaussian

In case a particular joint may cause singularity, it is necessary to treat this point as a limit, by changing the scoring function as proposed by figure 3.2. In particular, the peaks of the two resulting distribution may also be changed in order to make one portion of the range more favorable than the others (and hence to force, for instance, an "elbow-up" configuration).

Basically, the algorithm computes the joints configuration that the robot needs to pick the objects, makes use of this scoring function to assign a score to the configuration of the specific joint, and finally takes the score of the joint with worst angular displacement (close to its limits/to singularities), and the average scores as metrics to improve. Again, the particular implementation of this algorithm is going to be discussed in the following sections.

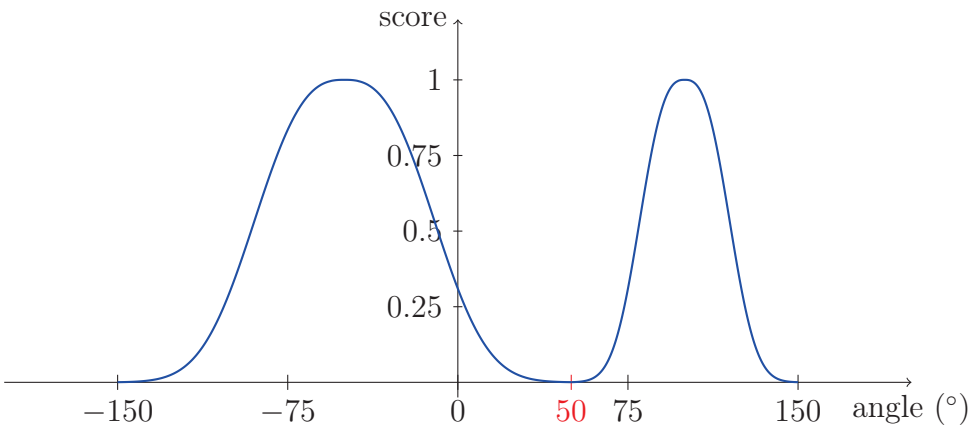


Figure 3.2: A generalized normal distribution scoring function in case of singular configuration at 50°. in singularities, the score must fall to 0, so two distributions are created and each one works in a different section of the domain

3.3 Genetic algorithms

While developing the intuitions presented above in real code, the necessity to compare performance, time-requirements and results with other methods arose. Among the different choices, also present in the reviewed literature, both genetic and pattern search algorithm approaches have been chosen. In this section, the general idea behind the former class of algorithms is going to be presented, leaving the actual implementation to be treated later on.

As explained in many works, like [16], a genetic algorithm (GA) takes inspiration from Darwin's evolution theory: the main idea is that only the strongest elements in a population shall be able to reproduce and spread their genes.

In a few words, a GA has the following basic scheme: an initial population is randomly selected and evaluated with an objective/fitness function. The more an individual is close to the optimal (being it the minimum or the maximum of the objective function, or even a specific value), the higher its probability to reproduce. Then, a bunch of elements in the initial population is selected according to this probability, imitating the effects of natural selection, and they are randomly coupled with each other. Coupling means randomly mixing the genetic code like a so called *genetic crossover*. Say that each individual has n genes of types A, B, $\ldots$, N: the children of a couple will have the same sequence, where each gene is randomly selected from one of their parents. This is called uniform crossover, but many alternatives are available (like the one-point crossover, that randomly selects a position in the genetic code and all the genes before that point are inherited from parent 1, all the others from parent 2). Mutation is also allowed, and consists of a random mutation of one or more genes. The new born population replaces the old one and reproduction continues.

Unfortunately, despite they are known in the scientific community, there is no proof or theoretical analysis regarding their convergence. On the other hand, heuristics often show good results, and they are adaptable to many different scenarios: indeed, the choice of objective function, crossover, type of mutation and replacement of old population may lead to very different algorithms. In fig. 3.3 a scheme of a simple GA is shown and discussed.

3.4 Pattern search algorithms

Another derivative-free optimization method used in this work is the so-called pattern search algorithm (PSA) [17]. Basically, this method finds (at least) a sub-optimal solution by looking for the neighbors of the current point. In a few words, the algorithm works as follows:

1. Given a starting point x in the domain (for simplicity, say $\Re$), an objective function f to minimize and a pattern k, evaluate $f(x+k) = f_1$ and $f(x-k) = f_2$.

2. If $f_1 < f(x)$ and $f_1 < f_2$, assign $x = x + k$. If $f_2 < f(x)$ and $f_2 < f_1$, assign $x = x - k$. Otherwise, if $f(x) \leq f_1, f_2$, reduce the pattern ($k = k/d, d > 1$).

3. Restart from 1. Repeat until convergence or other termination condition (e.g.: $k \leq k_{min}$).

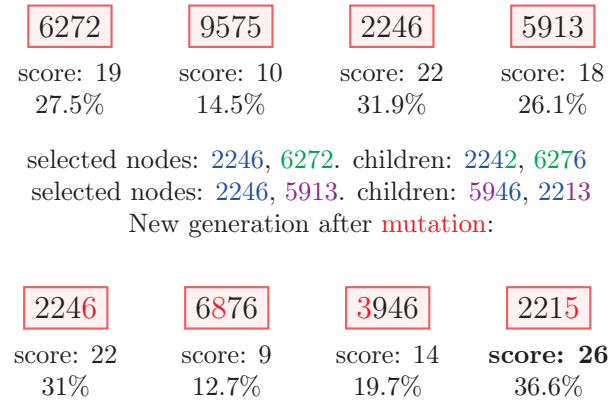


Figure 3.3: An iteration of a genetic algorithm to find the decimal number 1111. Each gene represents a digit from 1 to 9, and the value of the fitness function is: $score = (9 - digit_1) + (9 - digit_2) + (9 - digit_3) + (9 - digit_4)$ and has to be maximized. The probability of reproduction is the score divided by the sum of the scores (e.g.: for the first individual in the first population it is $\frac{19}{(19+10+22+18)}$). Each couple generates two sons that, after mutation, make up the second generation

As said, it is a very simple algorithm used to look for a minimum (or maximum) of an objective function in a specific domain without evaluating its derivatives. This method may be extended to n-dimensional spaces (in this work the space will be 6-dimensional, dealing with rotations and translations).

Actually, the implementation of this method in the project is different, since there is no shrinking pattern and it is necessary to deal with a domain that requires different step-sizes (namely one for rotations and another one for translations).

This category of optimization methods have been used mainly for optimizing the number of objects in a bin that the robot can pick. However, it may be interesting to compare it also with the specifically tailored algorithms for velocity and range optimization.

Chapter 4

Development of the algorithms

In this chapter, the specific algorithms developed to solve the problem are explained, together with further considerations on the choices that the author made.

4.1 Velocity optimization

As already stated in section 3.1, velocity optimization may take into account the MVR and try to apply gradient descent to find a good minimum. However, considering the complexity, non-differentiability and non-linearity of the objective function, as well as all the considerations about singularity previously presented, a new, simpler approach should be formulated.

In particular, this new method starts from the fact that robot manipulators typically have different motors to drive each single joint. This implies that each axis has different velocity and acceleration profiles, as well as maximum torque and many other characteristics. At the same time, relative positioning of two or more joints may influence each other's limits, also for safety reasons. The manufacturers of industrial robots set some velocity limits, like those reported in table 4.1.

Table 4.1: Maximum velocities for each joint of the test robot, drawn from the official manual by ABB

Robot type	**axis 1**	**axis 2**	**axis 3**	**axis 4**	**axis 5**	**axis 6**
IRB 6700-235/2.65	$100°/s$	$90°/s$	$90°/s$	$170°/s$	$120°/s$	$190°/s$

Given a simple rotational motion of one joint, to get the time needed to sweep the specified angle it is necessary to divide the angular displacement

by its velocity. Of course, neglecting the additional torque produced by the offset of the tool with respect to the flange (one of the allowed starting relaxations) and ignoring the acceleration profile to keep calculations as simple as possible, the result using data in table 4.1 is an indication of the minimum time that each joint needs to perform that motion, hence it suffices to take the greatest of the times as a bottleneck. To make it simple, a vector of weights $W = [1/v_1, 1/v_2, \ldots, 1/v_6]$ may store the multipliers to use for each entry of the joints' velocities vector $\dot{q} = [\dot{q}_1, \dot{q}_2, \ldots, \dot{q}_6]$. Hence, the metrics to be used are: $max(W_1 \cdot \dot{q}_1, W_2 \cdot \dot{q}_2, \ldots, W_6 \cdot \dot{q}_6)$.

However, this only returns the necessary time for a given PTP (point-to-point) motion. If a complex trajectory must be tracked, it is necessary to split it into multiple sub-motions. In this work, the assumption is that the points of the trajectory must be taken such that the desired time from point to point is the same (thus, for instance, a circular trajectory with constant velocity should have equally spaced points). With this constraint, it is just necessary to take the maximum among the times of the PTP motions in which the trajectory is divided. Hence, the TCP/tool pose that minimizes this maximum will be the target. In theory, all PTP motions other than the one taken as maximum should be below the limit.

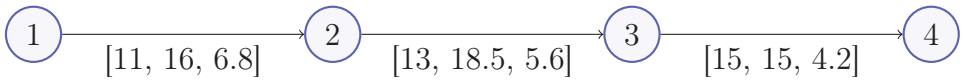
In the following, a pseudocode illustrating the algorithm in detail is reported:

```
% Initialization
Generate test TRAJECTORY as sequence of waypoints (position + rotation);
Generate 50 initial TCP poses inside allowed range (INITIALPOSES array);
For each POSE in INITIALPOSES:
while POSE is not feasible for TRAJECTORY:
|_    choose a new POSE randomly;

% Optimization
FinalPoses = [];
For each POSE in INITIALPOSES:
|    MaxScore = MAX(EvaluatePoseVelocity(POSE));
|    Choose randomly ENTRY = 1,2,3,4,5,6;
|    Choose randomly SIGN = -1,+1;
|    CONTINUEIMPROVE = true;
|    NumOfIterationsNoImprove = 0;
|    NumOfIterationsImprove = 0;
|    while(CONTINUEIMPROVE):
|    |    NewPOSE = ChangePose(POSE, ENTRY, SIGN);
|    |    NewMaxScore = MAX(EvaluatePoseVelocity(NewPOSE));
|    |    if (NewMaxScore<MaxScore):
|    |    |    POSE = NewPOSE;
|    |    |    MaxScore=NewMaxScore
|    |    |    NumOfIterationsNoImprove = 0;
```

```
|    |    |    NumOfIterationsImprove += 1;
|    |    else:
|    |    |    NumOfIterationsNoImprove += 1;
|    |    |    NumOfIterationsImprove = 0;
|    |    |    Choose randomly ENTRY = 1,2,3,4,5,6;
|    |    |_   Choose randomly SIGN = -1,+1;
|    |
|    |    if (NumOfIterationsImprove > 25):
|    |    |    Choose randomly ENTRY = 1,2,3,4,5,6;
|    |    |_   Choose randomly SIGN = -1,+1;
|    |    if (NumOfIterationsNoImprove > 40):
|    |_   |_    CONTINUEIMPROVE = false;
|
|    If POSE collides with objects in the scene during trajectory:
|    |    reset POSE;
|    |    If POSE have been reset > 3 times:
|    |    |    neglect POSE
|    |    else:
|    |    |_   restart improving;
|    else:
|_   |_   FinalPoses.Append(POSE)

% Methods
def EvaluatePoseVelocity(TCPpose):
Score = [];
For each POINT in TRAJECTORY (except last one):
|    NEXT_POINT = POINT + 1;
|    Set the robot to reach POINT with TCPpose. Save config Q1;
|    If unreachable or singular:
|    |_    return [10000 10000 ... 10000];
|    Set the robot to reach NEXT_POINT with TCPpose. Save config Q2;
|    If unreachable or singular:
|    |_    return [10000 10000 ... 10000];
|    Qdot = Q2 - Q1;
|    W = [1/100  1/90  1/90  1/170  1/120  1/190];
|    MaxTime = MAX(W_i*Qdot_i);
|_   Score.Append(MaxTime);
return Score;

def ChangePose(Pose, entry, sign):
Switch entry:
|    Case (1): Pose.x += sign*10mm;
|    Case (2): Pose.y += sign*10mm;
|    Case (3): Pose.z += sign*10mm;
|    Case (4): Pose.roll += sign*(PI/32);
|    Case (5): Pose.pitch += sign*(PI/32);
|_   Case (6): Pose.yaw += sign*(PI/32);
if Pose outside range:
|    Choose randomly ENTRY = 1,2,3,4,5,6;
```

```
73 |    Choose randomly SIGN = -1,+1;
74 |_   ChangePose(Pose, ENTRY, SIGN);
75 return Pose;
```

In this pseudocode it's possible to notice that the initial poses are selected randomly from the admissible range. To evaluate a pose (function `EvaluatePoseVelocity`), the algorithm just computes the difference between the robot configuration of one point in the trajectory and the following one, and weighs it by a vector `W` that is the inverse of the joints velocity to obtain the minimum time needed by every joint to track that portion of trajectory, as a vector. The returned score is the array of all the maximum times, one entry for each point-to-point segment of the trajectory. The main part of the algorithm (line 11) then takes the maximum of this vector (i.e., the time needed by the slowest joint on the slowest segment), and tries to change the pose in order to decrease this value. In case of unfeasibility or singularity, the returned score is an array of 10000. It's worth noting that this optimization criterion works in case the designed trajectory must be tracked from point to point by keeping a constant TCP velocity, since the bottleneck is the slowest segment. In case this is no more true and just the total time is needed, substitute lines 11 and 19 with `SUM(...)`, instead of `MAX(...)`. In the following, a practical example for a 4-points trajectory, 3-axes robot:



$W_i \cdot \dot{q}_i$: [1.1, 3.2, 3.4]; [1.3, 3.7, 2.8]; [1.5, 3.0, 2.1]

Scores: [3.4, 3.7, 3.0]

MAX: 3.7

Figure 4.1: A practical example of the velocity optimization algorithm with a 3-axes robot. Underneath the arrows, the angle swept by each joint is reported. The weights vector W is $[0.1, 0.2, 0.5]$ (hence joint 1 is the fastest). The scores are just the maxima of each resulting weighted vector, and the final time is the maximum among all maxima

Once the starting pose has been evaluated, it is randomly changed: a direction of improvement (1 to 6, corresponding to the entries of the TCP pose vector) is chosen, and it is increased or decreased by 1cm in case of offset, or $\frac{\pi}{32}$rad in case of orientation, recalling that the new pose must lie inside the admissible range. The new pose is then evaluated,

and if the maximum of the score vector, that is actually the minimum time necessary, is lower, the new pose substitutes the starting pose and optimization continues. If the poses improves in one direction more than 25 iterations, a new direction is chosen to avoid to stick to the old one (a procedure that recalls ϵ-greedy policies in reinforcement learning, even if now there is no probability of choosing the "best" action). If, instead, there is no improvement, the direction is again randomly changed.

This algorithm may look very similar to the already presented family of derivative-free pattern search algorithm. The difference is basically the choice of the candidate improvement direction: since the optimizer is dealing with a robot that may be subject to singularity, range limits, the search space lies in a 6-dimensional real space and many other issues, the function to be optimized may be very complex. Hence, choosing the improvement direction only checking which adjacent pose has the best score, while maybe another, apparently worse one would lead to the optimum, may limit the possibilities. In this case, the algorithm also tries less "attractive" directions. Moreover, there is no necessity to use shrinking patterns, since a change of less than 1cm or $\frac{\pi}{32}$rad would be irrelevant for many practical purposes. Anyway, an implementation of a PSA will be shown and discussed later on.

Lastly, a collision check is performed: if the new TCP pose is such that the robot collides with the environment, the found pose is deleted and the starting pose is re-optimized for at most 3 times, otherwise it is discarded. Checking collision only at the end speeds-up the procedure, since it is a time-consuming task. If the pose still collides, chances are that the starting pose cannot be improved since any direction would lead to a collision. Of course, this may cause that the absolutely best pose that doesn't collide is not found, but since it is not required to obtain the absolutely best TCP pose, this is an admissible trade-off.

It is also worth recalling that optimization is done considering a simple PTP motion: if the trajectory is instead a linear motion, there is no guarantee that the found metrics is optimized. However, choosing very close trajectory points should minimize this and other non-idealities.

4.1.1 An alternative version of the algorithm

If the required optimization target is to focus just on the fastest joint (i.e., try to set the TCP pose in order to use only that joint, as much as possible), a valuable alternative is to take the weights vector `W` and square all its entries: in this way, a joint with a maximum velocity that is 1/2 of

the fastest one will be considered doubly slower, so the maximum velocity is 1/4. A comparison between the weights is reported in tab. 4.2. This choice does not guarantee at all that the new TCP pose will make all the joints still but the fastest one, however it may help a human user to find a suitable pose with this property. To make instead all the slower joints not move, a penalization should be added each time more than one axis moves. However, this choice shall be considered only if the user already knows what to expect as an optimum. Instead, the assumption is that whoever makes use of this optimization algorithm just knows the admissible zone of the TCP/tool, and may lead in some way to the optimum just by suggesting this range. In the following of this work, a comparison between the solutions is going to be made.

Table 4.2: Weights vector for the two versions of the velocity optimization algorithm. Vectors are normalized to highlight the differences: actually, they are not in the algorithm. The velocities are converted from °/s to rad/s. Notice that the weight of the last joint is halved, while joints 2 and 3, the slower ones, increase

Algorithm	W_1	W_2	W_3	W_4	W_5	W_6
Joint velocity	0.4571	0.5079	0.5079	0.2689	0.3809	0.2406
Fastest joint	0.4599	0.5678	0.5678	0.1592	0.3194	0.1274

4.2 Range optimization

As for working range optimization, it is necessary to recall that this method is mostly suited in case of sparse poses, for instance random bin picking: it was not designed to optimize a whole trajectory even if, in principle, it could work anyway.

For the robot under evaluation (ABB IRB 6700-235/2.65), ranges are reported in tab. 4.3. There, it is possible to notice that, if the joints are set at their centers, the robot will be in singular configuration (as can be seen in fig. 4.2), in particular for joints 5 and 3 that make the previous and subsequent links aligned or almost aligned, respectively.

Table 4.3: Ranges of axes, drawn from the official manual of ABB IRB 6700-235/2.65

	axis 1	**axis 2**	**axis 3**	**axis 4**	**axis 5**	**axis 6**
min limit	−170°	−65°	−180°	−300°	−130°	−360°
max limit	170°	85°	70°	300°	130°	360°
range	340°	150°	250°	600°	260°	720°
center	0°	10°	−55°	0°	0°	0°

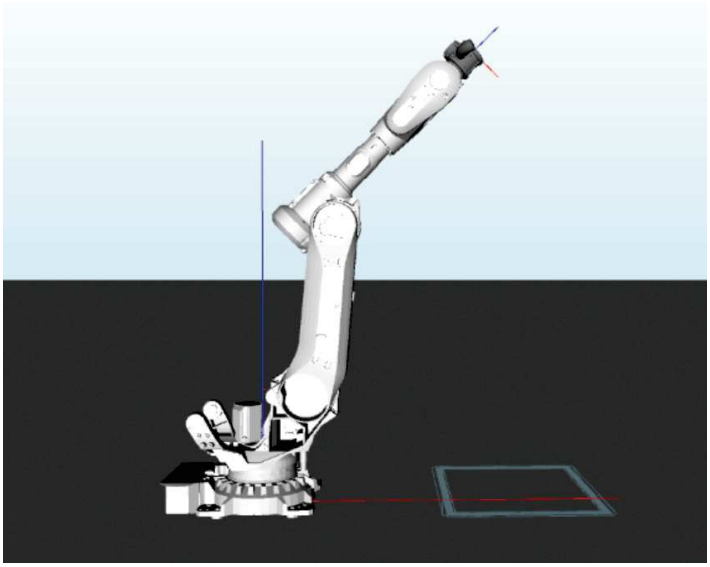


Figure 4.2: The robot with all the axes at the center of their range: it is possible to notice a singular configuration.

Hence, as optimization criterion, the score to be associated with each TCP pose must be related to the ability to make the joint ranges as close as possible to their centers, as well as far from singular configurations. This is done by evaluating each joint i by the already presented Generalized Gaussian Distribution:

$$peak \cdot e^{-(|q_i - \mu|/\alpha)^\beta}$$

with $\alpha = \frac{\sqrt{2} \cdot range}{6}, \beta = 2.7, \mu = center$. Peaks must go to 1, in general. For joints 3 and 5 two distributions are created, one with ranges $-180°; -90°$ (peak 0.6) and $-90°; 70°$ (peak 1) , the other with ranges $-130°; 0°$ (peak 0.6) and $0°; 130°$ (peak 1). The different peaks are meant to lead, if possible, the configuration to the highest score but, in case this was not possible, the score would not be 0. The resulting shape of the distributions are reported in fig. 4.3. It is worth noting that this choice of the score function allows to weigh differently an improvement of the same amount for two different joints: for instance, a deterioration by 15° for axes 3 may lead from the peak to almost 0.5, while joint 6, if starting from the central configuration, would not even be minimally affected. The pseudocode of the algorithm is pretty much the same of the previous one, with just some differences.

```
% Initialization
Generate test TRAJECTORY as sequence of waypoints (position + rotation);
```

```
Generate 50 initial TCP poses inside allowed range (INITIALPOSES array);
For each POSE in INITIALPOSES:
|    while POSE is not feasible for TRAJECTORY:
|_   |_   choose a new POSE randomly;

% Optimization
FinalPoses = [];
For each POSE in INITIALPOSES:
|    AverageScore = Average(EvaluatePoseRange(POSE));
|    MinScore = MIN(EvaluatePoseRange(POSE));
|    Choose randomly ENTRY = 1,2,3,4,5,6;
|    Choose randomly SIGN = -1,+1;
|    CONTINUEIMPROVE = true;
|    NumOfIterationsNoImprove = 0;
|    NumOfIterationsImprove = 0;
|    while(CONTINUEIMPROVE):
|    |    NewPOSE = ChangePose(POSE, ENTRY, SIGN);
|    |    NewAverageScore = Average(EvaluatePoseRange(NewPOSE));
|    |    NewMinScore = MIN(EvaluatePoseRange(NewPOSE));
|    |    if (NewAverageScore>AverageScore & NewMinScore>=MinScore*0.9):
|    |    |    POSE = NewPOSE;
|    |    |    MaxScore=NewMaxScore
|    |    |    NumOfIterationsNoImprove = 0;
|    |    |    NumOfIterationsImprove += 1;
|    |    |    if (NewMinScore > MinScore):
|    |    |    |_   MinScore = NewMinScore;
|    |    else:
|    |    |    NumOfIterationsNoImprove += 1;
|    |    |    NumOfIterationsImprove = 0;
|    |    |    Choose randomly ENTRY = 1,2,3,4,5,6;
|    |    |_   Choose randomly SIGN = -1,+1;
|    |
|    |    if (NumOfIterationsImprove > 25):
|    |    |    Choose randomly ENTRY = 1,2,3,4,5,6;
|    |    |_   Choose randomly SIGN = -1,+1;
|    |    if (NumOfIterationsNoImprove > 40):
|    |_   |_   CONTINUEIMPROVE = false;
|
|    If POSE collides with objects in the scene during trajectory:
|    |    reset POSE;
|    |    If POSE have been reset > 3 times:
|    |    |    neglect POSE
|    |    else:
|    |    |_   restart improving;
|    else:
|_   |_   FinalPoses.Append(POSE)

% Methods
def EvaluatePoseRange(TCPpose):
```

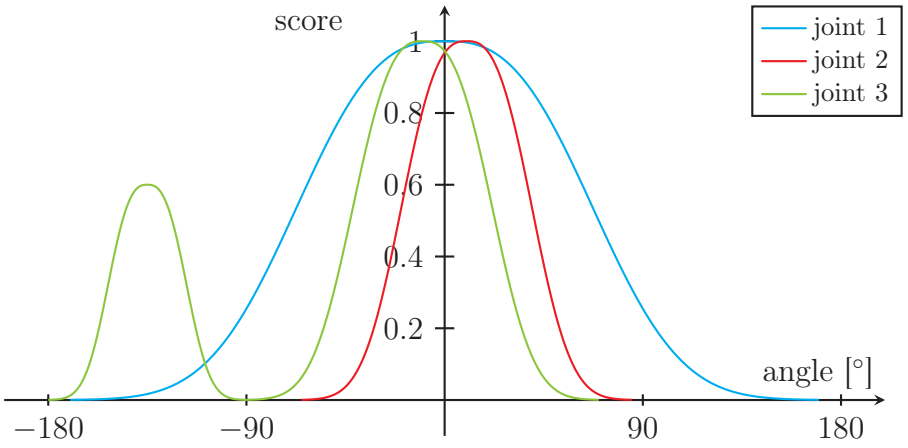
```
Score = [];
For each POINT in TRAJECTORY:
|    NEXT_POINT = POINT + 1;
|    Set the robot to reach POINT with TCPpose. Save config Q;
|    If unreachable or singular:
|    |_   return [-1 -1 ... -1];
|    GaussQ = [];
|    For i from 1 to 6 :
|    |_   GaussQ[i] = GeneralizedGaussian(Q[i]);
|    MinScore = MIN(GaussQ);
|_   Score.Append(MinScore);
return Score;

def ChangePose(Pose, entry, sign):
% the same as before
```

The main difference is that now each configuration vector q is associated with a score that is the minimum among the generalized gaussian values of all its entries (worst joint configuration for that point). For each point in the "trajectory", that minimum is saved in an array. New TCP poses are considered to be better only if the average of this array is strictly greater and the minimum doesn't decrease more than 10% than the previous pose. If instead the minimum increases, the new value becomes the lower limit to assess the following poses, so both average and minimum improve, hence the new TCP pose will be better for sure.

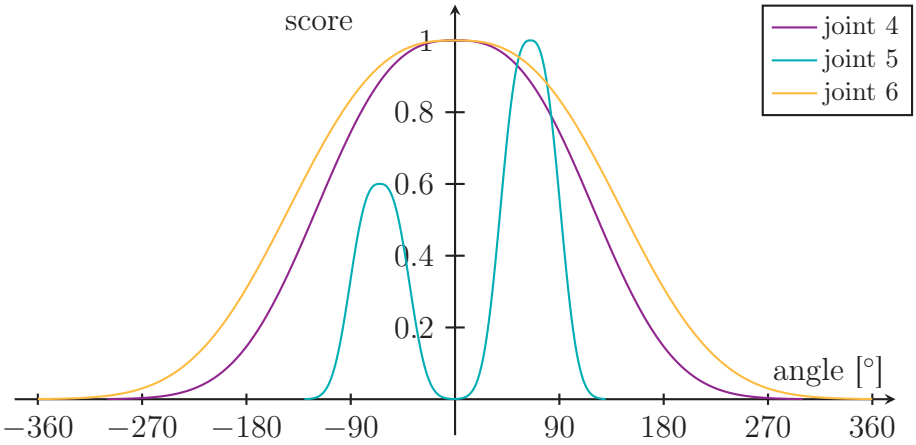
4.3 Maximization of number of picked items and PSA

As already mentioned, the focus of the hosting company is mainly random bin picking applications. Hence, it was worth it to try to optimize the tool/TCP pose in order to make it more effective.

Considering an example in which there are multiple items in a box that the robot has to pick, the target should be to maximize the number of objects that can be successfully picked without collisions. In this case, there is not a metric represented by a real number, like velocity and range, but just the fraction of good picks. Since even a small change in the tool pose may drastically reduce the score, a research algorithm based only on this metric and on a random direction of improvement may be complicated. These considerations led to the implementation of a pseudo-pattern search algorithm, that was afterwards extended also in the previous



(a) Joints 1, 2 and 3



(b) Joints 4, 5 and 6

Figure 4.3: Joint ranges and associated scores for robot ABB IRB6700-235/2.65. It is clear that joints 1, 4 and 6 are the less restricted ones, so optimization will focus more on the others, that may also suffer from singularity

cases of velocity and range optimization, with slight changes[1].

4.3.1 The pseudo-pattern search algorithm

```
% Initialization
Generate N randomly placed objects in a bin.
For each one, save possible pick points (PICKPOSITIONS, array of arrays).
Ask user to choose how many poses to return (M, typically 3).
Generate initial TCP poses inside allowed range as follows:
|    Rotation always 0;
|    X = 0, 100, ..., X_max
|    Y = 0, 100, ..., Y_max
|_   Z = 0, 100, ..., Z_max
Save them in INITIALPOSES array;
Select M poses with function EvaluatePosePicks(POSE):
first, select poses with highest number of successful picks.
Sort them according to their range-associated score.
Delete all other poses from INITIALPOSES.

% Optimization
FinalPoses = [];
For each POSE in INITIALPOSES:
|    Score = EvaluatePosePicks(POSE);
|    NEIGHBORS = all neighboring poses of POSE (25 cm/22.5 deg);
|    BETTERNEIGHBORS = [];
|    BETTERNEIGHscores = [];
|    while (true):
|    |    for each NeighPose in NEIGHBORS:
|    |    |    Nscore = EvaluatePosePicks(NeighPose);
|    |    |    if (Nscore.COUNTNONZERO() > Score.COUNTNONZERO):
|    |    |    |    BETTERNEIGHBORS.APPEND(NeighPose);
|    |    |    |    BETTERNEIGHscores.APPEND(Nscore);
|    |    |    else if (Nscore.COUNTNONZERO() == Score.COUNTNONZERO):
|    |    |    |    if (Nscore.AVERAGE() > 1.05*Score.AVERAGE())
|    |    |    |    |    BETTERNEIGHscores.APPEND(Nscore);
|    |    |_   |_   |_   BETTERNEIGHBORS.APPEND(NeighPose);
|    |
|    |    if (BETTERNEIGHBORS.EMPTY()):
|    |    |    FinalPoses.APPEND(POSE);
|    |    |    break;
|    |    else:
|    |    |    best = BestIndexInArray(BETTERNEIGHscores);
|    |    |    Score = BETTERNEIGHscores[best];
```

[1]Since translations and rotations are measured with different units (mm and degress), for velocity and range a random "domain" is initially chosen - whether distance or angular difference - and neighboring poses are found only in that domain. After that, the other one will be explored and so on until convergence. Moreover, the choice of initial TCP poses is kept random.

```
|    |    |    NEIGHBORS = neighboring poses of BETTERNEIGHBORS[best];
|    |    |    BETTERNEIGHBORS.CLEAR();
|_   |_   |_   BETTERNEIGHscores.CLEAR();

% Methods
def EvaluatePosePicks(TCPpose):
RANGESCORES = [];
GOODPICKS = [];
For each OBJECT in PICKPOSITIONS:
|    tempSCORES = [];
|    For each PICKpt in OBJECT:
|    |    Set robot to reach PICKpt with TCPpose;
|    |    If OK, save EvaluateRange2(TCPpose,PICKpt) in tempSCORES;
|    |_   Otherwise, save 0.0 in tempSCORES;
|    Save tempSCORES.MAX() in RANGESCORES,
|_   and associated PICKpt in GOODPICKS;
return (RANGESCORES, GOODPICKS);

def EvaluateRange2(TCPpose, POINT):
Save robot configuration Q;
GaussQ = [];
For i from 1 to 5 :
|_   GaussQ[i] = GeneralizedGaussian(Q[i]);
return MIN(GaussQ);

def BestIndexInArray(Scores):
Idx = 0;
for i from 1 to Scores.COUNT():
|    if (Scores[i].COUNTNONZERO() > Scores[Idx].COUNTNONZERO()):
|    |    Idx = i;
|    else if (Scores[i].COUNTNONZERO() ==Scores[Idx].COUNTNONZERO()):
|    |    if (Scores[i].AVERAGE() > Scores[Idx].AVERAGE()):
|_   |_   |_   Idx = i;
return Idx;
```

The developed algorithm is different if compared to the random search one: instead of a fixed number of 50, this time initial poses are created equally spaced (100 mm in X, Y and Z) to be sure to span the whole range. A minor difference is that, if possible, the rotation of the tool is kept = 0 for all the axes. Starting poses are evaluated by means of the fraction of picked objects over the total and, in case of two equal poses, the one with better joints configuration (on average) is preferred: this forces the manipulator to grab all the object without getting too far from the center of joints ranges or close to singularities. It is also worth noting that the angle of the sixth joint is neglected, since its range is very wide and, being the last one, values closer to its limits may be inevitable. Then, only the best starting poses are taken and improved, by this meaning that the

optimizer looks for the adjacent poses both in translations and rotations (indeed, it is not expected to move too far away from the initial position of the TCP), restarting optimization from the best one. To count the number of picked objects, a simple collision check is performed. However, the major difference is that now, for every item in the bin, many pick positions are evaluated and the best one in terms of joints ranges is returned. This makes optimization much more time-consuming. In addition, the data to be provided to the optimizer are much different: there is not just a list of homogeneous matrices encoding position and orientation of the object, but rather a list of lists of them, each one being a possible "pick position" for grabbing the item.

It is important to notice that the function `EvaluatePosePicks` in the pseudocode returns also the picking points of each object. They will be useful later on when showing the solution and saving data. In addition, in this function a possible pick point on the object is said to be good if no collision is caused and there are not issues as far as reachability and singularity are concerned.

A comparison between random search and pattern search algorithms for velocity and range optimization will be carried out later on when discussing the results with specific trajectories.

4.4 Development of a genetic algorithm

One of the biggest drawbacks of the methods just presented, despite their simplicity and intuitiveness, is the way used to find a possible "direction of improvement". Being a robot manipulator a redundant, complex, non-linear system that often needs numeric algorithms to compute inverse kinematics, it is not easy to find an analytical method to retrieve an optimal TCP pose. Many papers that were reviewed before starting made use of optimization methods to approximate the function to be minimized, or even fancy algorithms, like "ant colony" [18] or genetic ones [12].

In the following, the pseudocode of a genetic algorithm for velocity optimization is reported: range optimization is similar. However, as far the maximization of number of picked items is concerned, no genetic algorithm was built given the complexity of the objective function and the higher time-requirements due to the number of needed initial poses.

```
% Initialization
Generate test TRAJECTORY as sequence of waypoints (position + rotation);
Generate 200 initial TCP poses inside allowed range (ALLPOSES array);
```

```
ALLPOSESSCORES = [];    % array of arrays!
For each POSE in ALLPOSES:
|    while POSE is not feasible for TRAJECTORY:
|    |_   choose a new POSE randomly;
|_   ALLPOSESSCORES.Append(EvaluatePoseVelocity(POSE))

% Optimization
ALLPOSESMAXIMA = [];
for POSESCORE in ALLPOSESSCORES:
|_   ALLPOSESMAXIMA.Append(POSESCORE.Max);

CumulativeReproductionProbability = UpdateProb(ALLPOSESMAXIMA);
int ITERS = 0;

BESTposes = ALLPOSES;
BESTscores = ALLPOSESSCORES;

while (ITERS < 25):
|
|    NEWGENERATION = [];
|    NEWGENERATIONSCORES = [];
|
|    for GENIDX from 1 to ALLPOSES.Count()/2:
|    |    Pose1 = ChooseWithProbability(randomDouble(1.0),
|    |                CumulativeReproductionProbability, ALLPOSES)
|    |    Pose2 = ChooseWithProbability(randomDouble(1.0),
|    |                CumulativeReproductionProbability, ALLPOSES)
|    |
|    |    [Child1,Child2] = Reproduction(Pose1, Pose2);
|    |    ChildPose1 = Mutation(Child1);
|    |    ChildPose2 = Mutation(Child2);
|    |    ScoreChild1 = EvaluatePoseVelocity(ChildPose1);
|    |    ScoreChild2 = EvaluatePoseVelocity(ChildPose2);
|    |
|    |    NEWGENERATION.Append(Child1);
|    |    NEWGENERATION.Append(Child2);
|    |    NEWGENERATIONSCORES.Append(ScoreChild1.Max);
|    |_   NEWGENERATIONSCORES.Append(ScoreChild2.Max);
|
|    if (MIN(NEWGENERATIONSCORES) < MIN(BESTscores)):
|    |    BESTscores = NEWGENERATIONSCORES;
|    |    BESTposes = NEWGENERATION;
|    |    ALLPOSES = BESTposes;
|    |    ALLPOSESMAXIMA = BESTscores;
|    |    ITERS = 0;
|    else:
|    |    ALLPOSES = NEWGENERATION;
|    |    ALLPOSESMAXIMA = NEWGENERATIONSCORES;
|    |_   ITERS ++;
```

```
|
|_   CumulativeReproductionProbability = UpdateProb(ALLPOSESMAXIMA);

% Methods
def ChooseWithProbability(randomDouble, CumulProbab, PosesArray):
for i from 0 to CumulProbab.Count()-1:
|    if (CumulProbab[i] > randomDouble):
|_   |_   return PosesArray[i];

def UpdateProb(PosesMaxima):
Probabilities = [];
CumulativeProbabilities = [];
maxTime = MAX(PosesMaxima);
sumOfTimes = SUM(PosesMaxima);
for i from 0 to PosesMaxima.Count()-1:
|    Probabilities.Append((maxTime - PosesMaxima[i]) /
|                          (maxTime*PosesMaxima.Count() - sumOfTimes));
|    if (i == 0):
|    |    CumulativeProbabilities.Append(Probabilities[0])
|    else:
|    |    CumulativeProbabilities.Append(CumulativeProbabilities[i-1] +
|_   |_                                    Probabilities[i]);

def Mutation(POSE):
Choose randomly ENTRY = 1,2,3,4,5,6;
Choose randomly SIGN = -1,+1;
return ChangePose(POSE, ENTRY, SIGN); %see pseudocodes before

def Reproduction(Pose1, Pose2):
Create two empty poses Child1, Child2;
%X
Choose randomly PARENT = 1,2;
if (PARENT == 1):
|    Child1.X = Pose1.X
|    Child2.X = Pose2.X
else:
|    Child1.X = Pose2.X
|_   Child2.X = Pose1.X
%Y
Choose randomly PARENT = 1,2;
if (PARENT == 1):
|    Child1.Y = Pose1.Y
|    Child2.Y = Pose2.Y
else:
|    Child1.Y = Pose2.Y
|_   Child2.Y = Pose1.Y
%... same for Z, A, B, c
return [Child1, Child2];
```

```
102 %randomDouble(x) returns a double 0<= but <x;
103 %randomBool() a random boolean variable
```

Basically, what the algorithm does is to instantiate 200 initial poses randomly within the admissible TCP range. Then, it assigns a reproduction probability based on the distance from the worst score within all the poses (function `UpdateProb`). The higher this distance, the higher the probability. Cumulative probability is just used to effectively implement a random choice based on this score. Two poses are randomly selected in this way, and they generate two children poses (with uniform crossover) subject to one random mutation each. The two children are evaluated and they populate a new array of poses. This happens 100 times, i.e. another generation of 200 poses is created in this way. Then, the entire new population scores are assessed: if a new pose is better than the best generation so far (by this meaning that it has a lower maximum/bottleneck), the last population becomes the new best, otherwise the algorithm runs again. Keeping track of the best generation allows to continue the evolution without concern of losing a possibly good result. The "while" loop with `ITERS` that goes from 0 to 25 is just a way to stop the algorithm: if 25 consecutive reproductions without generation of better children occur, optimization is halted. After this, the best n poses (where n is chosen by the user, by best meaning the poses with lowest maximum scores) are returned.

As far as joints range optimization is concerned, the only difference is that the worst score is now the minimum, as already seen in the previous algorithms. So the new generation becomes the best only if the new minimum is greater than the old one. Hence, in this case the optimization criterion given by the improvement of the average is lost, but, as shown in the following, convergence to a good TCP pose still happens.

It is worth analyzing the differences between genetic optimization and the initial version of the algorithm (from now onward referred to as "**random search**", since the choice of the direction of improvement is random). As already stated, genetic algorithms do not have strong convergence properties, and this holds also for the other algorithm: indeed, the latter relies on the initialization of 50 poses and randomly selects a direction of improvement, but both depend on initialization since local minima are a major drawback in the functions used to assign a score to the poses.

Collision checking is easier for the random search algorithm, since when it converges to a good pose this check is performed, while the genetic algorithm version verifies collision at the moment of returning the best poses, with no possibility to correct it by restarting, unlike the other one.

Indeed, since the initial population is 200 poses, and it is not possible to foresee how many others will be generated, including collision checking in the middle may make the total run time excessive.

The choice of 50 initial poses for random search is due to time reasons: the higher this number, the greater the time needed to analyze all the poses. The correlation between number of poses and time is less obvious for genetic algorithms, since it is all based on random reproduction and mutation. However, the trials showed that 200 initial poses still guarantee fast convergence to a good result. Also, having 4 times the initial number of poses increases the probability to initialize a very good pose already at the beginning. In addition, the smaller the search space, the faster convergence, that is also true for random-search. Trials showed a run time between 3 and 7 minutes for the genetic algorithm, and 6 to 12 minutes for the random search one.

Another problem of GA is that the final poses tend to be similar one to the other, since every child is more likely a son of the best poses. This does not happen for random search algorithm since every starting pose may be substantially different if compared to the others, and convergence steps lead to even further TCPs. This means that, if the user cares about the possibility of having multiple choices to evaluate, the older version is (in principle) to be preferred.

It is worth noting that the algorithm that maximizes the number of picks works only in random-search or pattern-search mode. The former was just a trial: the most accurate results were achieved by the latter.

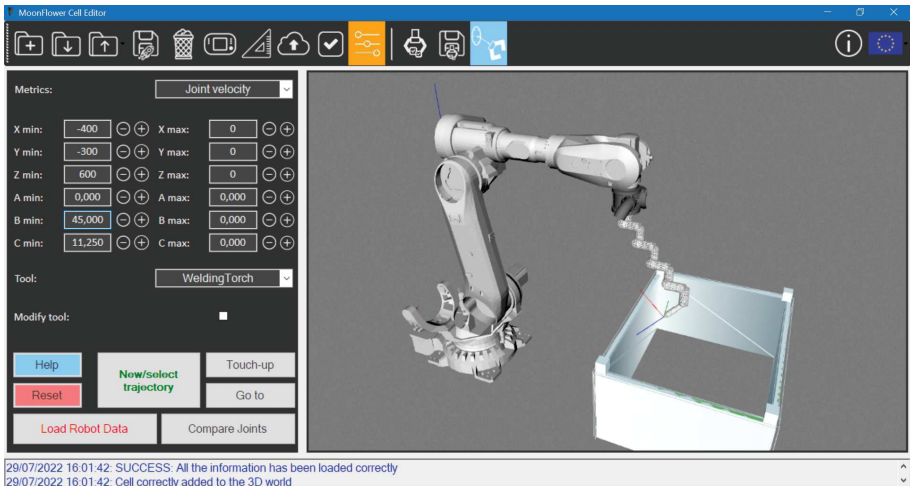
4.5 General implementation details

4.5.1 The user interface

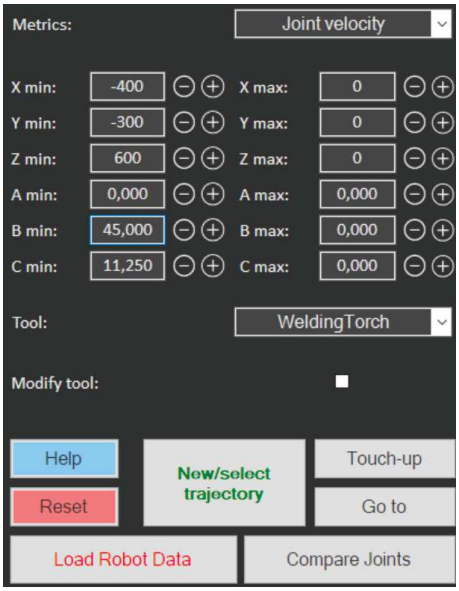
In fig. 4.4 some screenshots of the designed optimizer are reported. In the following, the description of the settings:

After selecting the optimizer (the blue button on the top bar of fig. 4.4a), the user has to select a ".xml" data file used to build the connection link between flange and tool. This was developed thanks to other softwares provided by the hosting company, and the standard one consists of:

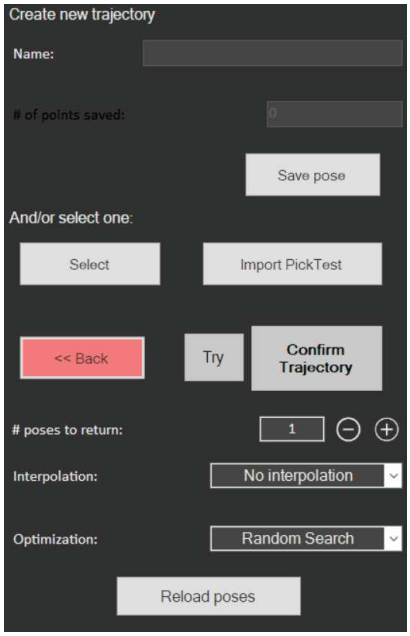
- A circular "counterflange". Height = 1cm, radius = 8.5cm
- A cylindrical link extending in Z direction (it exits from the flange) by 15% of the total Z-offset. Radius = 4.5cm



(a)



(b)



(c)

Figure 4.4: Screenshots from the developed software. Fig. 4.4a displays the whole system, while fig. 4.4b, 4.4c collect all the settings that the user may act on to change the behavior of the optimization algorithm

- A sequence of 4 Y-X-Z cylindrical links: X and Y ones occupy 25% each of the offset in that direction, the first 3 in Z 20% and the last one in Z 25%. Radius = 3cm

Of course, many other options may be developed. However, to keep the solution simple, only cylindrical mono-directional links were able to be included. Then, the optimization metric as can be seen in fig. 4.4b, that may be joint velocity, fastest joint selection, joint range or number of picks maximization, has to be selected. After the selection of the tool, the user can choose the admissible joint range by acting on the 12 text boxes: X,Y,Z are the linear offset in the flange reference frame, while A,B,C the rotation with ZYX convention. In addition, it is possible to change the reference frame from TCP to tool-base ("Modify tool" checkbox). Among all the buttons below, the most important ones are "Load Robot Data", that allow to select the file storing the information about maximum velocity and admissible range of the joints, as well as possible singularities (fig. 4.5), and "New/select trajectory" that opens the trajectory selection window shown in fig. 4.4c.

```
<Data OTS="MoonFlowerCellEditor.RobotData">
  <RobotName>ABB IRB6700-235_2.65</RobotName>
  <J1MaxVel>100</J1MaxVel>
  <J2MaxVel>90</J2MaxVel>
  <J3MaxVel>90</J3MaxVel>
  <J4MaxVel>170</J4MaxVel>
  <J5MaxVel>120</J5MaxVel>
  <J6MaxVel>190</J6MaxVel>
  <J1_MinLimit>-170</J1_MinLimit>
  <J1_MaxLimit>170</J1_MaxLimit>
  <J2_MinLimit>-65</J2_MinLimit>
  <J2_MaxLimit>85</J2_MaxLimit>
  <J3.1_MinLimit>-90</J3.1_MinLimit>
  <J3.1_MaxLimit>70</J3.1_MaxLimit>
  <J3.2_MinLimit>-180</J3.2_MinLimit>
  <J3.2_MaxLimit>-90</J3.2_MaxLimit>
  <J4_MinLimit>-300</J4_MinLimit>
  <J4_MaxLimit>300</J4_MaxLimit>
  <J5.1_MinLimit>0</J5.1_MinLimit>
  <J5.1_MaxLimit>130</J5.1_MaxLimit>
  <J5.2_MinLimit>-130</J5.2_MinLimit>
  <J5.2_MaxLimit>0</J5.2_MaxLimit>
  <J6_MinLimit>-360</J6_MinLimit>
  <J6_MaxLimit>360</J6_MaxLimit>
</Data>
```

Figure 4.5: Screenshots of the robot data file to be loaded. It is possible to see the maximum velocities and joint ranges (also including sub-ranges due to singularities)

This window allows to choose the optimization trajectory or even to define a new one (indeed, the company's software is provided with a simulated teach pendant that is able to move to the robot joints), or also to import a PickTest, that is the result of another functionality of the appli-

cation that computes the poses to pick all the items in a box. The user may then select the number of optimal poses to return, ordered from the best one; whether to interpolate or not the points choosing linear or PTP interpolation (only for display purposes) and, finally, the optimization algorithm (random search, pattern search or genetic, if allowed).

4.5.2 Initialization of the optimizer

After setting all the parameters, it is necessary to initialize a starting set of poses to optimize. As said, for the random-search algorithm 50 poses are created, while the genetic one starts with 200 ones. All of them are randomly instantiated, according to the C# code snippet reported here:

```
// Define a possible starting set of TCP Poses (1 every 5 cm or pi/8)
// TCPLim_ is a tuple <min;max> storing the range limits
double angle = Math.PI/8;
int NPosX = (int)Math.Floor((TCPLimX.Item2 - TCPLimX.Item1) / 50) + 1;
int NPosY = (int)Math.Floor((TCPLimY.Item2 - TCPLimY.Item1) / 50) + 1;
int NPosZ = (int)Math.Floor((TCPLimZ.Item2 - TCPLimZ.Item1) / 50) + 1;
int NRotA = (int)Math.Floor((TCPLimA.Item2 - TCPLimA.Item1) / angle) + 1;
int NRotB = (int)Math.Floor((TCPLimB.Item2 - TCPLimB.Item1) / angle) + 1;
int NRotC = (int)Math.Floor((TCPLimC.Item2 - TCPLimC.Item1) / angle) + 1;

ToolPose.X = TCPLimX.Item1 + 50 * Random.Next(0, NPosX);
ToolPose.Y = TCPLimY.Item1 + 50 * Random.Next(0, NPosY);
ToolPose.Z = TCPLimZ.Item1 + 50 * Random.Next(0, NPosZ);
ToolPose.A = TCPLimA.Item1 + angle * Random.Next(0, NRotA);
ToolPose.B = TCPLimB.Item1 + angle * Random.Next(0, NRotB);
ToolPose.C = TCPLimC.Item1 + angle * Random.Next(0, NRotC);
```

Basically, every 5cm for linear offset or $\frac{\pi}{8}$rad for rotation, a (possible) initial pose is created, except for "number of picks maximization" that starts, if possible, with zero rotation and from fixed poses (see section 4.3). This means that the standard TCP range space ($\pm$350mm in X and Y, $200; 700$mm in Z, $\pm 90°$ rotation about Z and $\pm 45°$ about Y and X) has $8 \times 8 \times 11 \times 9 \times 5 \times 5 = 158400$ initial poses among which the starting set is chosen. The choice is actually completely random. One may ask why not keeping the choice of starting poses in the whole continuous real space $\Re^6$. The reason is only to make the following optimum-search algorithm faster: The next pose will be 1cm or $\pi/32$rad different, so the step size is fixed, and starting from any point of the real axis it may happen that convenient standard positions or rotations are skipped (like the $(0, 0, 0)$ position or the angles $0°, 45°, 90°$). So the possibilities are limited, also considering that the tool will be possibly designed and installed by a human, who will likely

round off to the first decimal place, being such a small difference from the found optimal tool almost irrelevant. Indeed, also the admissible values of the range variables in the user interface are limited and quantized.

Another objection may regard the so reduced number of initial poses compared to the possibilities (50 or 200 out of 158400, or even more). Beyond obvious time-related reasons, to justify this choice, it is worth recalling that the robot will be positioned in an environment with obstacles or generic limits, hence a big part of the workspace may be excluded from the admissible motion region. Therefore, many initial TCP/tool poses will not be feasible. That is the reason for the following check before starting optimization: the trajectory is tested with the initial tool and, if any collision is found, a new random pose will be selected. However, the algorithm does not test whether two identical starting poses are chosen: chances are that the randomness of the choice of the search trajectory will lead to very different poses anyway. Again, the optimizer that deals with maximization of picked object is different, as already shown.

After this initialization, the optimizer starts. Every final pose comes with an evaluation. In case of velocity optimization a real number, the smaller the faster, is returned. As far as range optimization is concerned a real value in $[0; 1]$ is provided, the greater the better, while for picked items optimization an integer for the number of reachable objects is returned, as well as a score related to the joints configurations. Finally, the best poses are shown and saved according to these metrics.

Chapter 5

Results and discussion

Before digging into the results, it is necessary to recall the typical TCP range limits imposed in order to avoid the tool to move far apart from the flange. The developed program allows to select two different frames: the TCP frame itself, or the tool base (i.e. the mounting point to be attached to the flange). Typically, the limits are set to ± 350mm for X and Y directions (minimum and maximum are always symmetric), $+100\text{mm} \cdots + 700\text{mm}$ for Z direction, $\pm 90°$ for roll and $\pm 67.5°$ for pitch and yaw. However, the user has the possibility to change them.

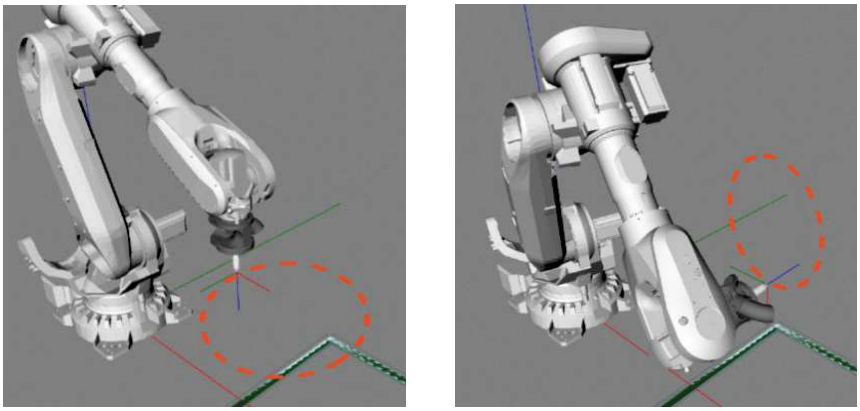
5.1 Velocity optimization

To assess the capabilities of the velocity optimization algorithm, many trajectory have been studied. The first two are just circles: one lying on a plane perpendicular to the Z-axis of the world frame, the other one belonging to another plane perpendicular to the Y-axis, as shown in figure 5.1. As far as this kind of optimization is concerned, only the random search and genetic algorithms give good results. Indeed, the pattern search method were proven to be much slower while converging, in many cases, to the same solution, as reported in the following.

5.1.1 Two circular trajectories

As tool, a simple model of the tip of a welding torch (16cm long) was created, as depicted in the image. The points on the trajectory have been recorded such that one axis of the tracking tool's TCP would always be parallel to the radius. Each point is just 45° from the other, so a total of 9 points form each circle. The expected result of the optimization algorithm

is to have the flange of the robot directly above the center of the circle, so as to use only the last and fastest joint to track it. The TCP range, however, was initially restricted to ±450mm or ±400mm, respectively, in X and Y, 100 . . . 500mm in Z, ±90° rotation about Z and 0° about Y and X: in this way, finding the optimal was easier and faster.



(a) (b)

Figure 5.1: Two simple circular trajectories to evaluate velocity optimization algorithms. They are circles lying on planes that are perpendicular to one axis of the robot reference frame. The former circle has a radius of 400mm, the latter 350mm

Indeed, the random-search algorithm provides as best results a TCP frame located at $X = 0\text{mm}, Y = -400\text{mm}, Z = 300\text{mm}; A = 90°, B = C = 0°$, depicted in figure 5.2, or $X = -280\text{mm}, Y = -280\text{mm}, Z = 200\text{mm}; A = 45°, B = C = 0°$. Actually, different runs of the algorithm may find different values of Z: indeed, changing the offset in this direction does not impact on the result at all. It is worth noting that the two TCPs are almost the same even if they are placed differently: actually, $\sqrt{280^2 + 280^2} \simeq 396$, hence, despite a not completely aligned tool and a small motion of the other joints, also the second set of coordinates is almost optimal.

For the first alternative, the maximum PTP motion for each joint to track the trajectory is:

$$\begin{bmatrix} 0° \\ 0° \\ 0° \\ 0° \\ 0° \\ 45° \end{bmatrix} \text{ the time needed is: } \begin{bmatrix} 0.0\text{s} \\ 0.0\text{s} \\ 0.0\text{s} \\ 0.0\text{s} \\ 0.0\text{s} \\ 0.237\text{s} \end{bmatrix} \tag{5.1}$$

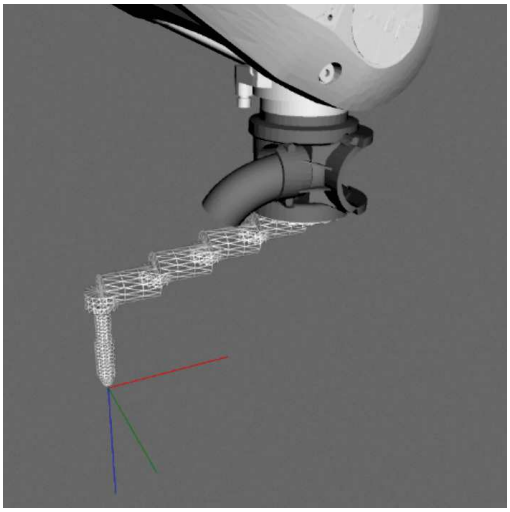


Figure 5.2: Found tool for the first circular trajectory. Coordinates of the TCP in the flange reference frame: $X = 0\text{mm}, Y = -400\text{mm}, Z = 300\text{mm}; A = 90°, B = C = 0°$

Indeed, by plugging directly the welding torch on the flange and forcing the trajectory, the maximum amount that each joint needs to track[1] would be:

$$\begin{bmatrix} 11.2° \\ 14.7° \\ 16.5° \\ 0.0° \\ 4.2° \\ 55.2° \end{bmatrix} \tag{5.2}$$

It is clear to see from comparing 5.1 and 5.2 that not only the joints 1,. . . ,5 do not move anymore, but also the last joint has a smaller maximum motion. Of course, the assumption is that any PTP movement must last the same time, otherwise there would be no guarantee that the found optimal is effectively better. In case of welding, this is not a too strict assumption, since power and velocity of the torch should be kept constant during the entire trajectory.

As far as the other circle is concerned, the best TCP frame is $X = 0\text{mm}, Y = 350\text{mm}, Z = 100\text{mm}; A = 90°, B = C = 0°$, or $X = 0\text{mm}, Y =$

[1]Note that each entry does not correspond to the same motion, in general: for instance, joint 1 may have its maximum motion when moving from point 1 to 2, while joint 2 from point 4 to 5

$-350\text{mm}, Z = 100\text{mm}; A = -90°, B = C = 0°$. Also in this case the algorithm finds that multiple options are available. For both, the maximum motion of each joint is again:

$$\begin{bmatrix} 0° \\ 0° \\ 0° \\ 0° \\ 0° \\ 45° \end{bmatrix} \text{ the time needed is: } \begin{bmatrix} 0.0\text{s} \\ 0.0\text{s} \\ 0.0\text{s} \\ 0.0\text{s} \\ 0.0\text{s} \\ 0.237\text{s} \end{bmatrix} \tag{5.3}$$

The non-optimized standard torch position would lead to maximum motions of:

$$\begin{bmatrix} 2.3° \\ 13.7° \\ 19.2° \\ 4.2° \\ 3.0° \\ 58.1° \end{bmatrix} \tag{5.4}$$

The genetic algorithm versions of both can eventually find roughly the same final TCP poses, without leading to the exact optimal. For instance, the XY circle is tracked with a TCP with offset in X of 350mm instead of 400: this can be justified by the fact that the genetic algorithm does not optimize every single pose by modifying the location and orientation in any direction and checking whether it improves, like the random-search one. Finding the real optimum is less likely, so it stops near. Anyway, it may reach good results that provide the user a valuable intuition on how to design the tool.

As far as the pattern search algorithm is concerned, results are almost identical to the ones given by the random search method. The only difference is the time needed to converge to such solutions: starting from the current tool, all the neighboring positions are analyzed. Therefore, while in principle the PSA goes directly to the "best neighbor", testing all the directions before choosing slows down convergence. Indeed, time requirements are often doubled (from 10 to 20 minutes). Since velocity and range optimization are designed to optimize many poses, and they may take several steps from starting position to local optimum, it is clear that a faster method is to be preferred, given that results do not change. This is true also for the other trajectories tested in this work, so the pattern search method will not be considered for now.

Fastest joints selection

As far as the algorithm with squared weights is concerned, in case the TCP range is set to be the same reported above no difference in the optimal tool pose can be found. Thus, in order to examine the capabilities of this new algorithm, the allowed TCP range were extended to see whether it could find or not the optimum. Hence, the second rotation (about the Y axis of the new TCP frame) were allowed to be in the range $\pm 45°$, for the X-Y circle. The first solution for the TCP pose is: [−400mm; 40mm; 500mm; $-5.625°$; 0°; 0°] (from now onward the pose will be reported in the form [X; Y; Z; A; B; C]). Hence, no relevant difference arises when compared to the former solutions, albeit a very small and almost negligible motion about joints 1, 2,..., 5. Other solutions are less intuitive, and they introduce a small rotation of the other joints. For instance, the third TCP pose is [−350mm; −350mm; 500mm; 45°; $-5.625°$; 0°] and it produces a maximum motion for each joint of:

$$\begin{bmatrix} 0.23° \\ 0.33° \\ 0.37° \\ 4.04° \\ 3.95° \\ 45.14° \end{bmatrix} \tag{5.5}$$

One may also want to study what happens when the full TCP range can be swept: the third rotation $-45° \leq C \leq 45°$ was introduced. As far as the X-Y circle was concerned, the best TCP pose found is [−450mm; −410mm; 610mm; 45°; $-11.25°$; 0°], with maximum motions of:

$$\begin{bmatrix} 0.79° \\ 1.17° \\ 1.32° \\ 8.06° \\ 8.02° \\ 45.34° \end{bmatrix} \tag{5.6}$$

Thus, the bottleneck is still the last joint, with a slightly larger maximum rotation. Indeed, enlarging the allowed range makes it harder for the optimizer to find exactly the optimum, hence more initial poses or iterations would be necessary. All the other solutions are similar or irrelevant.

A similar result holds for the X-Z circle, with the full TCP range. The optimum TCP pose found is [120mm; −310mm; 350mm; $-61.875°$; $-5.625°$;

5.625°]. It is worth noting that here $\sqrt{120^2 + 310^2} \simeq 332 < 350$, and maximum motions are:

$$\begin{bmatrix} 0.40° \\ 1.90° \\ 2.80° \\ 5.80° \\ 5.74° \\ 45.38° \end{bmatrix} \tag{5.7}$$

Still, the bottleneck is the sixth joint. All the other solutions are less relevant.

5.1.2 A semicircle on the YZ plane

Suppose the robot has to grind, mill or perform any other operation on the internal surface of a large semicircular container. For this purpose, a trajectory like that reported in figure 5.3 was designed. In particular, it is a circular arc belonging to a plane perpendicular to the X axis of the robot frame. Its radius was designed to be large enough to put the algorithm under strain, and study what happens when the true optimal is at the very limit of the allowed TCP range, or even outside of it.

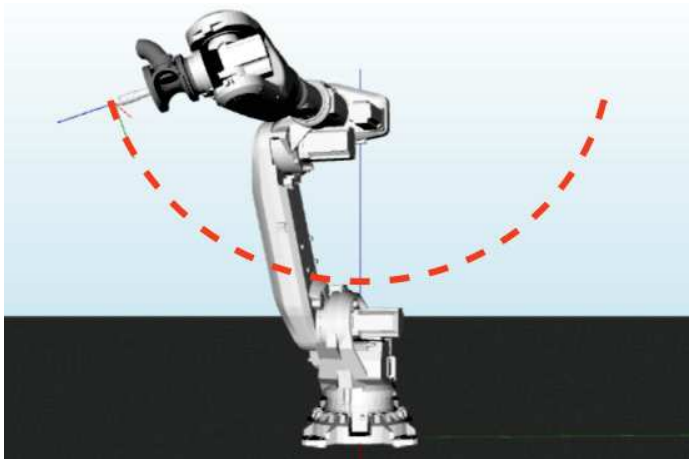


Figure 5.3: A large semicircular trajectory to evaluate velocity optimization algorithms. It is a circle with radius $\simeq$ 800mm

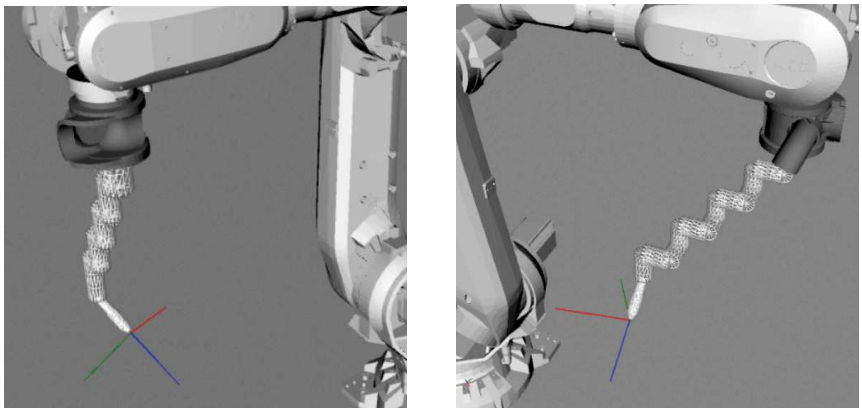
By plugging directly the welding torch on the flange and forcing the trajectory, that is divided in 15 points, the resulting maximum motions

are:

$$\begin{bmatrix} 3.13° \\ 0.88° \\ 5.75° \\ 11.0° \\ 1.52° \\ 2.02° \end{bmatrix} \text{ the time needed is: } \begin{bmatrix} 0.031\text{s} \\ 0.010\text{s} \\ 0.064\text{s} \\ 0.065\text{s} \\ 0.013\text{s} \\ 0.011\text{s} \end{bmatrix} \tag{5.8}$$

Hence, the bottlenecks are joints 3 and 4. Actually, joint 3 is slow, being its maximum velocity $90°/s$, while the other one is almost doubly faster ($170°/s$).

It is interesting to see what the random-search algorithm with standard weights finds here, considering the standard TCP range. This time, all quantities are referred to the base of the tool, not to its tip. The first "optimal" pose returned is [20mm; −350mm; 700mm; 67.5°; 45°; 5.625°], while the fourth one is a little further: [350mm; −350mm; 390mm; −28.125°; 16.875°; 11.25°], and they are depicted in figure 5.4.



(a) (b)

Figure 5.4: Two sample tool poses for the large circular arc. (5.4a) is [20mm; −350mm; 700mm; 67.5°; 45°; 5.625°], while (5.4b) is [350mm; −350mm; 390mm; −28.125°; 16.875°; 11.25°]. The main difference is that, in the former case, the robot tends to keep itself behind the trajectory, while the latter forces it to get at the center of the circumference that the arc belongs to

The maximum motions for each one are:

$$\begin{bmatrix} 4.58° \\ 1.05° \\ 4.18° \\ 9.55° \\ 1.82° \\ 4.13° \end{bmatrix} \text{ and } \begin{bmatrix} 0.45° \\ 0.58° \\ 1.79° \\ 9.63° \\ 1.48° \\ 1.91° \end{bmatrix} \tag{5.9}$$

It is clear to see that the algorithm tends to make use of the fourth joint, that is the second fastest one, just behind the sixth joint. Before analyzing the results of genetic optimization, it is worth noting that the latter pose returned makes all the joints but J4 almost still, that is charged with just a slightly bigger angle to sweep. This fact suggests that using square weights may lead directly to a solution that makes use of this axis only.

The GA returns as first tool pose [−350mm; −210mm; 700mm; 28.125°; 45°; 0°], whose maximum motions are similar to the first results of the random-search algorithm (5.9). The differences are the second and third best poses, that are similar to the last pose returned by the previous method, and the fourth best pose that enhances the motion of joint 4: the pose is [300mm; 110mm; 700mm; −5.625°; 45°; −11.25°] and it results in maximum motions of:

$$\begin{bmatrix} 0.35° \\ 0.07° \\ 0.48° \\ 10.0° \\ 0.08° \\ 0.15° \end{bmatrix} \text{ the time needed is: } \begin{bmatrix} 0.004\text{s} \\ 0.001\text{s} \\ 0.005\text{s} \\ 0.059\text{s} \\ 0.001\text{s} \\ 0.001\text{s} \end{bmatrix} \tag{5.10}$$

It is clear that, when considering possible acceleration limits of the joints, this solution may possibly be the best one since all the joints are still but one, that instead will have almost equal PTP motions. In figure 5.5 the overall motion is shown: almost all the links do not move.

Fastest joints selection

As far as the second version of the velocity optimization algorithm is concerned, when considering the standard random-search algorithm the first tool pose found is [350mm; −350mm; 380mm; −16.875°; 16.875°; 16.875°],

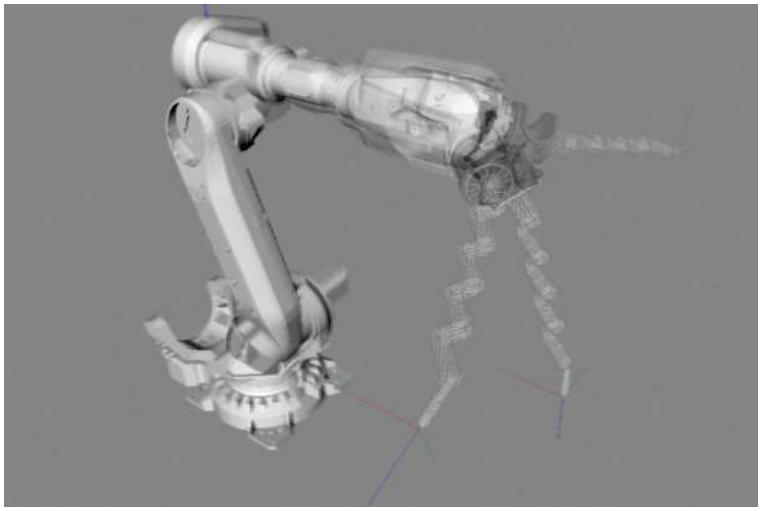


Figure 5.5: Three superimposed screenshots of the robot tracking the circular arc with the fourth optimal pose found by the genetic algorithm, standard weights

that is very similar to the third pose too. Maximum motions are:

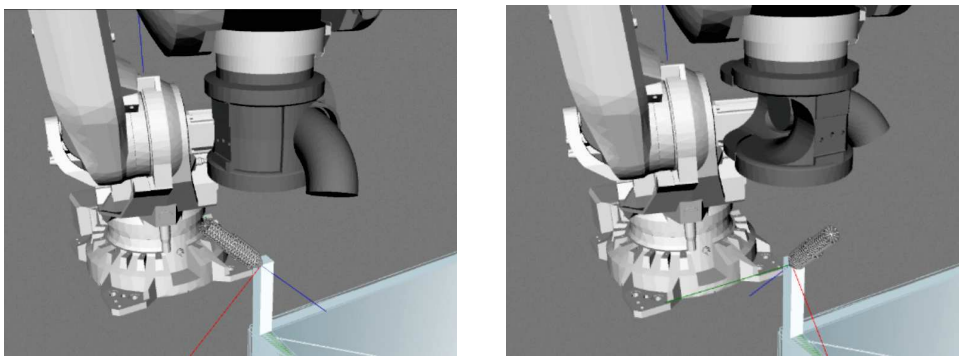
$$\begin{bmatrix} 0.44° \\ 0.45° \\ 1.50° \\ 9.68° \\ 1.37° \\ 1.66° \end{bmatrix} \tag{5.11}$$

Again, this shows that, considering the allowed TCP range, the best solution is to use almost exclusively the fourth joint, making its revolute axis passing almost through the center of the circumference to be tracked. Similar results are returned also by the genetic version of the optimization method.

Bottom line, what the algorithms show is a suggestion of TCP placement. Of course, the objective function may have many local minima and finding the most suitable one among the options is something that a human user may be required to do. Up to now, the algorithm with squared weights seems to provide better results. However, sometimes a counter-intuitive pose is suggested, as shown by the following trial.

5.1.3 Welding a right angle

As further example, the robot is asked to track a 90° sharp angle. Indeed, in real welding applications, it is not uncommon to find profiles that make a sudden bend: in this cases, the power of the torch is lowered to allow the robot to perform a wider motion. In other scenarios, the TCP is set to lie exactly on the center of the axis of the last joint, so as to move from one straight line to the next one just by a rapid motion of this latter one. This is shown in fig. 5.6.



(a) (b)

Figure 5.6: Start and end points of the 90° welding trajectory. No connection between flange and tool is shown for clearness

Hence, a 7-points trajectory was created. The only difference among the points was a 15° rotation about an axis parallel to the Z axis of the robot reference frame. the [X;Y;Z] position is exactly the same for all the points and it corresponds to $[1513\text{mm}; -330.4\text{mm}; 874\text{mm}]$. As previously stated, in many cases the TCP is set to lie on the flange normal axis. Hence, as reference tool a welding torch with TCP at [0mm; 0mm; 300mm; 0°; −45°; 0°] was created (it is worth noting that the trajectory must be tracked by keeping a constant 45° angle with respect to the ground). The resulting maximum motions are, as expected:

$$\begin{bmatrix} 0° \\ 0° \\ 0° \\ 0° \\ 0° \\ 15° \end{bmatrix} \tag{5.12}$$

As usual, the results of random-search algorithm with standard weights first. This time, however, the allowed range for rotations was reduced to

$[\pm 90°;\ -45° \dots 22.5°;\ \pm 22.5°]$, in order to help the algorithm not to try too "twisted" tools.

The first pose found is [−110mm; 110mm; 200mm; −22.5°; −5.625°; 22.5°], referred to the TCP frame, and it allows to obtain an interesting result, albeit evidently non-optimal. Indeed, maximum motions in this case are:

$$\begin{bmatrix} 3.61° \\ 2.76° \\ 4.33° \\ 7.94° \\ 4.55° \\ 9.14° \end{bmatrix} \text{ the time required is: } \begin{bmatrix} 0.036\text{s} \\ 0.031\text{s} \\ 0.048\text{s} \\ 0.047\text{s} \\ 0.038\text{s} \\ 0.047\text{s} \end{bmatrix} \tag{5.13}$$

It is evident that joints 3, 4 and 6 require the same maximum time to perform a PTP motion. If all the joints could be capable of instantaneous accelerations, it is clear that joint 6, with an angle taken from 15 to 9 degrees, would be faster in tracking the set of points when compared to the previous, standard tool. Moreover, it is worth recalling that the standard tool requires the last joint to perform a 15° motion for each step, while now each sub-motion is different and it may even be lesser than 9.14°: indeed, the last but one motion of joint 6 in this case is just 8°. Hence, what really makes a difference here is the acceleration of each single joint, that is not provided.

All the other poses have very similar results.

Again, the GA leads to similar poses also in this case. However, the first run already suggests what the optimal solution of the squared-weights algorithms will be. Indeed, the third pose returned by a test-run of the genetic optimizer was: [0mm; 20mm; 210mm; −33.75°; −22.5°; 16.875°]. Maximum motions in this case are:

$$\begin{bmatrix} 2.19° \\ 1.97° \\ 2.58° \\ 5.72° \\ 3.50° \\ 10.81° \end{bmatrix} \text{ the time required is: } \begin{bmatrix} 0.022\text{s} \\ 0.022\text{s} \\ 0.029\text{s} \\ 0.034\text{s} \\ 0.029\text{s} \\ 0.057\text{s} \end{bmatrix} \tag{5.14}$$

Even if the resulting motion of the sixth joint is now slightly larger, it is evident that all the other joints require less time, and accelerations and decelerations may now impact less on a real robot. In fact, the greatest effort is that of the last joint, but all the others contribute in some sense to reduce the angle to be swept. Figure 5.7 represents the tool.

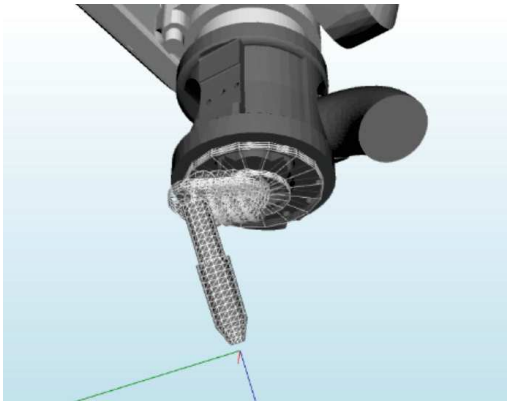


Figure 5.7: A sample tool for welding the right angle, returned by the standard-weights genetic algorithm. It is worth noting that the tool tip is oriented towards the sixth axis, i.e. the center of the flange

Fastest joints selection

As far as the random-search method is concerned, when using squared weights the TCP does not move too far from the flange, nor rotations are too enhanced. For instance, the first TCP pose returned after running the optimizer once was: [10mm; 10mm; 200mm; 50.625°; −16.875°; 16.875°]. Actually, each of the the four pose returned had the same rotation about X (i.e. rotation C), and the same offset in Z. X and Y do not move too far and, moreover, independently on the value of rotation A, also rotation B keeps between −16.875° and −22.5°. Maximum motions produced with the first configuration are:

$$\begin{bmatrix} 2.25° \\ 1.77° \\ 2.37° \\ 6.63° \\ 3.61° \\ 10.59° \end{bmatrix} \tag{5.15}$$

So it is clear that the previous optimization with GA and standard weights was able to spot in advance a possible configuration that makes use of the last joint, but also tries to move the others to decrease the angle to be swept, and hence time. It is interesting to notice that the first three joints keep almost still: being they mainly responsible for the position of the robot, they do not contribute as decisively as the other axes. Hence, the motion is just a matter of moving the wrist wisely. All the other configurations of the tool give similar results.

Genetic optimization in this case returns further tools: the first TCP pose returned is [100mm; 10mm; 250mm; 22.5°; −16.875°; 16.875°]. It is also interesting to compare its results with the fourth pose, that is [−210mm; −150mm; 300mm; −28.125°; −39.375°; 22.5°], that keeps the tool tip further away. The first tool gives as maximum motions:

$$\begin{bmatrix} 1.91° \\ 1.61° \\ 2.04° \\ 6.8° \\ 3.68° \\ 11.25° \end{bmatrix} \tag{5.16}$$

Thus, reducing even more the effort of the first joints. The fourth one instead:

$$\begin{bmatrix} 1.78° \\ 2.34° \\ 2.7° \\ 4.16° \\ 3.91° \\ 12.73° \end{bmatrix} \tag{5.17}$$

This last option is interesting because, while all the other tools follow the structure also shown in figure 5.7, namely the TCP points towards the rotation axis of last joint, in this case the tool is set to head far from the flange.

Now, choosing the right tool according to the shown results may be tricky and not immediate. Probably, for many applications keeping the torch nozzle exactly on the sixth axis is still easy and intuitive. It would be actually very interesting to analyze how the acceleration limits of the joints impact on the optimal solutions found.

5.2 Range optimization

Another target regarding the optimization of the tool is joints range optimization. The end-effector must be modified in order to make the joints work far enough from singularities, but also from their mechanical limits. This makes the robot able to cope with unforeseen events and obstacles: having enough space for each joint to move means that the chances to find a free trajectory are higher. Moreover, being far from singularities means that there are not directions that require a too big motion of the joints.

To test the developed algorithm, 3 test trajectories were developed (where actually trajectories are set of objects to be picked up). The first one is just a single point located in front of the robot ($Y = 0$), used to verify the effectiveness of the algorithm, while the other ones have been designed thanks to the functionalities of the company's software, meaning the so-called "pick tester": it allows to create some objects inside a box, and it designs a set of trajectory points that the robot must follow to grab them. To guarantee a collision-free motion, a series of pick positions is tested for each item. For each objects 5 trajectory points, starting from home position, are found.

In order to optimize not only the configuration of the joints while the robot is picking the object, but also to take into account the motion toward it, the last and third but last points of the approaching motion were included in the other two test trajectories, that comprise three and four objects, respectively.

5.2.1 1 point trajectory

The unique point of this trajectory is [1972.22mm; 0mm; 252.78mm; 90°; 0°; 169°], referred to the robot frame, reported in figure 5.8. There, it is possible to notice both the tool used for this task, that is an electromagnet to pick up metallic items (also depicted in figure 5.9), and the fact that in this position the robot is close to a singular configuration: indeed, joint 5 makes previous and subsequent links almost aligned.

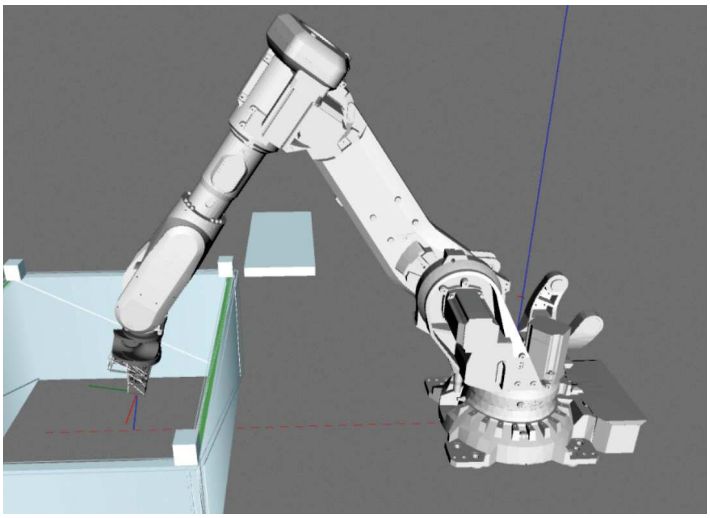


Figure 5.8: The 1-point trajectory to test range optimization algorithms. The robot TCP is exactly on the point

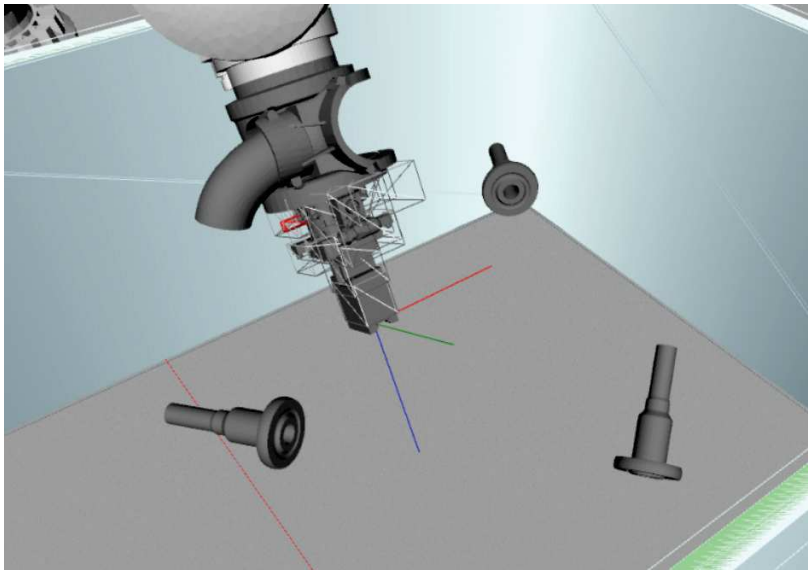


Figure 5.9: A view of the objects used for pick testing and the electromagnet (M130) attached to the flange. Each item can be picked in many positions

First, results of the random-search version of the optimizer. It is worth noting that the TCP space is limited: indeed, Y is set to be identically 0, and no rotation about Z, X axis are permitted. These are not restrictive assumptions, since the point is exactly in front of the robot, hence it makes no sense at all to design a tool that can lie to the right or left of the flange.

The resulting poses are [−270mm; 0mm; 700mm; 0°; −22.5°; 0°], that is actually found twice and shown in figure 5.10, [−160mm; 0mm; 700mm; 0°; −16.875°; 0°] and [−60mm; 0mm; 700mm; 5°; −11.25°; 0°]. All of them encode the TCP frame with respect to the flange frame.

To evaluate the improvement of the optimizer, it is important to compare the joints configuration before and after[2], i.e. with the standard tool

[2]In the following, the initial configuration and the new one are reported. The improvement for each joint is just how much that axis has moved close to the center of the gaussian (refer to chapter 4.2 for details). If improvement is negative, it is actually a worse configuration. For instance, if joint 1 moves from 8° to −5°, the improvement is 3°, since it reduced the distance from the best value that is 0°. Moreover, the best configuration is always the one with the greatest peak

and the new tool:

$$\begin{bmatrix} 0° \\ 38.0° \\ 17° \\ 0° \\ 24° \\ 0° \end{bmatrix}, \text{ that becomes: } \begin{bmatrix} 0° \\ 25.7° \\ 6.8° \\ 0° \\ 70° \\ 0° \end{bmatrix} \text{ with improvement: } \begin{bmatrix} 0° \\ 12.3° \\ 10.2° \\ 0° \\ 37° \\ 0° \end{bmatrix} \quad (5.18)$$

It is important to recall the best angle of the joints: they are 0° for 1, 4 and 6, 10° for 2, −10° for 3 and 65° for 5. Probably, the algorithm cannot reach these values because of the restricted TCP range (it is worth noting that all the returned poses set Z to the limit, i.e. 700mm). Nevertheless, the improvement of each joint that gets a better configuration is between 10 and 40 degrees.

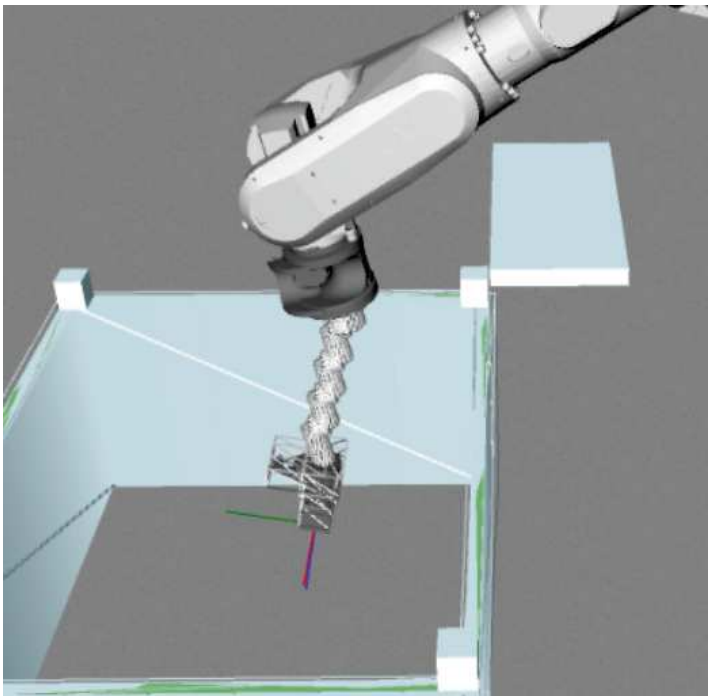


Figure 5.10: The best tool returned by the random-search algorithm for the 1-point test, with the best tool

As far as genetic optimization is concerned, the best poses returned are [−140mm; 0mm; 700mm; 0°; −16.875°; 0°], [−130mm; 0mm; 690mm; 0°; −16.875°; 0°], returned twice, and [−130mm; 0mm; 670mm; 0°; −16.875°;

0°]. It is possible to see that they are not too far from the optimum found by the random-search algorithm. The best pose gives:

$$\begin{bmatrix} 0° \\ 38.0° \\ 17° \\ 0° \\ 24° \\ 0° \end{bmatrix}, \text{ that becomes: } \begin{bmatrix} 0° \\ 21.4° \\ 2.7° \\ 0° \\ 71.8° \\ 0° \end{bmatrix} \text{ with improvement: } \begin{bmatrix} 0° \\ 16.6° \\ 14.3° \\ 0° \\ 34.2° \\ 0° \end{bmatrix} \tag{5.19}$$

In this case, joints 2 and 3 reach even a better configuration, despite the fifth joint must give up some degrees of improvement. Hence, solutions from the two methods are almost similar here.

5.2.2 3 objects trajectory

After having proved that the algorithm works and leads to a meaningful result, it is necessary to apply it to a more realistic scenario: multiple objects randomly placed in a bin that have to be picked up, but keeping the robot in the best possible configuration. Three items (figure 5.11) were randomly instantiated in a box, to this end. For each object, both the picking position of the tool and an intermediate position just above the piece were included in the optimization trajectory. The optimization variable were set to be the TCP pose, and its range as the standard one.

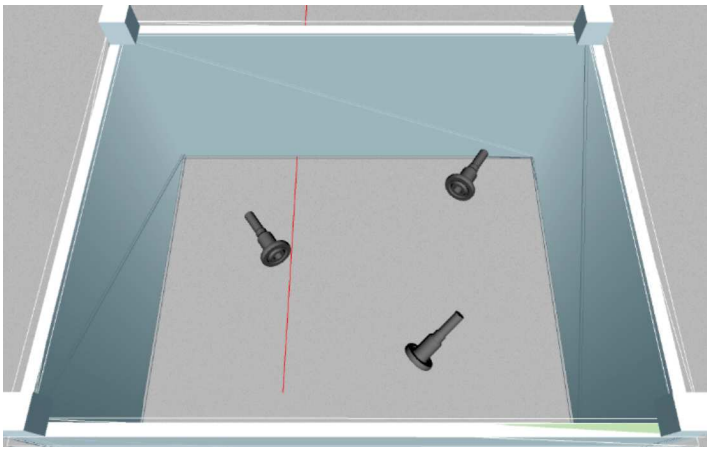


Figure 5.11: Three objects test for range optimization

By plugging the M130 magnetic gripper on the flange and studying the configuration taken when trying to reach the points, it is evident that the robot tends again to a singular configuration since joint 5 is close to 0,

hence aligning the axes of joints 4 and 6, as shown in figure 5.12. Actually the fifth axis is $-17.8°$, but given that the robot may perform a complicated motion to detach the piece (in case other items not to be hit are nearby), it is better to lower the possibilities of passing through a singularity.

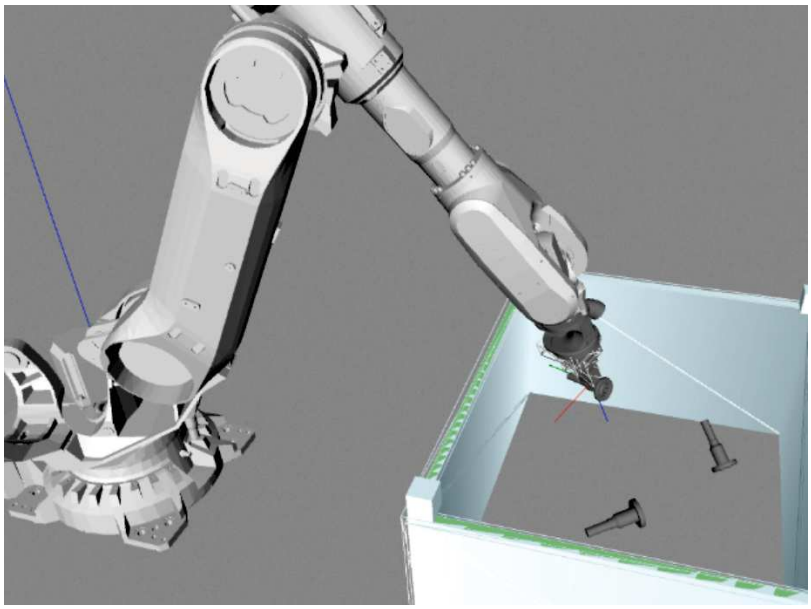


Figure 5.12: Almost singular configuration for the three objects case

Random search returns 4 slightly different poses ([−230mm; 210mm; 700mm; −78.75°; 0°; 22.5°]; [−80mm; 250mm; 700mm; −90°; 0°; 16.875°];[−230mm; 140mm; 700mm; −45°; 0°; 16.875°]; [−310mm; 360mm; 660mm; −90°; −5.625°; 28.125°]). In the following, only the first one is going to be treated since all the others return very similar results.

Taking just 3 points out of 6 of the whole set (i.e. the points corresponding to the TCP touching the piece and ready for picking it up), the

configuration changes in this way:

$$\begin{bmatrix} 1.4^\circ \\ 60.8^\circ \\ -18.0^\circ \\ 63.0^\circ \\ 54.9^\circ \\ 20.9^\circ \end{bmatrix}, \text{ that becomes: } \begin{bmatrix} -5.5^\circ \\ 34.1^\circ \\ 11.4^\circ \\ 88.1^\circ \\ 55.0^\circ \\ 79.0^\circ \end{bmatrix} \text{ with improvement: } \begin{bmatrix} -4.1^\circ \\ 26.7^\circ \\ -13.4^\circ \\ -25.1^\circ \\ 0.1^\circ \\ -58.1^\circ \end{bmatrix}$$
$$\begin{bmatrix} 17.7^\circ \\ 29.0^\circ \\ 22.8^\circ \\ 191.4^\circ \\ -17.8^\circ \\ 37.8^\circ \end{bmatrix}, \text{ that becomes: } \begin{bmatrix} 15.8^\circ \\ 22.6^\circ \\ 2.1^\circ \\ -12.7^\circ \\ 61.8^\circ \\ -47.2^\circ \end{bmatrix} \text{ with improvement: } \begin{bmatrix} 1.9^\circ \\ 6.4^\circ \\ 20.7^\circ \\ 178.7^\circ \\ 79.6^\circ \\ -9.4^\circ \end{bmatrix}$$
$$\begin{bmatrix} -2.6^\circ \\ 39.1^\circ \\ 7.1^\circ \\ 190.1^\circ \\ -36.2^\circ \\ 135.6^\circ \end{bmatrix}, \text{ that becomes: } \begin{bmatrix} -14.1^\circ \\ 25.1^\circ \\ 6.5^\circ \\ 23.4^\circ \\ 71.0^\circ \\ 18.5^\circ \end{bmatrix} \text{ with improvement: } \begin{bmatrix} -11.5^\circ \\ 14.0^\circ \\ 0.5^\circ \\ 166.7^\circ \\ 95.2^\circ \\ 117.1^\circ \end{bmatrix} \tag{5.20}$$

It is clear that the first pose gets evidently worse, while the others improve. To understand this, it is necessary to recall that the algorithm tries to find a unique pose of the TCP for three objects. Hence, bad pick positions are improved at the expense of already good ones. Indeed, the second and third positions present a fairly large improvement of all joints but one (and it is worth noting that joints 1 and 6 are respectively the third and first larger-range axes, so they can afford such a deterioration). The most critical joints are instead joints 3 and 5 (suffering from singularity) and joint 2: in the first position, it is clear that 3 is not that far from the best, so it gets worse to allow joint 2 to get close to its center, while axes 1, 4 and 6 worsen without getting too far. A very interesting situation regards the second and third position: they show even joints that improve by more than 100°.

The GA returns as best 4 TCP poses: [−90mm; 240mm; 700mm; −90°; 0°; 16.875°]; [−100mm; 240mm; 630mm; −90°; −5.625°; 22.5°];[60mm; −20mm; 680mm; −90°; 5.625°; 16.875°]; [−90mm; 5.625mm; 690mm; −90°; 5.625°; 22.5°], thus not that different from the ones returned by the other method.

The first TCP pose on the same 3 points gives the following results:

$$\begin{bmatrix} 1.4° \\ 60.8° \\ -18.0° \\ 63.0° \\ 54.9° \\ 20.9° \end{bmatrix}, \text{ that becomes: } \begin{bmatrix} -5.4° \\ 35.0° \\ 10.3° \\ 81.2° \\ 55.1° \\ 94.4° \end{bmatrix} \text{ with improvement: } \begin{bmatrix} -3.9° \\ 25.8° \\ -12.3° \\ -18.2° \\ 0.2° \\ -73.5° \end{bmatrix}$$
$$\begin{bmatrix} 17.7° \\ 29.0° \\ 22.8° \\ 191.4° \\ -17.8° \\ 37.8° \end{bmatrix}, \text{ that becomes: } \begin{bmatrix} 15.8° \\ 22.2° \\ 3.4° \\ -8.9° \\ 56.5° \\ -37.7° \end{bmatrix} \text{ with improvement: } \begin{bmatrix} 1.9° \\ 6.8° \\ 19.4° \\ 182.6° \\ 74.2° \\ 0.1° \end{bmatrix}$$
$$\begin{bmatrix} -2.6° \\ 39.1° \\ 7.1° \\ 190.1° \\ -36.2° \\ 135.6° \end{bmatrix}, \text{ that becomes: } \begin{bmatrix} -13.5° \\ 25.18° \\ 6.7° \\ 20.5° \\ 65.6° \\ 31.2° \end{bmatrix} \text{ with improvement: } \begin{bmatrix} -10.9° \\ 13.9° \\ 0.2° \\ 169.6° \\ 100.6° \\ 104.4° \end{bmatrix} \tag{5.21}$$

Comparing 5.20 and 5.21 it is clear to see that the improvements caused by the two methods are really similar, even if the final TCP poses may have not negligible differences. This again proves that the function to be optimized has a quite irregular and complex shape, but on the other side shows that the difference between the two algorithms may be very small if the initial poses are chosen appropriately.

5.2.3 4 objects trajectory

Another simulation with 4 pickable objects in different poses was carried out. The objects are depicted in fig. 5.13. The purpose of this test is to check what happens when the number of different picking positions to optimize increases, in terms of quality of the solution.

The first (and best) TCP pose returned by the random search algorithm is [−180mm; −200mm; 700mm; −67.5°; 22.5°; 22.5°]. The improvement of just the first two "pick positions" are shown in the following, since for the

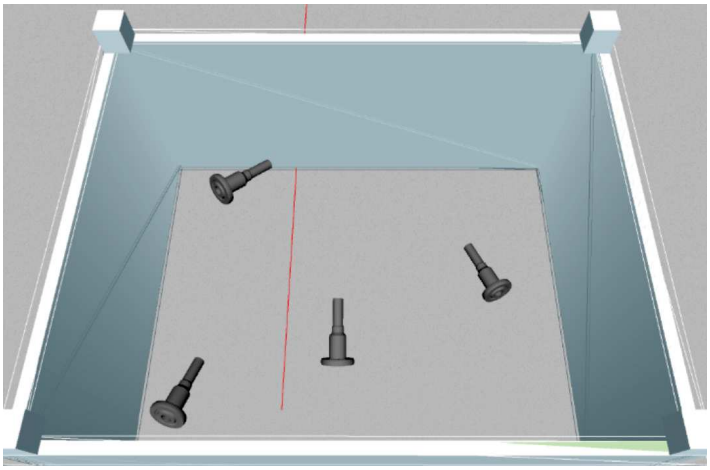


Figure 5.13: Four objects test for range optimization

others similar comments hold:

$$\begin{bmatrix} -9.2° \\ 54.2° \\ -17.1° \\ 30.7° \\ 32.0° \\ 158.8° \end{bmatrix}, \text{ that becomes: } \begin{bmatrix} -9.0° \\ 50.1° \\ -37.5° \\ -4.3° \\ 77.4° \\ -103.9° \end{bmatrix} \text{ with improvement: } \begin{bmatrix} 0.2° \\ 4.1° \\ -20.4° \\ 26.4° \\ 20.6° \\ 55.0° \end{bmatrix}$$
$$\begin{bmatrix} -3.5° \\ 70.9° \\ -41.0° \\ 35.0° \\ 83.8° \\ -8.3° \end{bmatrix}, \text{ that becomes: } \begin{bmatrix} -13.3° \\ 54.6° \\ -28.1° \\ 57.4° \\ 74.4° \\ 54.8° \end{bmatrix} \text{ with improvement: } \begin{bmatrix} -9.8° \\ 16.3° \\ 12.9° \\ -22.4° \\ 9.3° \\ -46.5° \end{bmatrix} \tag{5.22}$$

It is clear that, intuitively, increasing the number of objects the algorithms finds more difficulties in retrieving a TCP that may improve all the picking positions. It is possible to notice that the first TCP pose improves pretty much for all joints but 3 (one of the most critical ones, actually), while the second one is worsened except for joints 2, 3 and 5. Likely, the standard tool was already in a good position when picking this item, so it cannot do anything but worsen it to improve the other points, like the first one.

The GA returns as best TCP pose [−240mm; 250mm; 650mm; −90°;

$-5.625°$; $0°$], with the following results:

$$\begin{bmatrix} -9.2° \\ 54.2° \\ -17.1° \\ 30.7° \\ 32.0° \\ 158.8° \end{bmatrix}, \text{ that becomes: } \begin{bmatrix} -18.3° \\ 46.1° \\ -22.0° \\ 30.0° \\ 42.0° \\ -114.4° \end{bmatrix} \text{ with improvement: } \begin{bmatrix} -9.1° \\ 8.1° \\ -4.9° \\ 0.7° \\ 10.0° \\ 44.4° \end{bmatrix}$$

$$\begin{bmatrix} -3.5° \\ 70.9° \\ -41.0° \\ 35.0° \\ 83.8° \\ -8.3° \end{bmatrix}, \text{ that becomes: } \begin{bmatrix} -4.9° \\ 65.5° \\ -56.0° \\ 35.2° \\ 106.9° \\ -267.7° \end{bmatrix} \text{ with improvement: } \begin{bmatrix} -1.4° \\ 5.5° \\ -14.9° \\ -0.2° \\ -23.2° \\ -259.4° \end{bmatrix} \tag{5.23}$$

So it is clear that the second item is grabbed with a worse tool. This is again due to the characteristics of the developed genetic algorithm that, compared to the results of the other method in this scenario, appears less precise. However this fact also highlights that reducing the number of objects to pick would lead to a better situation.

5.3 Number of picks optimization

After velocity and range optimization, the focus of the work was shifted towards optimizing a more complex scenario: in a random bin picking application, a box may contain a very huge amount of pieces to be found, grabbed and placed correctly. Even if changing the tool for every batch of products may be unfeasible, it is nonetheless interesting to analyze possible solutions to the problem of selecting a better TCP, to avoid collisions with the environment or the items themselves. To this end, the random search algorithm was used, but at the end the PSA-based method was proven to give better results without selecting too "contorted" tools (refer to chapter 4.3 for more details).

One may ask why using pattern search only in this case: actually, the reason is that the high amount of pieces in the box increases computational time to assess a TCP pose, hence the advantages of using a fast random selection of the direction of optimization are lost. Instead, pattern search, despite it needs more time to look for all the best neighboring poses, goes directly to the possible best one. Moreover, in this case there is not just one function to optimize, but two ones: on one hand the main objective

function returns the number of successfully grabbed items (an integer); on the other hand, the second metric is the joints' values score, already used in range optimization, that discriminates between two tools with the same number of picked objects. In practice, the best tool also tries to find the best possible picking positions in terms of distance from singularity and range limits.

Before examining in depth the results, it is worth recalling how the developed algorithm works. First of all, the number of initial poses is not fixed: since improvement steps will be less, it is necessary to instantiate enough poses to equally span the whole TCP range, so every 10cm in X, Y and Z a new initial pose is created. At the same time, all poses will be such that the X and Y coordinates are non-negative: the range to explore is hence reduced and, since the last axis is no more studied in the range limits, this does not restrict too much the admissible pick-poses, given that joint 6 can now assume any angle.

For each initial tool pose, and for each item to be picked, a series of candidate picking positions on the object are tested, returning the one that grabs it without collisions and with the most centered joints configuration.

The pseudo pattern search algorithm then analyzes all the neighboring tool poses (25cm or $\frac{\pi}{8}$rad), selecting the one that gives the higher amount of picked objects or, if the number is the same, puts the robot in the best joints configuration. The total time required to find the optimal TCP (when the number of returned poses in 3) is around 1 hour and a half on a 8 GB Ram, Intel® Core™ i5 PC, for any of the following tests.

5.3.1 Test with 23 objects

The first test comprises 23 randomly generated objects in a box, as shown in fig. 5.14. They can be grabbed with the same magnet illustrated before.

The admissible TCP range is set to be ±300mm in X, Y and 200mm; 500mm in Z, while maximum rotations are kept the same. The first step is to analyze the capabilities of the robot when the tool is attached to the flange, and also when the standard tool provided by Euclid Labs is employed (that already has an offset, and previously shown in fig. 2.1). An item is said to be successfully picked if the robot can place its tool on a pick position without colliding or forcing a singular configuration. The algorithm selects the pick position that allows to meet these conditions, but also has the most favorable joints configuration.

By running the developed pattern search-based algorithm (it is worth noting that, differently from the other results, initialization and improve-

Figure 5.14: The first test with 23 objects

ment direction are not random), the optimal TCP pose is: [300mm; 300mm; 400mm; 0°; 0°; 22.5°], shown in fig. 5.15. The result of this tool is that all the objects but one are successfully picked. Initial and optimal tools effects are shown in fig. 5.16.

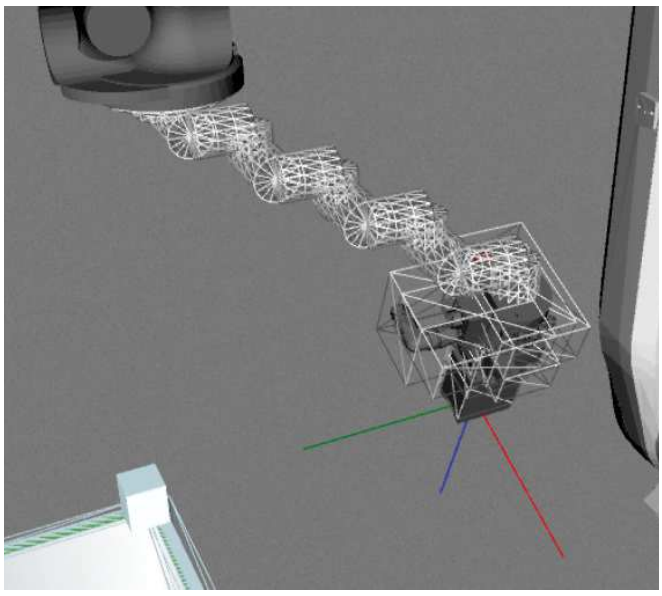
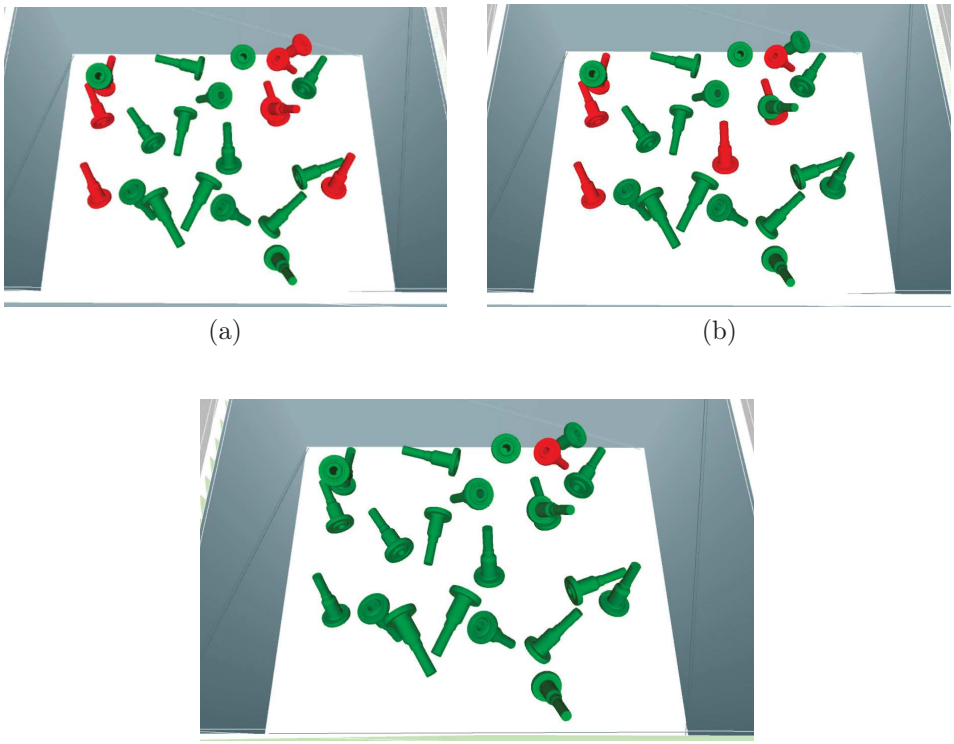


Figure 5.15: The first optimal tool for 23 objects test. Notice that the TCP is also slightly rotated

The increased distance of the TCP from the flange, together with the fact that each object may have many possible pick positions, is the key to the success of the optimal tool: two similar TCP poses in the space may force very different joints configurations, hence finding one that does not



(a) (b)

(c)

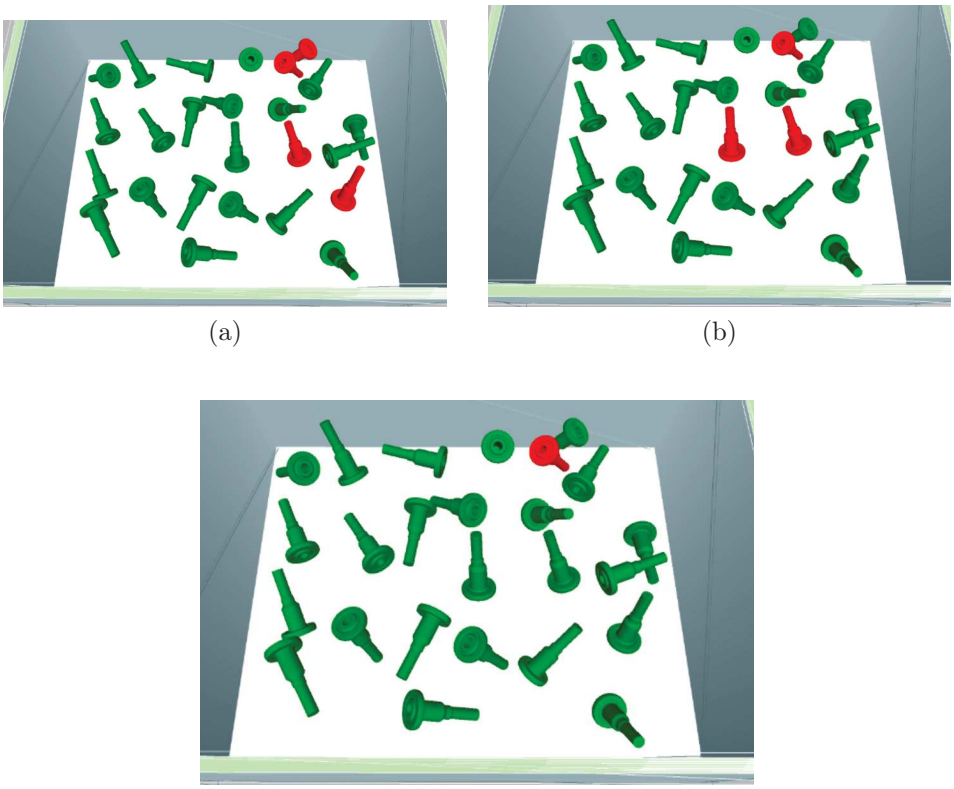
Figure 5.16: These three images show the capabilities of the basic and optimal tools when picking the 23 random items. 5.16a is for the magnet directly installed on the flange, that cannot pick 8 objects (the red ones), while 5.16b is the result of the tool with standard offset by Euclid Labs that is instead unable to grab 6 objects. 5.16c is instead the effect of the optimized tool: the only item that cannot be picked is, indeed, too close to the box and to other objects

collide may be easier.

5.3.2 Test with 25 objects

Another successful test makes use of 25 objects: many of them are similar to the previous ones, but in general they are more spread in the box, also closer to its walls, and their height from the ground is now more similar. The optimizer finds as optimal TCP pose: [175mm; 200mm; 500mm; 0°; −22.5°; 0°]. The entire scenario and results with standard and optimal tools are depicted in fig. 5.17. However, it is worth noting that, in principle, this optimal tool is such that all the objects can be grabbed: one is

excluded only because the tool would go underneath other items to pick it, so it would be physically unfeasible. Nonetheless, the second optimal tool provided ([300mm; 100mm; 500mm; 0°; 0°; 0°]) correctly returns that one as not reachable, while picking all the others. Pictures of the first and second optimal tools in fig. 5.18.



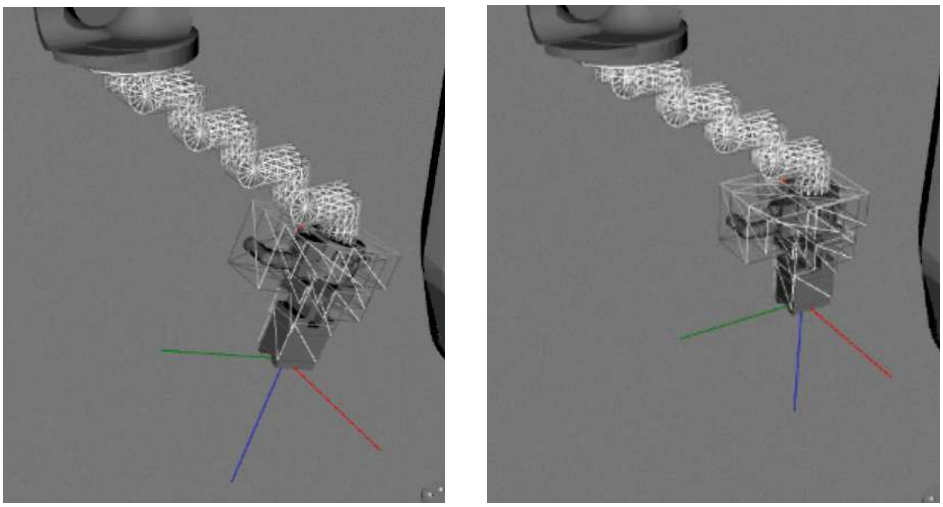
(a) (b)

(c)

Figure 5.17: These three images show the capabilities of the basic and optimal tools when picking the 25 random items. 5.17a is for the magnet directly installed on the flange, that picks 21 items, while 5.17b shows the 22 items picked by the standard Euclid Labs tool. 5.17c is instead the effect of the optimized tool: only 1 item cannot be picked

5.3.3 Real tests with a Nachi robot

All the previous tests were simulated. Hence, it is interesting to see the results of the developed optimizer for bin picking in a real environment. To this end, a robotic cell with a smaller Nachi MZ12-01 robot and a box



(a) (b)

Figure 5.18: The first and second optimal tools for 25 objects test. In this case the second tool is closer to the flange and not rotated

with real scanned items was built. The gripper was different too, and can be seen in fig. 5.19.

For this scenario, the box with real items was scanned multiple times and data were fed to the hosting company's software. Hence, 3 tests with slightly different objects were used for optimization.

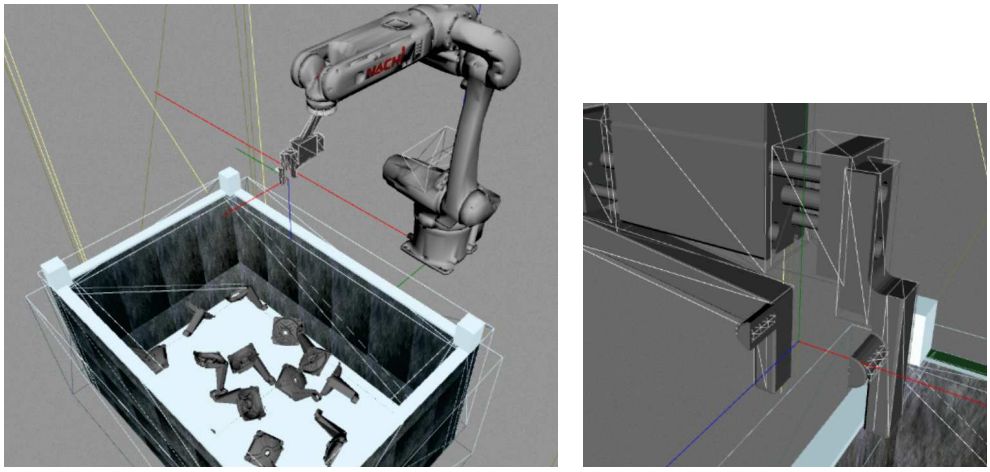
As usual, before optimizing the tool, it is necessary to test the capabilities of the standard tool, that already has a small offset. The three tests results are depicted in fig. 5.20.

Then, it is important to redesign the connection link between flange and tool base: indeed, since the Nachi robot is smaller, also this component must be thinner. The final diameter of the cylinders making up the link is now equal to 3cm. Moreover, as admissible TCP range, X and Y are reduced to ±300mm, and Z to 150mm; 450mm, while rotations are kept the same. The addressed variable is the TCP pose and not the tool base pose, while the optimization algorithm is again the pseudo-pattern search method.

As far as range limits are concerned, they are drawn from both the official manual of the robot and the specific cell software limits:

1. Joint 1 is allowed to rotate from $-90°$ to $90°$;

2. Joint 2 from $-80°$ to $180°$;



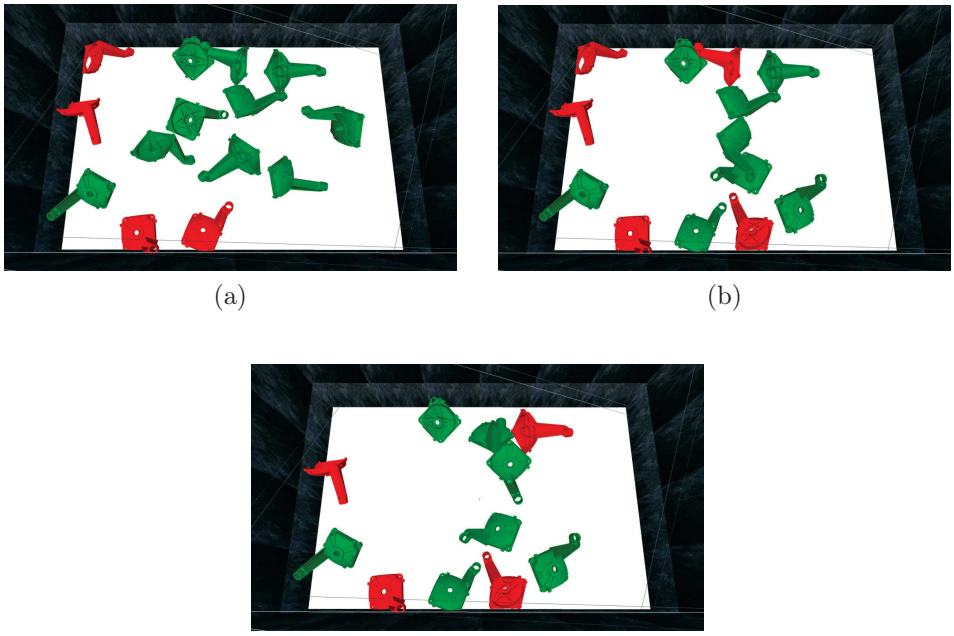
(a) (b)

Figure 5.19: The real cell with Nachi robot and its new gripper. Notice that now the gripper has two possible TCPs: 5.19b shows both (the main gripper and the smaller one on the side)

3. Joint 3 from $-82°$ to $120°$, but causes a singular configuration at $\approx 83°$;
4. Joint 4 from $-190°$ to $190°$;
5. Joint 5 from $-140°$ to $140°$, but causes a singular configuration at $0°$;
6. Joint 6 from $-360°$ to $360°$.

The optimization step on all the three scans leads to interesting results: the first pose of the TCP is the same for all of them, namely [300mm; 300mm; 150mm; 0°; 0°; 0°], while the second and third poses change. This is probably due to the fact that many items are the same. It is also interesting to see that, also in this case, the best TCP pose lies far from the flange in X and Y, but close to it in Z, as already commented upon. Results in fig. 5.21.

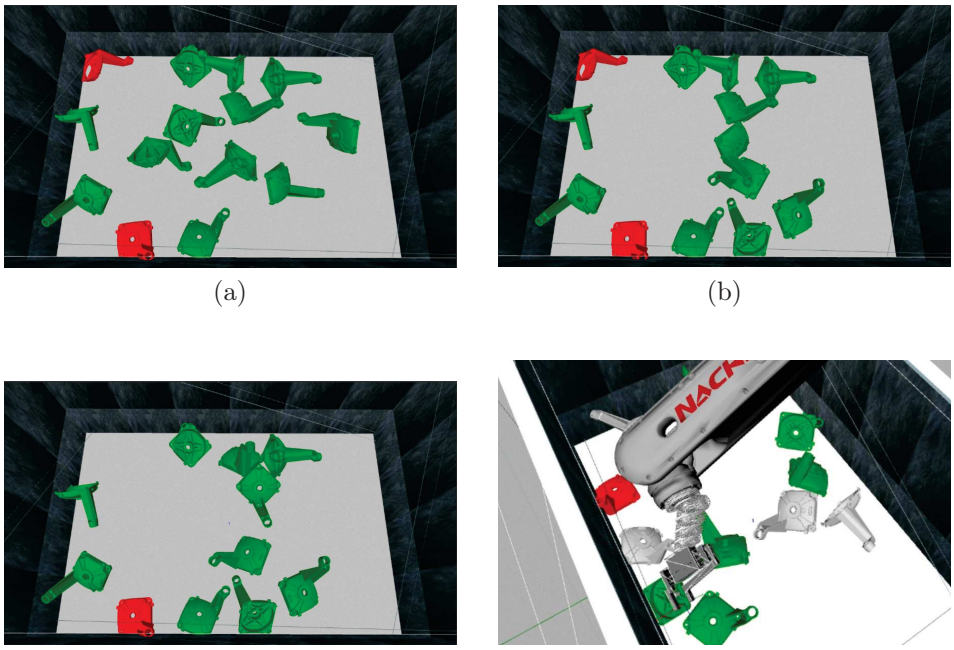
It is easy to see that the number of unreachable objects (because of distance or collisions) is drastically reduced. The items that still cannot be grabbed are always the same: probably, this is due to a collision caused by the tool itself. By comparing figures 5.20 and 5.21, it is clear that the most critical objects are those close to the box walls: indeed, the robot links may need to pass through them to grab the item. With a further TCP it is possible to find a pick position compatible with such locations.



(a) (b)

(c)

Figure 5.20: Capabilities of the standard tool for Nachi robot in the real environment. The tool is able to pick all but four items in first and third scan-based tests, and all but five in the second one. Both TCPs are used.



(a) (b)

(c) (d)

Figure 5.21: Capabilities of the optimal tool for Nachi robot in the real environment. The tool is able to pick all but two items in first and second scan-based tests, and all but one in the third one. Both TCPs are used. in 5.21d the optimal tool grabs an item with the second gripper

Chapter 6

Conclusions and future developments

In this work some techniques for TCP optimization have been developed and tested, together with some possible metrics. Due to the lack of prior research upon this area, only some aspects of the entire topic have been explored. For instance, velocity and range optimization show promising results, but a work that focuses just on them is needed to fully analyze the possibilities.

Indeed, as far as velocity optimization is concerned, the found tools give the user a valuable intuition on where the real ones should be installed to obtain the fastest motion with smaller joints angle. However, in further works it may be necessary to take into account also acceleration profiles, velocity limits in particular working conditions, and maybe also a more refined analysis upon non-idealities involving the motors.

Working range optimization needs instead to be extended to other scenarios. An interesting study may be the choice of the right scoring function (linear, gaussian, generalized normal or others) to obtain better results. Nonetheless, for deterministic, fixed motions the developed optimizer may be a useful tool to help engineers to find a better TCP placement to avoid singularities or getting stuck in joints limits.

As far as the maximimization of the number of picked items applied to bin picking is concerned, it is probably more practical, given also the effort that has been put in finding an algorithm returning a good, feasible pose. It was nonetheless the aspect that gave birth to the whole work. Despite time requirements for this kind of optimization, it may give a valuable help to identify the general shape of a tool when working on bin picking.

As already stated, the main difficulty in applying these results is given by the unavailability of devices that can be installed between flange and

tool, able to move the latter in order to find a better TCP pose. However, designing many tools to be installed for different tasks may be, for now, enough.

Concerning the optimization algorithms, genetic and pattern search methods show good results, as well as the random search algorithm developed to speed up iterations and convergence. However, a plethora of possible alternatives is available, hence a comparison between different options may be very useful.

Appendix A

Symmetric Generalized Normal Distribution

A symmetric generalized normal distribution is a continuous probability distribution driven by two parameters α and β. Its PDF is:

$$\frac{\beta}{2\alpha\Gamma(1/\beta)} e^{-(|x-\mu|/\alpha)^{\beta}} \tag{A.1}$$

Where $\Gamma(x) = \int_0^{\infty} t^{x-1} e^{-t} dt$ is the Gamma function, μ is the mean value and α, β are the two scale and shape real parameters, respectively. For $\beta = 2, \sigma^2 = \alpha^2/2$, it corresponds to the gaussian distribution with mean μ and standard deviation σ. In the algorithms proposed in this work, the peak of this distribution is always 1. Therefore, to analyze the effect of the parameters α and β, just the exponential part of eqn. A.1 is going to be considered.

In figures A.1, A.2 the effects of changing the two parameters α, β are visible.

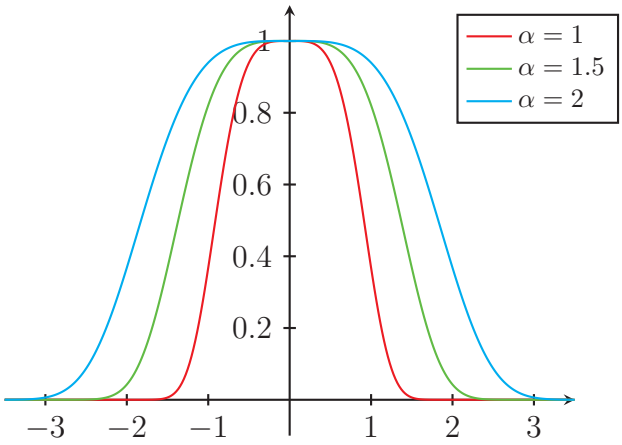


Figure A.1: Effects of α on a symmetric generalized normal distribution. All PDFs have peak = 1. As α increases, the effect is that of increasing the variance of a gaussian: the distribution becomes wider. $\beta = 4$.

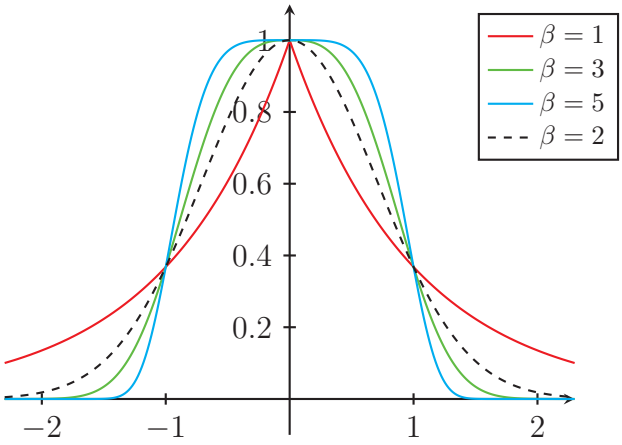


Figure A.2: Effects of β on a symmetric generalized normal distribution, compared with a standard gaussian distribution (dashed, black). All PDFs have peak = 1. As β increases, the peak becomes wider and the descent steeper. $\alpha = 1$, that corresponds to a gaussian with variance $\sigma^2 = 0.5$.

Bibliography

[1] F. Stuhlenmiller, S. Weyand, J. Jungblut, L. Schebek, D. Clever, and S. Rinderknecht, "Impact of cycle time and payload of an industrial robot on resource efficiency," *Robotics*, vol. 10, no. 1, 2021.

[2] N. Barnett, D. Costenaro, and I. Rohmund, "Direct and indirect impacts of robots on future electricity load," in *2017 ACEEE Summer Study on Energy Efficiency in Industry*, 2017.

[3] T. Sobh and D. Toundykov, "Optimizing the tasks at hand [robotic manipulators]," *Robotics & Automation Magazine, IEEE*, vol. 11, pp. 78–85, 01 2004.

[4] J.-Y. Park, P.-H. Chang, and J.-Y. Yang, "Task-oriented design of robot kinematics using the grid method," *Advanced Robotics*, vol. 17, no. 9, pp. 879–907, 2003.

[5] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl, "Time-energy optimal path tracking for robots: a numerically efficient optimization approach," in *2008 10th IEEE International Workshop on Advanced Motion Control*, pp. 727–732, 2008.

[6] C. Hansen, J. Öltjen, D. Meike, and T. Ortmaier, "Enhanced approach for energy-efficient trajectory generation of industrial robots," in *2012 IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 1–7, 2012.

[7] A. Mohammed, B. Schmidt, L. Wang, and L. Gao, "Minimizing energy consumption for robot arm movement," *Procedia CIRP*, vol. 25, pp. 400–405, 2014. 8th International Conference on Digital Enterprise Technology - DET 2014 Disruptive Innovation in Manufacturing Engineering towards the 4th Industrial Revolution.

[8] S. Liu, Y. Wang, X. V. Wang, and L. Wang, "Energy-efficient trajectory planning for an industrial robot using a multi-objective optimi-

sation approach," *Procedia Manufacturing*, vol. 25, pp. 517–525, 2018. Proceedings of the 8th Swedish Production Symposium (SPS 2018).

[9] A. Makhal and A. K. Goins, "Reuleaux: Robot base placement by reachability analysis," 2017.

[10] O. Porges, R. Lampariello, J. Artigas, A. Wedler, C. Borst, and M. A. Roa, "Reachability and dexterity: Analysis and applications for space robotics," 05 2015.

[11] Q. Fan, Z. Gong, B. Tao, Y. Gao, Z. Yin, and H. Ding, "Base position optimization of mobile manipulators for machining large complex components," *Robotics and Computer-Integrated Manufacturing*, vol. 70, p. 102138, 2021.

[12] A. Nektarios and N. A. Aspragathos, "Optimal location of a general position and orientation end-effector's path relative to manipulator's base, considering velocity performance," *Robotics and Computer-Integrated Manufacturing*, vol. 26, no. 2, pp. 162–173, 2010.

[13] N. A. Aspragathos and S. Foussias, "Optimal location of a robot path when considering velocity performance," *Robotica*, vol. 20, no. 2, pp. 139–147, 2002.

[14] `https://www.yaskawa.eu.com/products/robots/glossary`.

[15] T. Weingartshofer, C. Hartl-Nesic, and A. Kugi, "Optimal tcp and robot base placement for a set of complex continuous paths," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9659–9665, 2021.

[16] C. Reeves, "Genetic algorithms," *Handbook of Metaheuristics*, vol. 146, pp. 109–139, 09 2010.

[17] X. Zhang, Q. Zhou, and Y. Wang, "An efficient pattern search method," *Journal of Applied Mathematics and Physics*, vol. 1, no. 4, pp. 68–72, 2013.

[18] W. Bu, Z. Liu, and J. Tan, "Industrial robot layout based on operation sequence optimisation," *International Journal of Production Research*, vol. 47, no. 15, pp. 4125–4145, 2009.

List of Tables

List of Figures