University of Padua
Department of Information Engineering
Master Thesis in Control Systems Engineering

External axis position optimization for the trajectory generation of an industrial robot

Supervisor:
Prof.Ruggero Carli
University of Padua

Co-Supervisor:
Dr. Francesco Tomietto
Euclid Labs s.r.l

Candidate:
Riccardo Marcon
Student ID:
2076733

Academic Year 2024/2025

Dedicated to my parents,
for their unwavering support
to make my dreams come true.

Abstract

This thesis addresses the challenge of optimizing the position of an external linear axis in trajectory generation for industrial robots. Systems equipped with an additional external axis, such as a linear track, often face inefficiencies due to suboptimal positioning during task execution. The goal of this research is to develop a systematic methodology for quantifying the optimal configuration of the external axis and to implement this method within a trajectory optimization framework for six-axis manipulators. The research begins by establishing a mathematical model to evaluate the relationship between external axis position, system kinematics, and task requirements.

An optimization algorithm is then proposed to determine the ideal external axis configuration for a given task, ensuring that the manipulator operates in its most efficient configuration. Building on this, a second method is developed to optimize the entire trajectory, integrating external axis positioning within the motion planning process. This dual-layer optimization ensures that both the external axis and the internal joints contribute to a smooth and efficient trajectory.

The methods are validated through simulations and experiments in an industrial robotic system, demonstrating significant improvements in task execution efficiency. This research provides a practical tool for industries seeking to improve the performance of external axed robotic systems, paving the way for more agile and cost-effective automation solutions.

Sommario

Questa tesi affronta la sfida di ottimizzare la posizione di un asse lineare esterno nella generazione di traiettorie per robot industriali, I sistemi dotati di un asse esterno aggiuntivo, come un binario lineare, spesso presentano inefficienze dovute a un posizionamento subottimale durante l'esecuzione dei compiti. L'obiettivo di questa ricerca è sviluppare una metodologia sistematica per quantificare la configurazione ottimale dell'asse esterno e implementare questo metodo all'interno di un framework di ottimizzazione delle traiettorie per manipolatori a sei assi. La ricerca inizia con la creazione di un modello matematico per valutare la relazione tra la posizione dell'asse esterno, la cinematica del sistema e i requisiti del compito. Successivamente, viene proposto un algoritmo di ottimizzazione per determinare la configurazione ideale dell'asse esterno per un determinato compito, garantendo che il manipolatore operi nella configurazione più efficiente. Successivamente, viene sviluppato un secondo metodo per ottimizzare l'intera traiettoria, integrando il posizionamento dell'asse esterno nel processo di pianificazione del movimento. Questa ottimizzazione a doppio livello garantisce che sia l'asse esterno che le giunzioni interne contribuiscano a una traiettoria fluida ed efficiente.

I metodi sono stati validati attraverso simulazioni ed esperimenti su un sistema robotico industriale, dimostrando miglioramenti significativi nell'efficienza dell'esecuzione dei compiti. Questa ricerca fornisce uno strumento pratico per le industrie che cercano di migliorare le prestazioni dei sistemi robotici con assi esterni, aprendo la strada a soluzioni di automazione più agili ed economiche.

Contents

Chapter 1

Introduction

1.1 Context and application

In advanced industrial environments, especially those with large workspaces or complex tasks, it is common to integrate manipulators with an external linear axis, often referred to as a seventh degree of freedom. This additional motion capability allows the entire system to translate along a straight path, significantly expanding the operational range. Such configurations are widely adopted in sectors such as automotive assembly, aerospace, welding, painting, and logistics, where flexibility and extended reach are essential.

By treating the linear axis as part of the kinematic chain, motion planning can be greatly improved, particularly when precise trajectories need to be followed across distributed workstations. One of the main benefits of this setup is the ability to avoid singular configurations, situations where control becomes unstable or limited due to alignment issues within the mechanical structure. Moreover, the added mobility helps to overcome reachability constraints and allows more efficient posture optimization along complex paths.

This approach becomes particularly important when the workpieces are large enough that the system cannot perform all tasks while remaining fixed in one position. Although the introduction of industrial manipulators primarily increased productivity, coupling them with an external axis significantly impacts other aspects, such as reducing the complexity of overall industrial automation. Specifically, it eliminates the need for additional equipment to move or palletize parts before and after processing. Furthermore, it also reduces the risk of injury for operators, who no longer need to manually handle heavy or bulky items.

Furthermore, the extra degree of freedom allows for greater flexibility in performing multiple machining or assembly operations on a single workpiece without the need for separate stations. By enabling the manipulator to translate along a linear path, it becomes possible to efficiently access different areas of the same part, reducing setup times and improving

overall process integration.

Although collaborative devices (cobots) are gaining popularity due to their flexibility and ease of integration, traditional industrial arms continue to lead the market in terms of speed, precision, and efficiency. According to reports from industry experts, in high-volume production settings, these systems excel by maintaining throughput and accuracy, meeting the demands of large-scale manufacturing [1]. Similarly, analyses highlight that heavy duty manipulators outperform collaborative units in repetitive and high-speed applications [2].

Despite the advantages of cobots, such as safer human interaction and simple programming, their lower speed and payload capacity make them less suitable for certain tasks. Studies indicate that collaborative arms operate at reduced velocities to enable force-feedback safety features, which can limit their efficiency in demanding environments [3].

In summary, integrating an external linear axis into industrial automation systems not only enhances flexibility and workspace but also allows for differentiated operations on a single workpiece without the need for multiple stations. While collaborative devices bring safety and adaptability, traditional industrial manipulators remain the preferred solution for high-speed, high-precision production lines.

Figure 1.1: Kuka robot sympathetic to the sled

1.2 Thesis objectives

The main goal of this thesis is to improve the understanding and optimization of positioning strategies for industrial automation systems, especially those that involve manipulators with additional degrees of freedom provided by external linear axes. In modern production environments, where both the complexity and variability of tasks are increasing, the ability to analyze and control how a system behaves in specific spatial configurations is essential to ensure safety, repeatability, and efficiency. This work addresses these challenges through three interrelated research objectives.

The first objective is to discriminate and quantify the quality of a single pose taken by the manipulator with respect to a set of reference criteria. This analysis involves assessing how the pose relates to internal joint values (e.g., how far a joint is from its mechanical limit), the proximity to singularities, and the system's general dexterity or manipulability in that position. Various metrics will be considered and compared, including distance from nominal configurations, center of the feasible range or try to maintain the same distance from the flange. The purpose is to define robust methods to evaluate whether a given pose is advantageous, neutral, or critical, regardless of the task to which it belongs. These indicators are essential for identifying positions that may lead to instability or reduced motion freedom during operation.

Building on this, the second goal is to extend this evaluation to entire trajectories, particularly those defined through sequences of linear movements in Cartesian space. Many real-world applications involve straight-line interpolation between key points, common in welding, painting, or motion capture-driven tracking. In these cases, it is possible that several distinct spatial positions along the path result in the same manipulator configuration (i.e., the same joint arrangement). This thesis aims to detect and classify such occurrences, enabling an understanding of when and how configuration redundancy arises along a trajectory. Recognizing segments of motion that share the same configuration is important because it may signal inefficiencies, loss of redundancy usage, or limited adaptability in dynamic tasks. Conversely, identifying variation in configurations can offer more options for optimization and which is the best.

The third and most application-oriented objective is to develop optimization algorithms capable of computing feasible and advantageous configurations for each point of a path, using the evaluation methods developed in the first two objectives as cost functions. The aim is to design solutions that not only maintain kinematic feasibility but also actively seek optimal placement strategies along the external linear axis. These algorithms will incorporate multicriteria evaluation, balancing factors such as distance from singularities, joint usage uniformity, and smoothness of motion transitions. The strategies developed will be especially useful in scenarios where external axes are used to extend reach or

reorient the arm dynamically while maintaining task constraints. The inclusion of the external axis in the optimization loop enables intelligent trajectory planning, where all degrees of freedom, both internal and external, contribute to improving overall system behavior.

Together, these research objectives form a comprehensive methodology for improving path planning, pose evaluation, and motion control in industrial automation configurations. This is particularly important in systems that aim to mimic complex human-driven trajectories (e.g., motion capture) or operate in environments where workspace boundaries and singularities pose limitations. By addressing both the static (single pose) and dynamic (wide trajectory) dimensions of the problem, the thesis provides a solid foundation for more intelligent, robust, and adaptive robotic systems capable of performing complex tasks over large areas with high accuracy and minimal intervention. This is because one or more reference positions may be better depending on the type of work to be performed. For example, the longer a sled is, the greater the positioning error of the robot may be during a parametric program. Conversely, when one has particular tools to be used, then having the axes always working within a limited range can improve the end result or in some cases be a compulsory choice for reasons of space and cleanliness, for example during welding and painting work.

In addition, a future aspect to this elaboration may be conducting tests and subsequent application with an external axis that is not integral with the robot but nevertheless moves synchronously with it, such as welding cradles, particular lathes, etc.

1.3 Software Marvin

Marvin is a software solution developed by Euclid Labs to enable rapid and intuitive programming of industrial robots. Designed especially for industry professionals without extensive experience in robotic programming, Marvin simplifies the automation process by allowing users to manually record complex trajectories performed by an operator. These movements are then automatically converted into executable robot programs within minutes, significantly reducing the need for advanced programming expertise.

Marvin is hardware independent, supporting a wide range of trajectory acquisition devices and generating compatible code for most major industrial robot brands, including Yaskawa Motoman, KUKA, Fanuc, Epson, ABB, Stäubli, Hyundai, Denso, and others.

Operating offline, Marvin features a user-friendly interface that digitally replicates the robot's work cell. This allows users to verify, optimize, and simulate processes before implementation, minimizing downtime and enabling easy exploration of alternative paths, tool configurations, and process setups.

Figure 1.2: Marvin Logo

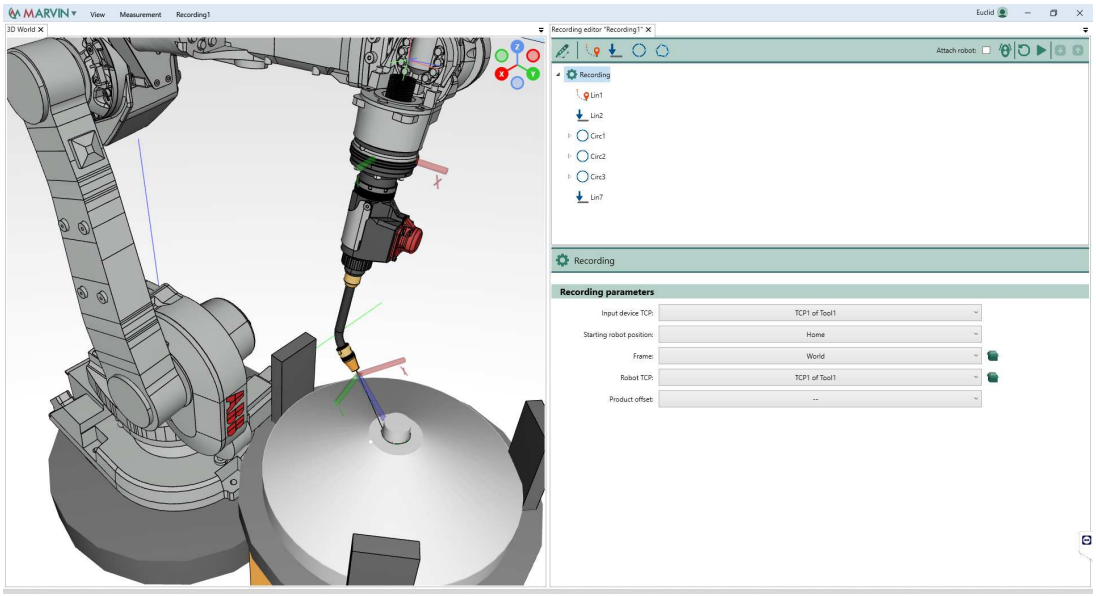


Figure 1.3: Marvin interface

To enhance programming flexibility, Marvin offers several automated features. It allows the integration of subprogram calls before or after user-defined events, supports comments and triggers for custom routines activated by entering specific volumes (such as automatic torch cleaning during welding), and enables dynamic speed adjustments for process and specific needs like sealing. Additionally, it provides post-processing of curved corners to ensure trajectories match the physical constraints of the end effector.
With these capabilities, Marvin empowers users to efficiently automate complex tasks while maintaining flexibility and control across a wide range of industrial applications.
This thesis aims to identify potential algorithms to be integrated into Marvin, the software developed by Euclid Labs for fast and intuitive industrial robot programming. The goal is to extend the software's capabilities to support the processing of large sized products, where the use of an external axis is required, though not necessarily a linear slide, as in the

project developed. The integration of these algorithms is designed to make Marvin even more flexible and effective, allowing the automation of complex manufacturing scenarios that require advanced coordination between robots and auxiliary axes.

1.4 Euclid Labs

This thesis was conducted in collaboration with EUCLID LABS, a leading company in high-tech solutions for robotics and industrial automation. Founded in 2005, it designs and develops flexible and effective software tools to address challenges such as small-batch production, high-quality requirements, and the automation of complex tasks. The company works closely with robot manufacturers, machine builders, and system integrators to create advanced applications for automatic programming, always aiming to increase the profitability of industrial automation.
At the core of its product portfolio is random bin picking software, which stands as the company's flagship solution, with over 100 installations worldwide. In addition, EUCLID LABS offers a wide range of offline programming tools that have been successfully applied to various automation tasks, including lens edging, panel machining, laser sorting, robot palletizing, press brake bending, riveting, clinching, marble polishing, Cartesian painting, automatic surface scanning, trajectory creation using haptic devices, and robot language translation. These products reflect the company's commitment to minimizing programming time while integrating adaptive capabilities into industrial systems.

Figure 1.4: Logo Euclid Labs

Chapter 2

Theoretical Background

2.1 Industrial robot

In industrial automation, the six-axis articulated robot plays a fundamental role by offering versatile, precise, and repeatable motion in three-dimensional space. Its mechanical structure consists of a serial chain of six revolute joints arranged to mimic the human arm and wrist. The base joint provides horizontal swivel, followed by two joints forming the shoulder and elbow, which control vertical elevation and reach. A three-axis wrist at the distal end enables pitch, yaw, and roll of the end-effector, allowing complex orientations and trajectories.

Each joint is driven by a high-torque brushless DC or servo motor coupled with a precision gearbox, ensuring adequate force to handle payloads while achieving fine positional resolution. Encoders integrated into each joint measure angles with resolutions on the order of ten-thousandths of a degree, feeding back into closed-loop control systems that enforce precise motion. The links connecting joints are typically made from high-strength aluminum or steel alloys, often featuring hollow sections to accommodate power and signal cables, pneumatic or hydraulic lines, and quick-change tool interfaces.

Modeling the kinematics of a six-axis robot involves solving two complementary problems. Forward kinematics uses homogeneous transformation matrices derived via the Denavit-Hartenberg convention to compute the position and orientation of the end effector from known joint angles.

An industrial robot is a mechanical system composed of a series of joints that allow it to move within three-dimensional space. The number of actuated joints determines the robot's degrees of freedom (DoF). A standard robotic manipulator typically possesses six degrees of freedom, which are sufficient to position its end-effector (commonly referred to as the TCP, or Tool Center Point) at any desired location and orientation in 3D space. This is because a pose in 3D space is defined by six parameters: three for position (x, y, z) and three for orientation (commonly expressed as roll, pitch, and yaw angles).

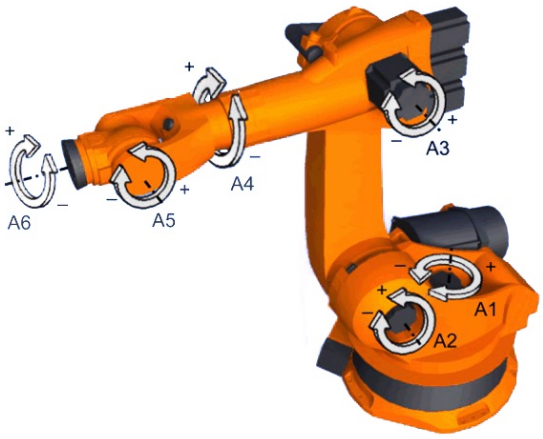


Figure 2.1: Joints rotation with respect to DH convention

When a robot has exactly six DoF and no mechanical constraints or obstacles are present, there are a finite number of joint configurations that can achieve a given TCP pose. In typical anthropomorphic robots, up to eight such inverse kinematic solutions may be found. This is due to the kinematic structure of the robot: There are generally binary choices in the shoulder (left / right), elbow (up / down), and wrist (flipped / not flipped). By combining these possibilities ($2 \times 2 \times 2$), one obtains up to eight distinct configurations that result in the same coordinates, although each represents a physically different arm posture. However, not all of these solutions are always feasible, as some may violate joint limits or lead to collisions.

2.1.1 Transformation matrix

The parameters of each link (link length, link twist, link offset, and joint rotation) generate a 4×4 transform matrix; the overall pose follows from the product of the six link transformations. Inverse kinematics reverses this process, determining the joint angles required to achieve the desired pose of the end effector. Analytical solutions typically yield several valid configurations, including elbow-up and elbow-down variants, and a path-planning algorithm selects the solution that optimizes motion smoothness, avoids singularities, and respects joint constraints.

The situation changes fundamentally when an external (seventh) axis is added to the system, such as a linear track on which the robot base is mounted or an additional rotary axis. In this case, the number of joint variables exceeds the six constraints imposed by the desired pose of the TCP. The system becomes kinematically redundant. From a mathematical perspective, this means that there are more unknowns than equations, leading to a solution space that is not discrete but continuous. In other words, there

exists a continuous set (a one-dimensional curve) of joint configurations that all result in the exact same position.

This property can be formally demonstrated by considering the forward kinematics function, which maps joint configurations from a 7-dimensional space (joint variables) to a 6-dimensional space. By fixing the position, one imposes six constraints (three coordinates and three rotation), and therefore the set of joint configurations that satisfy these constraints forms a manifold of dimension one in the configuration space. This implies that an infinite number of inverse kinematic solutions exist, each corresponding to a different internal configuration of the robot and its external axis, yet all yielding the same end-effector pose.

The general form of a homogeneous transformation matrix is the following.

$$T = \begin{bmatrix} R_{3\times3} & \mathbf{p}_{3\times1} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

where R represents the rotation of the frame and $\mathbf{p}$ represents the position of the origin of the frame.

A common method to describe robot kinematics is through the Denavit–Hartenberg (DH) convention. Using four parameters $(\theta_i, d_i, a_i, \alpha_i)$ for each joint, the transformation from frame $i-1$ to frame i is expressed as follows:

$$A_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

Here:

- θ_i is the joint angle (rotation about z_{i-1}),
- d_i is the joint offset (translation along z_{i-1}),
- a_i is the link length (translation along x_i),
- α_i is the link twist (rotation about x_i).

To compute the complete forward kinematics of a serial robot with n joints, the individual transformation matrices are multiplied in sequence:

$$T_n^0 = A_1 A_2 \cdots A_n \tag{2.3}$$

This final matrix T_n^0 gives the pose of the end-effector frame with respect to the base frame.

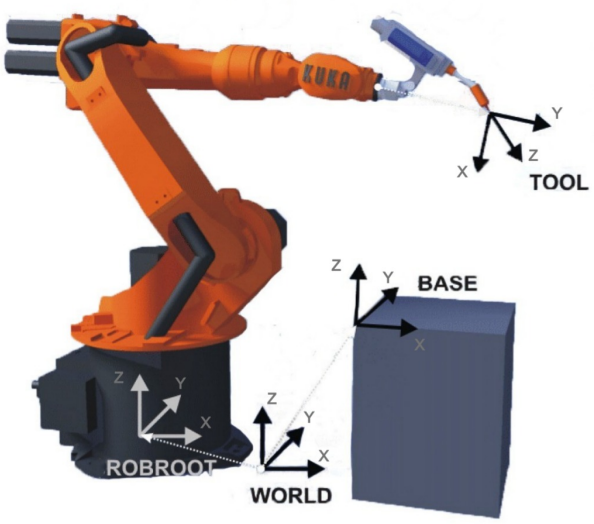


Figure 2.2: Different coordinates with respect to the world, robot, TCP, and base object

The reachable workspace of a six-axis robot forms a complex volume, often resembling twisted toroidal shells bounded by the geometric limits of each joint and link. Within this workspace, manipulability measures quantify the ability to generate velocities or forces in arbitrary directions, highlighting regions of high dexterity and flagging proximity to kinematic singularities where control becomes ill-conditioned. Equally important in production environments is repeatability, or repeatability: the robot's capacity to return its end-effector to a programmed pose under identical conditions. Repeatability is commonly specified as a two-sigma range (e.g., ± 0.02mm), representing the statistical spread of repeated measurements and ensuring consistent quality in tasks such as welding, pick-and-place or precision assembly.

In practice, six-axis robots excel in applications requiring complex spatial maneuvers combined with tight repeatability. In automotive assembly, they perform spot welding along intricate seams; in painting and coating operations, they maintain precise standoff distances and orientations over curved surfaces; in material handling, they execute high-speed pick-and-place cycles and palletizing; and in electronics manufacturing, they place miniature components onto circuit boards with sub-millimeter accuracy. By integrating vision systems and force/torque sensors, these robots adapt to variations in part position, execute compliant assembly, and conduct non-contact inspection, further broadening their applicability.

As manufacturing trends shift toward greater flexibility and customization, the six-axis robot remains a keystone of automation. Its human-like kinematic chain, coupled with high-precision actuation and feedback, ensures both versatility and consistency. Advances

in control algorithms, sensor fusion, and machine learning promise to enhance path planning and adaptive behaviors, strengthening the role of the six-axis robot in the factory of the future as a reliable, repeatable, and highly capable automation platform.

2.1.2 Status and Turn parameters in KUKA Robot

In robot manipulators with six degrees of freedom, specifying the Cartesian pose of the end effector - position (X, Y, Z) plus orientation (A, B, C) - does not uniquely determine the six joint angles. Due to the kinematic redundancy of the arm, a given TCP pose can correspond to multiple joint configurations (for example, elbow-up vs. elbow-down, wrist-flipped vs. non-flipped). KUKA resolves this ambiguity by augmenting each programmed pose with two integer parameters: **Status** (S) and **Turn** (T). When used together, (X, Y, Z, A, B, C, S, T) define a single unambiguous set of joint angles for point-to-point (PTP) motions, ensuring consistent and repeatable robot behavior, while, when the robots accomplish a linear trajectory (LIN), it does not take into account the specific configuration but only the coordinates.

The **Status** parameter is a 5-bit integer that encodes the kinematic "branch" that the robot should take, namely the configuration of the elbow and wrist axes. Bits 0–2 select among the eight possible combinations of elbow-up/down and wrist-basic/overhead positions; bit 4 is an informational flag (e.g., calibration state), and bit 3 is unused (normally the range is 0-7). In practice, bits 0–2 determine:

- **Elbow position** (axis A3): whether the third joint is flexed "up" ($A3 < 0°$) or "down" ($A3 \geq 0°$).
- **Wrist area**: whether the intersection of the A4-A6 axes lies in front of ('basic') or behind ('overhead') the shoulder plane.
- **Wrist tilt** (axis A5): whether the wrist is tilted toward or away from the base.

Bit	Function	0 (Bit = 0)	1 (Bit = 1)
0	Wrist area	Basic area (intersection of A4–A6 in front of shoulder)	Overhead area (behind the shoulder)
1	Elbow position	$A3 < 0°$ (elbow-up)	$A3 \geq 0°$ (elbow-down)
2	Wrist tilt	A5 tilted downward (reference A4 = 0°)	A5 tilted upward
3	—	Unused (always 0)	Unused (always 0)
4	Calibration flag	Non-absolute robot (taught point)	Absolutely accurate robot (informational only)

Table 2.1: Bit-level encoding of the Status parameter.

The **Turn** parameter is a 6-bit integer that fixes the sign (direction) of rotation for each of the six axes (A1 through A6). Each bit corresponds to one joint: a bit value of 0 means that the axis rotates via the positive shortest-path angle, while 1 means via the negative shortest-path angle. By specifying T, the programmer can command, for example, a rotation of 270 ° of axis A2 to be executed as +270° or –90°, whichever matches the intended motion. This eliminates the need for intermediate waypoints when rotations exceed ±180°, and prevents unintended wrap-around during execution.

Bit	**Axis**	**0 (Bit = 0)**	**1 (Bit = 1)**
0	A1	$A1 \geq 0°$	$A1 < 0°$
1	A2	$A2 \geq 0°$	$A2 < 0°$
2	A3	$A3 \geq 0°$	$A3 < 0°$
3	A4	$A4 \geq 0°$	$A4 < 0°$
4	A5	$A5 \geq 0°$	$A5 < 0°$
5	A6	$A6 \geq 0°$	$A6 < 0°$

Table 2.2: Bit-level encoding of the Turn parameter.

Together, **Status** and **Turn** guarantee that each programmed pose maps to a unique set of joint angles, eliminating inverse-kinematic ambiguity. This is crucial in high-precision applications, such as welding, assembly, or machining, where even slight deviations in elbow or wrist orientation can lead to part misalignment, collisions, or tool-center misplacement. Moreover, by inheriting S and T across sequential PTP commands, programs remain concise: common configurations need not be restated, yet any change in elbow or wrist posture can be introduced precisely by altering only the Status or Turn value at the relevant point.

Despite their power, S and T do not resolve every challenge: singularities, configurations in which two joint axes become parallel, still require careful avoidance or specialized handling (e.g., auxiliary singularity variables or path-blending commands). However, understanding and correctly using Status and Turn is foundational for robust KUKA programming. By embedding these bit-encoded parameters into every PTP instruction, the engineer obtains full control over the robot's kinematic branch and axis-rotation directions, yielding predictable, repeatable motion essential to advanced automation tasks.

2.2 External axis

To further enhance versatility, a linear external axis can be integrated beneath the robot's base, effectively converting the stationary arm into a mobile workstation. In this setup, the entire six-axis manipulator is mounted on a precision-guided rail, adding a straight-line translation to its native joint motions. This supplemental axis, often referred to as

the 'seventh axis', it allows the robot to move laterally throughout an extended work area, reaching points that lie well beyond the normal envelope of the fixed arm.

The external axis drive commonly employs a linear motor or ball-screw mechanism, chosen for its rigidity, rapid response, and sub-millimeter positioning accuracy. A linear encoder or scale provides closed-loop feedback, ensuring that the repeatability of the carriage is closely aligned with that of the robot joints (typically within ± 0.05mm at two-sigma). In coordinated motion planning, the rail's translation is treated as an additional variable in the forward and inverse kinematics equations: desired tool paths are achieved by solving for six joint rotations plus the linear offset that collectively place the end-effector on the intended trajectory.

By combining the linear rail with the articulated arm, the system gains seven independent axes of movement, greatly expanding the reachable volume without sacrificing the robot's inherent dexterity or repeatability. This extended reach is particularly advantageous for processes spanning long conveyors, large fixtures, or multiple workstations. The seventh axis also improves obstacle negotiation, enabling the arm to reposition its base before executing complex wrist gestures and reduces the need for manual re-fixturing or multiple robot cells. Overall, this hybrid configuration delivers a seamless integration of translational and rotational freedom, empowering high-precision automation across broader and more varied production layouts.

Because the combination of six articulated joints (or seven when including an external linear axis) introduces kinematic redundancy, a single target pose, defined by three Cartesian coordinates and three orientation angles, can be reached by infinitely many joint configurations. In mathematical terms, the problem of non-linear inverse kinematics of the manipulator exhibits a continuous family of solutions parameterized by one or more free variables (often called redundancy parameters). For a six-axis arm mounted on a seventh linear axis, one degree of redundancy remains even after specifying the full pose of the end effector, allowing the system to optimize secondary criteria such as obstacle avoidance, joint limit margins, or energy minimization.

In practice, the robot controller exploits this redundancy by implementing null-space projections: after computing one primary solution that achieves the desired tool position and orientation, additional joint motions lying in the null space of the Jacobian matrix can adjust the configuration without disturbing the end-effector pose. This capability enhances flexibility in crowded workcells, enabling dynamic reconfiguration of the arm's shape to clear obstacles or improve stiffness while maintaining precise task execution. Ultimately, the infinite solution set afforded by kinematic redundancy becomes a powerful tool for advanced path planning and real-time adaptation in complex industrial environments.

Mathematically the situation changes fundamentally when an external (seventh) axis is added to the system, such as a linear track on which the robot base is mounted or

an additional rotary axis. In this case, the number of joint variables exceeds the six constraints imposed by the desired pose of the TCP. The system becomes kinematically redundant. From a mathematical perspective, this means that there are more unknowns than equations, leading to a solution space that is not discrete but continuous. In other words, there exists a continuous set (a one-dimensional curve) of joint configurations that all result in the exact same position.
This property can be formally demonstrated by considering the forward kinematics function, which maps joint configurations from a 7 joint variables to a 6-dimensional space. By fixing the position, one imposes six constraints, and therefore the set of joint configurations that satisfy these constraints forms a manifold of dimension one in the configuration space. This implies that an infinite number of inverse kinematic solutions exist, each corresponding to a different internal configuration of the robot and its external axis, yet all yielding the same end-effector pose.
It is this one the focus of the thesis, between the infinity configuration possibility, finding the best with respect to different initial conditions and methods.

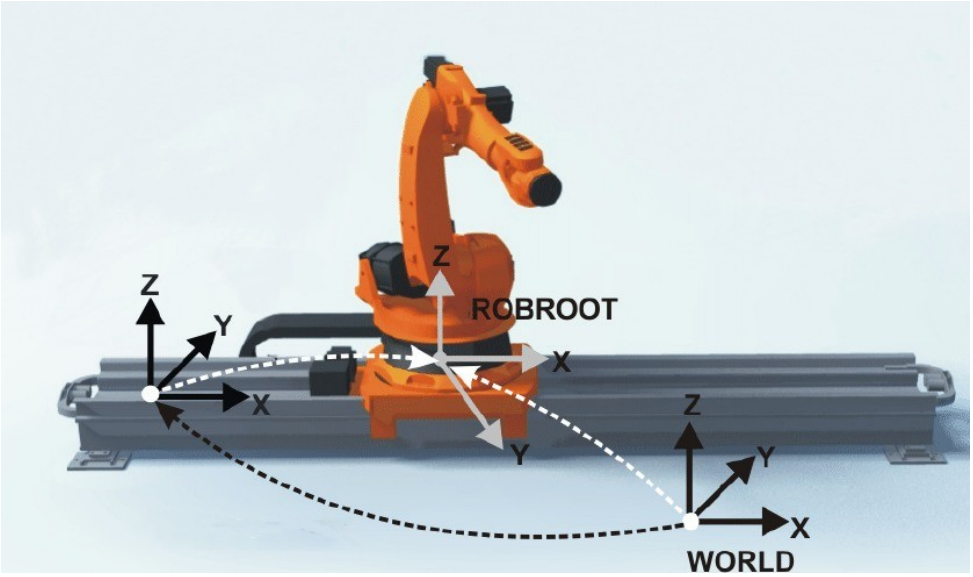


Figure 2.3: Reference coordinates of a robot integral to an external axis

2.3 Convex optimization

Convex optimization is a subfield of mathematical optimization concerned with minimizing (or maximizing) convex functions on convex sets. Its remarkable theoretical properties, including the global optimality of any local optimum, well-behaved duality, and powerful convergence guarantees, make it the foundation for many advanced numerical methods, including interior-point algorithms.

A *convex set* $\mathcal{C} \subseteq \mathbb{R}^n$ satisfies the property that for any two points $x, y \in \mathcal{C}$, the line segment that connects them also lies entirely within $\mathcal{C}$:

$$\forall \theta \in [0,1], \quad \theta x + (1-\theta)y \in \mathcal{C}.$$

Common examples include affine subspaces, Euclidean balls, and polyhedra defined by linear inequalities. A *convex function* $f : \mathcal{C} \to \mathbb{R}$ over a convex set satisfies

$$\forall x, y \in \mathcal{C},\ \forall \theta \in [0,1], \quad f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y).$$

This inequality ensures that the epigraph

$$\text{epi}(f) = \{(x,t) \mid x \in \mathcal{C},\ f(x) \leq t\}$$

is a convex set.

A *convex optimization problem* takes the standard form:

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \leq 0, \quad i = 1, \ldots, m, \\ & Ax = b, \end{aligned}$$

where the objective f_0 and inequality constraints f_i are convex functions, and $Ax = b$ describes affine equality constraints. Under convexity, any local minimum is a global minimum, and feasible regions inherit convexity from the intersection of convex sets.

Equality constraints are conditions in which a specific function of the decision variables must be exactly equal to a given value. These constraints are usually written as $Ax = b$, where it is a function involving one or more variables of the problem. In engineering applications, equality constraints are used when a particular relationship between variables must be strictly maintained. For example, in kinematic mechanisms, the length of a closed-loop linkage must be constant regardless of the joint positions; this can be modeled with an equality constraint.

Inequality constraints define a range or limit within which a function must lie. They are typically expressed as: $g(x) \leqslant 0$ or $g(x) \geqslant 0$. These are used in engineering to ensure that the design variables remain within safe or functional boundaries. For example, the stress in a material must not exceed its yield strength, or the power output of a generator must remain within a specific range.

In robotics, an inequality constraint might restrict the angle of the joint to avoid mechanical collisions. In fluid dynamics, it could ensure that pressure stays above a critical level to prevent cavitation. These limitations reflect realistic operational or safety limits.

Unlike equality constraints, inequality constraints provide flexibility, solutions can lie

anywhere within the allowed region. However, they still play a critical role in shaping the feasible space of the optimization problem.
In numerical optimization algorithms, inequality constraints are typically handled using methods such as penalty functions, barrier methods, or slack variables.
Another example is in electrical circuits, where the sum of voltages in a closed loop (by Kirchhoff's voltage law) must be zero. Equality constraints are also used to conserve mass, energy, or momentum in physical systems.
Mathematically, equality constraints are more rigid than inequality constraints because they do not allow any deviation. In optimization, handling them is often more challenging, as even slight violations make the solution infeasible.
Duality theory provides insight into problem structure via the Lagrangian

$$\mathcal{L}(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \nu^T (Ax - b),$$

where $\lambda \geq 0$ and ν are Lagrange multipliers. The dual function

$$g(\lambda, \nu) = \inf_x \mathcal{L}(x, \lambda, \nu) \tag{2.4}$$

yields a concave maximization problem. *Strong duality* holds under Slater's condition existence of a strictly feasible point, so that the optimal values of primal and dual coincide.
In the field of optimization, particularly in engineering applications involving nonlinear problems with constraints, the **Karush-Kuhn-Tucker (KKT) conditions** play a central role. These conditions provide a mathematical framework for identifying optimal solutions to problems with both *equality* and *inequality constraints*. The KKT conditions extend the classical method of Lagrange multipliers by incorporating inequality constraints and are considered necessary conditions for optimality under certain regularity assumptions.
Consider a general constrained optimization problem:

$$\begin{aligned} \text{Minimize} \quad & f(x) \\ \text{subject to} \quad & h_i(x) = 0, \quad i = 1, \ldots, m \\ & g_j(x) \leq 0, \quad j = 1, \ldots, p \end{aligned}$$

where x is the vector of decision variables, $h_i(x)$ are the equality constraints, and $g_j(x)$ are the inequality constraints.
The KKT conditions state that, at the optimal point x^*, there must exist Lagrange multipliers λ_i (for the equality constraints) and μ_j (for the inequality constraints, with $\mu_j \geq 0$) such that the following conditions are satisfied:

1. **Stationarity:**

$$\nabla f(x^*) + \sum_{i=1}^{m} \lambda_i \nabla h_i(x^*) + \sum_{j=1}^{p} \mu_j \nabla g_j(x^*) = 0$$

2. **Primal feasibility:**

$$h_i(x^*) = 0, \quad g_j(x^*) \leq 0$$

3. **Dual feasibility:**

$$\mu_j \geq 0$$

4. **Complementary slackness:**

$$\mu_j \cdot g_j(x^*) = 0$$

These conditions describe a balance between the objective function's gradient and the gradients of the active constraints. In engineering design, such conditions are used to ensure that systems operate within safety or physical limits while minimizing cost, weight, or energy.

It is important to note that the KKT conditions are *necessary* for a local optimum if certain regularity conditions, known as constraint qualifications (e.g., the Mangasarian-Fromovitz condition), are satisfied. In convex optimization problems, the KKT conditions are also *sufficient* for global optimality.

Among solution methods, *interior point methods* approach the boundary of the feasible region from the interior, using barrier functions (e.g., the logarithmic barrier) to enforce inequalities. At each iteration, a parameterized subproblem with a smooth objective is solved, often using Newton's method, gradually reducing the barrier weight to drive the iterates toward the optimum. The combination of polynomial-time complexity and practical efficiency makes interior-point algorithms a go-to choice for large-scale convex programs.

This primer outlines the geometric and algebraic underpinnings of convex optimization, preparing for a detailed exposition of interior-point methods, their convergence analysis, and implementation nuances.

2.3.1 Simplex Method

The Simplex Method is a widely used algorithm for solving linear programming problems, where the objective is to optimize a linear function subject to a set of linear constraints. Developed by George Dantzig in 1947, the method remains fundamental in operations research and engineering optimization.

In its standard form, a linear programming (LP) problem can be written as:

$$\text{Maximize} \quad Z = c^T x$$

$$\text{Subject to} \quad Ax \leq b, \quad x \geq 0$$

where x is the vector of decision variables, c is the coefficient vector in the objective function, A is the matrix of constraint coefficients, and b is the vector on the right-hand side.

The feasible region defined by the constraints of an LP problem forms a convex polyhedron. The Simplex Method exploits a crucial property of linear programs: if an optimal solution exists, it will be located at a vertex (corner point) of the feasible region. Instead of evaluating all possible solutions, the method moves from one vertex to another, always improving the objective function, until it reaches the optimum.

The procedure begins by converting all inequality constraints into equalities using *slack variables*, thereby transforming the system into a form suitable for matrix representation. The resulting system is organized into a **Simplex tableau**, which contains the coefficients of the constraints and the objective function.

At each iteration, the algorithm selects one *entry variable* (typically the one that will improve the objective the most) and one *exit variable* (the current basic variable that will be replaced). A process known as **pivoting** updates the tableau to reflect the new basis. This process repeats until no further improvement is possible, indicating that the optimal solution has been found.

The Simplex Method is particularly valuable in engineering for solving optimization problems related to resource allocation, production planning, transportation, network flows, and many others.

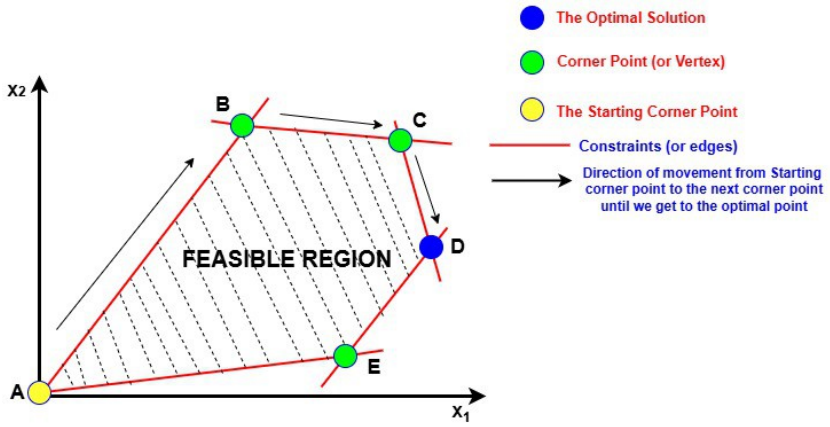


Figure 2.4: Graphic example of simplex method

Despite its exponential worst-case complexity, it performs efficiently on most real-world problems and has been the basis for many modern optimization software tools.

This method provides a powerful and systematic approach to solving linear optimization problems. Its ability to handle multiple variables and constraints with reliability and speed has made it a cornerstone of mathematical programming in both theoretical and applied contexts.

An other very powerful method is the *Interior Point Method*, which is more complicated than the previous, but in most cases is faster than the previous one, to try the optimal position for the external axes of the robot. I choose this one and it will be explained in the next chapter, 4.1.

Chapter 3

Problem Formulation

3.1 Cost function

In any optimization process, the cost function plays a central and critical role. Also known as the objective function, loss function, or fitness function (depending on the context), it serves as the mathematical expression of the goal we aim to achieve through optimization. The primary function of the cost function is to evaluate the quality of a solution. It assigns a numerical value to each possible solution, indicating how "good" or "bad" that solution is with respect to the desired outcome. Optimization algorithms use this information to guide their search, iteratively improving the solution by moving toward lower (or higher, in case of maximization) cost values.

In addition, the cost function defines the objective of optimization. Whether the goal is to minimize energy consumption, reduce time, increase efficiency, or find the shortest path, the cost function captures this goal in precise mathematical terms. A well-designed cost function ensures that the optimization results are not only mathematically optimal but also practically useful and aligned with real-world requirements.

In many scenarios, optimization problems involve multiple conflicting goals, such as minimizing cost while maximizing performance. In such cases, the cost function can be designed to balance these objectives using weighted combinations, allowing the algorithm to find a satisfactory trade-off.

Another crucial aspect is that the cost function directly affects the efficiency and stability of the optimization process. A poorly chosen cost function can mislead the algorithm into suboptimal solutions, slow down convergence, or even cause instability.

Ultimately, the cost function determines the success or failure of an optimization task, regardless of how advanced or sophisticated the algorithm is, if the cost function does not accurately represent the desired outcome, the optimization will not yield useful results.

In any optimization strategy, the cost function plays an essential role. Also known as the objective or loss function, it serves as the internal compass of the algorithm, evaluating

potential solutions and guiding the search for the best solution.
Two different cost functions were created because the positioning of the robot's outer axis is also affected by the relative value of all its inner axes, and this fact could somehow have favorable or unfavorable repercussions in finding the absolute optimal position. In this way we are going to note which aspect and therefore which axis has more relevance through the two algorithms examined, highlighting the different behavior given by the fact that one is a linear joint while the others are all rotoidal. Certainly I want to note a difference in computational speed in finding the required optimum, which should be about 6 to 1. Finally through the various graphs note whether one method or the other creates better motions throughout the trajectory, or produces instabilities or spikes in the variation of a certain axis.

3.1.1 External axis cost function

It assigns a numerical value representing the "cost" or penalty of a solution, allowing the algorithm to compare options and make informed decisions. In our case, it penalizes sled positions (E_1, E_2) that deviate from their ideal targets (E_1^{opt}, E_2^{opt}).
But this is not just a technicality; it is the heart of the entire optimization process. A well-designed cost function ensures:

- The optimization stays goal-oriented, not just mathematically valid.
- Small deviations are gently discouraged, while large ones are strongly penalized.

In multiobjective optimization, cost functions become even more critical. They enable the integration of several objectives into a single, tunable expression. This flexibility allows us to encode real-world priorities directly into the mathematical model.
Lastly, the cost function determines the convergence behavior of the algorithm. If poorly designed, it can mislead the optimizer, slow down performance, or produce irrelevant results. Thus, a properly formulated cost function is not just helpful, it is essential for success.
At the heart of our approach is this simple but powerful idea: the further your sled position strays from perfection, the harder we punish it mathematically. We do this with our cost function:

$$J(E) = w_E \left[(E_1^{\text{opt}} - E_1)^2 + (E_2^{\text{opt}} - E_2)^2 \right] \tag{3.1}$$

The track positions at each point - we call them E_1 (where you came from) and E_2 (where you are going) are sneaky little troublemakers. If they're even slightly off from their ideal positions (E_1^{opt} and E_2^{opt}), your robot's actual path drifts off course.

Cool trick: When we add other constraints later, we can tweak w_E to balance their importance. Think of it like adjusting ingredient ratios in a recipe, more salt here, less sugar there.

3.1.2 Internal joints cost function

To establish a comprehensive comparative framework for evaluating the two algorithms, a dual cost function approach was implemented. This methodology was designed to assess algorithm performance from two fundamentally different perspectives, thereby providing a more robust evaluation of their respective capabilities.

The first cost function focused on optimizing the performance of the outer axis by comparing its operational values with predetermined optimal reference parameters. This approach allowed for the assessment of precision and accuracy in single-axis control scenarios.

In contrast, the second cost function adopted a fundamentally different evaluation strategy. Rather than concentrating on individual axis optimization, this function was designed to quantify the collective deviation of all six robotic joints from a reference configuration. Specifically, the algorithm calculated the cumulative discrepancy between the current joint positions and the initial calculated position within the trajectory sequence, which served as the reference baseline.

This dual approach enabled a comprehensive evaluation that examined both localized performance (single-axis optimization) and system-wide coordination (multi-joint deviation analysis). The contrasting nature of these cost functions provided complementary insights into algorithm behavior, allowing for a more nuanced understanding of their respective strengths and limitations in different operational contexts.

The selection of the first calculated trajectory position as the reference point for the second cost function was strategically chosen to establish a consistent baseline that reflects the system's initial optimal state, thereby providing a meaningful benchmark for measuring subsequent deviations throughout the trajectory execution.

Then, we can translate these words into mathematical expression:

$$J(\mathbf{j}) = \sum_{k=1}^{n} \sum_{i=1}^{6} (j_{i,k} - j_{ref,i})^2 \tag{3.2}$$

where $\mathbf{j_{i,k}}$ is clearly the value of the joint i for the position k along the trajectory that has n different points. While, $\mathbf{j_{ref,i}}$ is the reference value chosen at the start of the algorithm.

3.2 Constraints

Robotic trajectory optimization requires careful balancing of performance objectives with critical physical and operational limitations. This document details three fundamental constraint categories essential for generating safe and executable motion paths in industrial robotic systems. These constraints address distinct but interconnected challenges, hardware protection, kinematic stability and physical feasibility.

- DC motors driving external axes are vulnerable to mechanical stress during rapid direction changes, particularly in orthogonal trajectory segments. Without explicit safeguards, micro-oscillations accelerate bearing wear and risk motor seizure.
- Linear trajectories demand consistent joint configurations to avoid singularities—instances where minor positional changes cause instantaneous loss of mobility. Mid-trajectory configuration shifts can trigger catastrophic operational halts.
- Mechanical range limits form non-negotiable boundaries for all movements. Exceeding these boundaries induces collisions, gear damage, or drive-train failures.

Their implementation involves both hard constraints (absolute requirements) and soft penalties (optimization nudges), creating a solution space that prioritizes safety over minimal-path efficiency. The following sections provide rigorous formulations of each constraint, their physical rationales, computational implications, and failure modes when violated.

1. Motor Protection Constraints

- **Mechanical Stress Prevention:** Rapid direction changes in DC motors cause bearing stress and accelerated wear, especially during orthogonal trajectory segments where lateral forces peak.
- **Constraint Logic:** For consecutive positions L_i and L_{i+1}, the displacement $|\Delta L| = |L_{i+1} - L_i|$ must exceed safety threshold δ_{tol} (e.g., $0.5°$). Violations trigger quadratic penalty $P(\Delta L) = k \cdot (\delta_{\text{tol}} - |\Delta L|)^2$.
- **Selective Activation:** Penalties apply *only* when $|\Delta L| < \delta_{\text{tol}}$, preserving cost function integrity during safe operations.
- **Physical Rationale:** Orthogonal motions induce maximum shear forces on motor shafts; micro-oscillations ($|\Delta L| \to 0$) cause cumulative bearing damage.
- **Failure Risk:** Unconstrained optimization allows "motor jitter" (high-frequency direction reversals), causing a seizure within operational hours.

2. Configuration Invariance Constraints

Since the study and tests are performed on a trajectory that has only linear movements then the status values must be those for the entire trajectory so that there are no configuration changes that would encounter singularities during the movement and this would create a big problem as the robot is unable to continue.

- **Singularity Prevention:** Joint alignment changes (e.g., elbow up to elbow down) cause instantaneous loss of mobility ("locking") during linear paths.
- **Constraint Logic:** Status variables (S, T) must remain constant:

$$S_1 = S_2 = \cdots = S_m, \quad T_1 = T_2 = \cdots = T_m$$

  where S represents joint configuration states and T represents tool orientation modes.

- **Physical Rationale:** Linear trajectories assume constant kinematic configuration; discontinuities induce infinite joint velocities.
- **Non-Negotiable Enforcement:** Solutions violating this are *infeasible* - the robot halts abruptly if $S_i \neq S_{i+1}$ occurs mid-motion.
- **Computational Load:** Requires $O(m)$ pairwise checks across all m trajectory points, dominating solve time for high-resolution paths.

3. Physical Operating Limits

- **Mechanical Boundary Enforcement:** Hard bounds prevent collisions, motor burnout, and gear damage:

$$E_{\min} \leq E_i \leq E_{\max} \quad \forall\, i = 1, \ldots, m$$

  where E_i is the position of the external axis (for example, rotation angle or linear slide position).

- **Physical Rationale:** Exceeding $[E_{\min}, E_{\max}]$ causes mechanical interference (e.g. arm collisions with the base) or overload of the drivetrain.
- **Dynamic Interaction:** Configuration invariance (S, T stability) can force E_i toward boundaries, shrinking feasible regions.

- **Zero-Tolerance Implementation:** Solutions breaching limits are immediately discarded during optimization.
- **Failure Mode:** Operating beyond the $E_{\max}$ jam mechanisms; positions below $E_{\min}$ induce gear backlash errors.

Table 3.1: Constraint Characteristics and Interactions

Constraint	**Type**	**System Interaction**
Motor Protection	Soft penalty	Conflicts with physical limits when δ_{tol} forces boundary approach; inactive during safe operations
Configuration Invariance	Hard equality	Dominates solution space; overrides motor protection if direction change alters S/T
Physical Limits	Hard boundary	Interacts with configuration invariance to shrink feasible regions; non-negotiable

3.3 Single position optimization methods

The first part of the thesis concerns optimization methods for calculating the position of the external axis (E1) for a 6-axis industrial robot mounted on a slide that serves as the seventh axis, given a reachable point within the robot's workspace. The purpose of these methods is to find the best possible position to avoid singularities and ensure that the robot can traverse a given trajectory without encountering singularity problems or unreachability issues due to one or more axes starting near their limits.
Basically, the goal of this thesis is to figure out which method is better in absolute terms, if any, or to identify, depending on the situation, which one is more advantageous to use.

3.3.1 Euclidean distance between point and flange

The first method consists of finding a fixed optimal distance, determined based on the robot's dimensions, where the algorithm must position the external axis at a value such that the Euclidean distance between the robot's base and its flange equals the desired value, always considering the three spatial dimensions x, y, and z. If this is not possible because the distance is greater than achievable, the algorithm proceeds by positioning the external axis as close as possible to the point, with respect to the point's x-coordinate. In case of further reachability problems, the algorithm iteratively moves away from the previously calculated value until a new reachable point is found.
This method was specifically designed to maintain the robot flange at a known or approximately constant distance from the robot base. This approach improves positioning accuracy, particularly during movements orthogonal to the external axis, since the linear unit typically remains stationary or moves very little. By limiting the movement of the external axis, positioning errors can be minimized, especially in applications involving parametric programs and long linear rails (for example, 10-20 meters) where mechanical imperfections, such as slight deviations from perfect linearity, may exist along the rail.
Furthermore, this strategy becomes even more advantageous when the robot has undergone advanced calibration procedures. Such calibrations improve the system's overall positioning repeatability, making it even more beneficial to keep the robot operating within a well-characterized and optimized working envelope.
This method is specifically designed to position the robot as far as possible from kinematic singularities, with the objective of ensuring a smooth and continuous trajectory along the entire path. Avoiding such critical configurations improves motion stability and can result in a higher average velocity over the trajectory, as the robot operates more efficiently in regions of greater manipulability.
A key aspect of the implementation lies in the customization of the Gaussian function parameters, particularly the location of the peak (i.e., the mean value), which does not

necessarily correspond to the geometric center of the joint's range. Instead, this value is defined according to the specific task that the robot is required to perform. For example, in welding operations or tasks performed within high-temperature environments—such as those near industrial furnaces—the robot's configuration is adjusted to maintain a posture that maximizes distance from heat sources, thereby preserving both the integrity of the robot and the quality of the process. On the other hand, when handling large payloads, the robot can be configured to maintain a more compact or 'crouched' position. This reduces the overall leverage of the manipulator, helping to limit vibrations and avoid resonant behavior in the parts being transported. Such postures also minimize mechanical stress, particularly in the wrist joints, which are generally more delicate than the primary axes.

This task-specific flexibility in configuration scoring, enabled by the Gaussian-based evaluation method, allows the system to autonomously favor robot postures that are not only kinematically stable but also optimized for the physical and environmental demands of the application.

3.3.2 Center of the robot joints

The second method seeks to obtain the best possible robot position by modifying the external axis value based on an optimal configuration of the 6 axes. The fundamental principle is based on evaluating the quality of each robotic configuration through the use of Gaussian functions applied to each axis, with the primary objective of avoiding kinematic singularities. For each reachable robot position where the external axis falls within its range, a coefficient determined by its configuration is calculated. This approach involves using the value of each axis in a given configuration as input to a Gaussian function, which has the axis range as extremes and a mean value specifically chosen based on the characteristics of the considered axis. The algorithm is designed to be configurable according to specific requirements: generally, the mean value coincides with the center of the axis range; however, for some axes this is not optimal. For instance, for the robot's fifth axis, the optimal mean value does not correspond to the center of the range since this position would lead the robot into a kinematic singularity configuration. The Gaussian function output yields a value between 0 and 1, based on how close the current position is to the predefined optimal value. Once this value is calculated for each of the six axes, all 6 results are summed to obtain an indication of how the robot configuration would be positioned. Clearly, the maximum value, indicative of a better configuration, is 6, while the minimum is 0. In the base approach of method 2, this procedure is repeated for every possible configuration along the entire external axis range, modifying the E1 value by a predetermined amount, typically 10-20mm, and selecting the configuration with

the highest score. Subsequently, the optimal external axis value is updated whenever the newly calculated coefficient exceeds the previous one. Once the entire traversable external axis has been examined, the E1 value saved for the best configuration is utilized. The fourth method represents an evolution of the second, maintaining the same Gaussian-based evaluation system but substantially changing the search strategy for the optimal E1 value. Instead of calculating the robot coefficient for the entire possible range using an exhaustive approach, this method employs an iterative bisection algorithm. The procedure begins by dividing the external axis range into n parts, calculating the coefficient for these n values, and identifying the one with the best score. This value becomes the center of a new range, smaller than the previous one by a factor of n, and the cycle is repeated for m iterations until convergence. This approach significantly reduces the number of evaluations required compared to the linear search method, accelerating the entire function calculation process while maintaining the same accuracy in determining the optimal external axis position.

3.3.3 Center of the external axis feasible range

The third method examines the external axis range reachable by the robot for a given position. Once this range is identified, the E1 value output from the optimization function will be the center of this range. In cases where multiple ranges are available, a situation that can occur when the selected point is physically very close to the slide, the algorithm is designed to select the range closest to the previous point, although this functionality has not yet been tested in real scenarios with multiple range configurations.
This method is designed to optimize the positioning of the external linear axis, thereby maximizing the maneuverability of the robot during operation. This aspect is particularly critical when handling large workpieces that require precise linear movements, especially in environments where the robot must operate in close proximity to substantial machinery, such as presses, bending machines, and panel benders. By strategically selecting the position of the external axis, the method helps the robot avoid operating in critical or constrained zones, which could otherwise compromise both the accuracy and speed of its movements.

Chapter 4

Implementation

4.1 Primal-Dual interior-points method

This section describes a basic primal-dual interior-point method [4], that can be applied to the robot external axis optimization problem. Primal-dual interior-point methods are closely related to the barrier method but exhibit several key differences that make them particularly suitable for constrained optimization problems in robotics.
The primal-dual interior-point approach operates with a single-loop iteration structure, eliminating the distinction between inner and outer iterations found in traditional barrier methods. At each iteration, both primal and dual variables are updated simultaneously, providing a more efficient computational framework. This characteristic is particularly advantageous when optimizing the external axis position E1, as it allows for concurrent consideration of both the robot configuration variables and the constraint multipliers.
The search directions in primal-dual interior-point methods are derived from Newton's method applied to modified Karush-Kuhn-Tucker (KKT) equations, which represent the optimality conditions for the logarithmic barrier centering problem. While these primal-dual search directions are similar to those arising in barrier methods, they are specifically tailored to handle both primal and dual variables simultaneously. This approach is particularly relevant for robot optimization problems where multiple constraints must be satisfied, including joint limits, singularity avoidance, and workspace boundaries.

Algorithm 1 Primal-dual interior-point method

given x that satisfies $f_1(x) < 0, \ldots, f_m(x) < 0$, $\lambda \succ 0$, $\mu > 1$, $\epsilon_{\text{feas}} > 0$, $\epsilon > 0$.
repeat
  Determine t. Set $t := \mu m/\hat{\eta}$
  Compute primal-dual search direction Δy_{pd}
  Line search and update:
    Determine step length $s > 0$ and set $y := y + s\Delta y_{\text{pd}}$
until $\|r_{\text{pri}}\|_2 \leq \epsilon_{\text{feas}}$ **and** $\|r_{\text{dual}}\|_2 \leq \epsilon_{\text{feas}}$ **and** $\hat{\eta} \leq \epsilon$

A significant advantage of primal-dual interior-point methods is that the primal and dual iterates are not required to be feasible throughout the optimization process. This flexibility is crucial in robotics applications, where maintaining strict feasibility at every iteration can be computationally expensive or even impossible due to the complex constraint structure of robotic systems.

Primal-dual interior-point methods often demonstrate superior efficiency compared to barrier methods, especially when high accuracy is required, as they can exhibit better than linear convergence rates. This performance advantage is particularly valuable in real-time robotics applications, where computational efficiency directly impacts system responsiveness. For fundamental problem classes including linear, quadratic, second-order cone, geometric, and semidefinite programming, customized primal-dual methods consistently outperform barrier methods.

In the context of robot external axis optimization, where the objective function may involve Gaussian-based configuration quality metrics and the constraint set includes joint limits and singularity avoidance conditions, primal-dual interior-point methods offer a robust framework for finding optimal solutions. The method's ability to handle both feasible and marginally feasible problems makes it particularly suitable for robotics applications where constraint boundaries are frequently encountered.

Although for general nonlinear convex optimization problems remain an active area of research, they show considerable promise for robotics applications. The integration of these methods with the Gaussian-based evaluation systems described previously provides a powerful optimization framework for determining optimal external axis positions while avoiding kinematic singularities and maintaining operational efficiency.

The primal-dual interior-point method addresses constrained convex optimization problems of the standard form.

$$\begin{aligned} &\text{minimize} && f_0(x) \\ &\text{subject to} \quad && f_i(x) \leq 0, \quad i = 1, \ldots, m, \\ &&& Ax = b, \end{aligned} \tag{4.1}$$

where the decision variable $x \in \mathbb{R}^n$, and the problem satisfies the following regularity conditions:

- The objective function $f_0(x)$ is convex and twice continuously differentiable
- The inequality constraint functions $f_i(x)$ are convex and twice continuously differentiable
- The matrix $A \in \mathbb{R}^{p \times n}$ defines the linear equality constraints with $\text{rank}(A) = p < n$
- The vector $b \in \mathbb{R}^p$ specifies the right-hand side of the equality constraints

The feasible region is defined as the intersection of the inequality constraint region $x : f_i(x) \leq 0, i = 1, \ldots, m$ and the affine subspace $x : Ax = b$. We assume throughout that the problem satisfies Slater's constraint qualification, ensuring that strong duality holds and the Karush-Kuhn-Tucker (KKT) conditions are necessary and sufficient for optimality.

$$\begin{aligned} Ax^* = b, \quad f_i(x^*) \leq 0, \quad i = 1, \ldots, m \\ \lambda^* \succeq 0 \\ \nabla f_0(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla f_i(x^*) + A^T \nu^* = 0 \\ \lambda_i^* f_i(x^*) = 0, \quad i = 1, \ldots, m. \end{aligned} \tag{4.2}$$

The central innovation of the primal-dual approach lies in its treatment of the complementary slackness conditions. Instead of directly enforcing $\lambda_i f_i(x) = 0$, the method solves a perturbed system where these conditions are relaxed to $\lambda_i f_i(x) = -1/t$ for a positive barrier parameter t.

$$f(x) = \begin{bmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{bmatrix}, \quad Df(x) = \begin{bmatrix} \nabla f_1(x)^T \\ \vdots \\ \nabla f_m(x)^T \end{bmatrix}. \tag{4.3}$$

The modified KKT conditions are expressed through the residual function $r_t(x, \lambda, \nu)$ which consists of three components arranged as a vector: the dual residual r_{dual}, centrality residual r_{cent}, and primal residual r_{pri}. starting from:

$$r_t(x, \lambda, \nu) = \begin{bmatrix} \nabla f_0(x) + Df(x)^T \lambda + A^T \nu \\ -\operatorname{diag}(\lambda) f(x) - (1/t)\mathbf{1} \\ Ax - b \end{bmatrix} \tag{4.4}$$

Differentiating them for each component:

$$r_{\text{dual}} = \nabla f_0(x) + Df(x)^T \lambda + A^T \nu \tag{4.5}$$

$$r_{\text{cent}} = -\operatorname{diag}(\lambda) f(x) - (1/t) \cdot \mathbf{1} \tag{4.6}$$

$$r_{\text{pri}} = Ax - b \tag{4.7}$$

Here, $Df(x) \in \mathbb{R}^{m \times n}$ denotes the Jacobian matrix of the constraint functions

$f(x) = [f_1(x), \ldots, f_m(x)]^T$, and $\mathbf{1}$ represents the vector of ones in $\mathbb{R}^m$.
The barrier parameter $t > 0$ controls the accuracy of the approximation to the original KKT conditions. As $t \to \infty$, the centrality residual approaches the true complementary slackness conditions, and the solution of the modified system converges to the optimal solution of the original problem.
The primal-dual method employs Newton's method to solve the nonlinear system $r_t(x, \lambda, \nu) = 0$. The Newton step $(\Delta x, \Delta\lambda, \Delta\nu)$ is computed by solving the linear system:

$$\begin{bmatrix} \nabla^2 f_0(x) + \sum_{i=1}^m \lambda_i \nabla^2 f_i(x) & Df(x)^T & A^T \\ -\text{diag}(\lambda) Df(x) & -\text{diag}(f(x)) & 0 \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta\nu \end{bmatrix} = - \begin{bmatrix} r_{\text{dual}} \\ r_{\text{cent}} \\ r_{\text{pri}} \end{bmatrix} \tag{4.8}$$

This system has a specific block structure where the coefficient matrix consists of three main blocks. The upper left block contains the Hessian H and the constraint Jacobian $Df(x)^T$ and A^T. The middle block involves the diagonal matrices formed from the dual variables and constraint values. The lower left block contains the equality constraint matrix A. The right-hand side vector consists of the negative dual, centrality, and primal residuals. where the Hessian of the Lagrangian is given by

$$H = \nabla^2 f_0(x) + \sum_{i=1}^m \lambda_i \nabla^2 f_i(x) \tag{4.9}$$

The KKT matrix possesses several important properties that ensure the method's effectiveness:

1. **Symmetry:** The matrix is symmetric by construction
2. **Block structure:** The structured form enables efficient solution techniques
3. **Regularity:** Under appropriate assumptions, the system remains non-singular throughout the optimization process

The solution of this system requires careful attention to numerical stability, particularly when the barrier parameter t becomes large or when constraints become nearly active.
Since iterates in the primal-dual method may not satisfy the original constraints, the traditional duality gap cannot be used to measure optimality. Instead, the method employs a surrogate duality gap. The surrogate duality gap is defined as

$$\hat{\eta}(x, \lambda) = -f(x)^T \lambda. \tag{4.10}$$

This quantity serves as a proxy for the true duality gap and guides the selection of the barrier parameter through the relationship. The barrier parameter is updated according

to the rule $t = \mu m/\hat{\eta}$. where $\mu > 1$ is a fixed parameter (typically $\mu = 10$) and m is the number of inequality constraints. This updating rule ensures that the barrier parameter increases appropriately as the algorithm approaches optimality, maintaining the balance between centering and progress toward the solution.

The complete primal-dual interior-point algorithm integrates the Newton step computation with appropriate line search procedures to ensure convergence while maintaining feasibility conditions.

The primal-dual interior-point algorithm proceeds iteratively as follows. Given an initial point:

$$(x^0, \lambda^0, \nu^0) \tag{4.11}$$

satisfying

$$f_i(x^0) < 0 \text{ and } \lambda^0 > 0, \tag{4.12}$$

and parameters

$$\mu > 1,\ \varepsilon_{\text{feas}} > 0,\ \text{and } \varepsilon > 0, \tag{4.13}$$

The algorithm repeats the following steps until convergence. First, update the barrier parameter using $t = \mu m/\hat{\eta}$. Second, compute the search direction by solving the KKT system for $(\Delta x, \Delta\lambda, \Delta\nu)$. Then, update the variables according to

$$(x, \lambda, \nu) \leftarrow (x, \lambda, \nu) + s(\Delta x, \Delta\lambda, \Delta\nu) \tag{4.14}$$

starting from:

$$y = (x, \lambda, \nu), \quad \Delta y = (\Delta x, \Delta\lambda, \Delta\nu), \tag{4.15}$$

The step size determination in the primal-dual interior-point method requires careful consideration to maintain the strict feasibility conditions that characterize interior-point approaches. Let us denote the residual function evaluated at the updated point $y^+ = (x^+, \lambda^+, \nu^+)$ as r^+ for notational convenience.

The line search procedure begins by establishing the maximum allowed step length that preserves the positivity constraint on the dual variables. This maximum feasible step size is computed as the largest value in the unit interval that ensures all components of the updated dual vector remain strictly positive:

$$s_{\max} = \sup\{s \in [0, 1] \mid \lambda + s\Delta\lambda \succeq 0\} \tag{4.16}$$

$$= \min\left\{1, \min\left\{-\frac{\lambda_i}{\Delta\lambda_i} \mid \Delta\lambda_i < 0\right\}\right\}. \tag{4.17}$$

This formulation prevents any dual variable from becoming negative or zero, which would

violate the fundamental requirements of the interior-point framework.

The backtracking line search algorithm initiates with an aggressive step size of $s = 0.99s_{\max}$, deliberately choosing a value slightly smaller than the theoretical maximum to provide a safety margin. The algorithm then employs a geometric reduction strategy, repeatedly multiplying the current step size by a reduction factor $\beta \in (0, 1)$ until two critical conditions are satisfied.

The first condition ensures that the updated primal variables maintain strict feasibility with respect to the inequality constraints:

$$f(x^+) \prec 0. \tag{4.18}$$

The second condition enforces sufficient decrease in the residual norm, implementing the Armijo condition adapted for the primal-dual context:

$$\|r_t(x^+, \lambda^+, \nu^+)\|_2 \leq (1 - \alpha s)\|r_t(x, \lambda, \nu)\|_2. \tag{4.19}$$

This condition guarantees monotonic progress toward the solution by requiring that each iteration achieves a proportional reduction in the residual magnitude.

The selection of backtracking parameters follows established practices from Newton-type optimization methods. The sufficient decrease parameter α is typically chosen within the range $[0.01, 0.1]$, providing a balance between ensuring meaningful progress and avoiding overly restrictive step size requirements. The reduction factor β is commonly selected from the interval $[0.3, 0.8]$, where smaller values lead to more conservative step sizes but potentially more iterations, while larger values allow more aggressive steps but may require additional backtracking iterations.

From a theoretical perspective, each iteration of the primal-dual interior-point algorithm can be viewed as a single Newton step applied to the modified KKT system $r_t(x, \lambda, \nu) = 0$. However, this Newton step is augmented with domain restrictions that maintain the essential feasibility conditions $\lambda \succ 0$ and $f(x) \prec 0$. Equivalently, we can interpret this as restricting the domain of the residual function r_t to the region where these strict inequalities hold.

The convergence analysis of this line search procedure follows directly from the theoretical framework established for infeasible Newton methods. The same mathematical arguments that guarantee finite termination of the backtracking line search in the infeasible Newton context apply equally to the primal-dual setting. This theoretical foundation ensures that the algorithm will always find an acceptable step size within a finite number of backtracking iterations, provided the Newton direction is well-defined and the current iterate satisfies the domain restrictions.

Finally, check the convergence criteria, the algorithm terminates when all of the following

conditions are satisfied: the primal feasibility condition $|r_{\text{pri}}|2 \leq \varepsilon\text{feas}$, the dual feasibility condition $|r_{\text{dual}}|2 \leq \varepsilon\text{feas}$, and the optimality gap condition $\hat{\eta} \leq \varepsilon$.
The biggest problem with this algorithm qua algorithm is definitely the fact that during the algorithm one has to do the calculation of an inverse matrix and thus it can cause interruptions during its use. for this reason the svd method was used to avoid getting impossible or indeterminate resolutions.
It is important also understand where is the starting point of the algorithm, because often if it is to much near the optimal the result cannot converge.

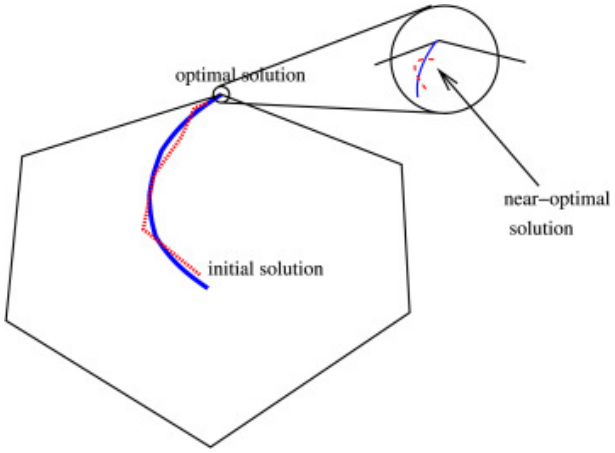


Figure 4.1: Graphic example of Interior Point method

4.2 Simulated Annealing Algorithm

Introduction

Simulated Annealing (SA) is a probabilistic metaheuristic inspired by the annealing process in metallurgy. It is designed to approximate the global minimum of a cost function in a large search space, especially when the function is non-convex or noisy. SA is particularly effective when deterministic methods get stuck in local minima, [5].
In this context, it is used to optimize the configuration of an external robotic axis along a pre-defined trajectory. The goal is to minimize deviation from an optimal value x_{opt}, while keeping the solution within a feasible range and penalizing constraint violations.

Algorithm 2 Simulated Annealing Algorithm

given initial solution x, temperature $T > 0$, cooling rate $\alpha \in (0, 1)$, stopping temperature $T_{\min} > 0$
    and objective function $f(x)$
$x_{\text{best}} \leftarrow x$
repeat
    Generate a neighbor solution x_{new} of x
    $\Delta f \leftarrow f(x_{\text{new}}) - f(x)$
    if $\Delta f < 0$ **then**
        $x \leftarrow x_{\text{new}}$
    else if $\text{random}(0, 1) < \exp(-\Delta f/T)$ **then**
        $x \leftarrow x_{\text{new}}$
    end if
    if $f(x) < f(x_{\text{best}})$ **then**
        $x_{\text{best}} \leftarrow x$
    end if
    $T \leftarrow \alpha T$
until $T < T_{\min}$
return x_{best}

Objective Functions

Since we wanted to compare which axes are best to use within the cost function, two types were created, the first in which we go to compare all 6 axes of the robot with the first reference value, while in the second case we analyze only the difference of the outer axis from its optimal value.

First cost Function

It is defined as the squared distance between the actual and optimal value of the external axis:

$$f(x) = (x - x_{\text{opt}})^2 + P \cdot \chi_{[x<x_{\min} \vee x>x_{\max}]} \tag{4.20}$$

where:

- x is the current value of the external axis,
- x_{opt} is the target value for optimal configuration (e.g., from first trajectory point),
- P is a penalty constant (e.g., 10^6 or 10^{10}),
- $\chi_{[]}$ is an indicator function that equals 1 if the condition is true and 0 otherwise,
- $[x_{\min}, x_{\max}]$ defines the feasible range for the external axis.

the main idea of the cost function is explain better at the top of the chapter 3.1.

Second cost Function

It is defined as the different between the robot joints value for the determine position and its reference:

$$f(x) = \sum_{i=1}^{6} [(x_i - x_{\text{ref}})^2 + P \cdot \chi_{[x_i < x_i, \min \vee x_i > x_i, \max]}] \tag{4.21}$$

where:

- x_i is the current value of the joint i,
- x_{ref} is the target value for optimal configuration (e.g., from first trajectory point),
- P is a penalty constant (e.g., 10^6 or 10^{10}),
- $\chi_{[]}$ is an indicator function that equals 1 if the condition is true and 0 otherwise,
- $[x_i, \min, x_i, \max]$ defines the feasible range for the current joint i.

Simulated Annealing Steps

Simulated Annealing mimics the cooling of materials and accepts worse solutions probabilistically to escape local minima. The algorithm consists of the following steps:

1. Initialization

- Choose an initial solution x_0 (for example, the midpoint of the feasible interval guarantees very good results in all the tests).
- Set the initial temperature T_0 and define the cooling schedule (for example, exponential, i prefer set a little decay to speed up the convergence).
- Set maximum number of iterations $N_{\max}$.

2. Evaluate Cost

Compute the cost of the initial solution using the objective function.

3. Iterative Improvement

For $k = 1$ to $N_{\max}$:

1. Generate a neighbor x' by adding a random perturbation to x:

$$x' = x + \Delta, \quad \Delta \sim \mathcal{U}(-\text{step}, \text{step})$$

where step size typically decreases with temperature:

$$\text{step}(T) = \text{step}_0 \cdot \frac{T}{T_0}$$

2. Evaluate the cost $f(x')$.

3. Calculate cost difference:

$$\Delta E = f(x') - f(x)$$

4. Accept the new solution with probability:

$$P = \begin{cases} 1, & \text{if } \Delta E < 0 \\ \exp\left(-\frac{\Delta E}{T}\right), & \text{otherwise} \end{cases}$$

5. Update the best solution found so far if $f(x') < f(x_{\text{best}})$.

6. Decrease the temperature:

$$T \leftarrow \alpha \cdot T$$

with the cooling rate $\alpha \in (0, 1)$.

4. Termination

The algorithm stops after a fixed number of iterations or when temperature reaches a threshold.

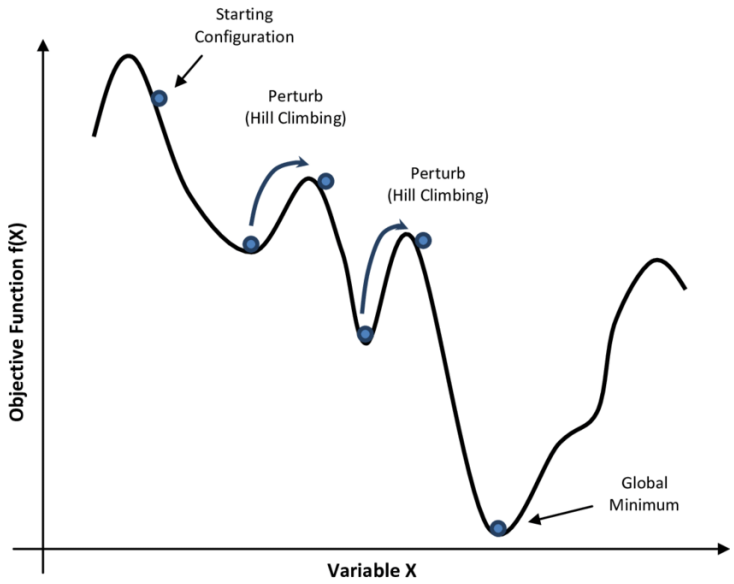


Figure 4.2: Graphic example of Simulated Annealing algorithm

Practical Considerations

In real-world applications, such as robot trajectory optimization:

- The feasible range $[x_{\min}, x_{\max}]$ is dynamically computed for the point in the trajectory.
- The optimal value x_{opt} is often derived from the configuration of the first point.
- The penalty terms ensure that the optimizer respects the actuator limits.

Conclusion

Simulated Annealing is a robust and flexible method for external axis optimization in robotic systems, especially when the search space is non-linear and constraint-laden. Although convergence is not guaranteed, careful tuning of temperature, step size, and penalty factors significantly improves the algorithm's performance.
Moreover, the same starting point can reach different results; in this case they are very similar and do not change the final trajectory travels by the robot.
In this case I do not expect that repeating the optimization starting from the same trajectory will always give me the same result, but close to it, this is because the search for the minimum points occurs with some probability within the cycle.

Chapter 5

Evaluation & Results

This chapter presents the results obtained by applying the methodologies described in Chapter 4, which focus on optimizing robotic motion using the Simulated Annealing algorithm. The implementation was carried out using the C# programming language, targeting the .NET Framework 4.8.0. Visual Studio 2022 was used as the Integrated Development Environment (IDE), offering robust tools for debugging, performance profiling, and user interface development.
The choice of C# and .NET was motivated by their strong support for object-oriented programming, multi-threading, and integration with Windows-based industrial automation systems. This environment also facilitated the development of a modular and maintainable code-base, enabling efficient testing and future scalability of the optimization framework.
The output type is to develop by Windows application, the choice was motivated by the need for a stable, user-friendly environment with rich graphical capabilities. Using the .NET Framework and C#, the application provides a responsive interface and integrates easily with Windows-based tools and external libraries. This approach ensures compatibility with industrial systems, supports offline usage, and facilitates rapid prototyping and testing of optimization algorithms.

5.1 Robot KUKA and linear unit for simulation

The whole project was carried out so that it could be replicated on multiple industrial robots, regardless of brand or size, the important thing being that they are six-axis industrial robots. The simulations were performed on a single type of robot, the KUKA KR 50 R 2100, which is dimensionally presented as a medium-sized one, a maximum reach of 2100 mm, in fact, it is widely used in industry, [6].

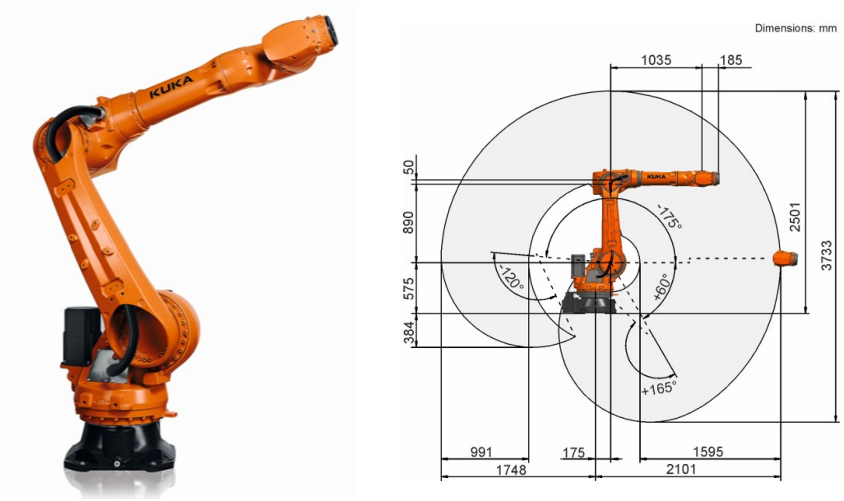


Figure 5.1: KUKA KR 50 R 2100

In addition to it, there is clearly also an external axis connected both physically and kinematically via software. An example of a linear unit might be KUKA KL 4000, in this case with a useful stroke of 5000 mm, [7].

Figure 5.2: KUKA KR 50 R 2100

5.2 Development environment

For the development and validation of this thesis, the simulator provided by Euclid Labs was adopted as the primary environment for testing and experimentation. This simulator, implemented in C#, it is built proprietary libraries developed by Euclid Labs that are specifically designed for trajectory generation and simulation of robotic systems. Its graphical interface presents a 3D rendering of the robotic manipulator, generated from CAD models, offering a visually accurate representation of the robot's physical structure. The user interface includes various interactive controls, referred to as "gadgets," that allow users to manipulate the robot's configuration in real time by adjusting the angular values of individual joints. This functionality allows a wide range of poses and movements to be simulated, enabling a precise analysis of robot kinematic behavior.
As illustrated in Figure 5.3, the interface allows for the import of trajectories acquired from real-world executions or the generation of synthetic paths. Once loaded, these trajectories are displayed in the simulation environment, providing visual feedback on the robot's motion. it shows how the simulator flags problematic sections of a trajectory using warning indicators to highlight points that are unreachable or violate mechanical constraints.
In addition you can also see a version of the KUKA teach pendant, where you can move the robot manually at will, depending on the situation with respect to the robot's axes, the world, or in reference to a designated base.

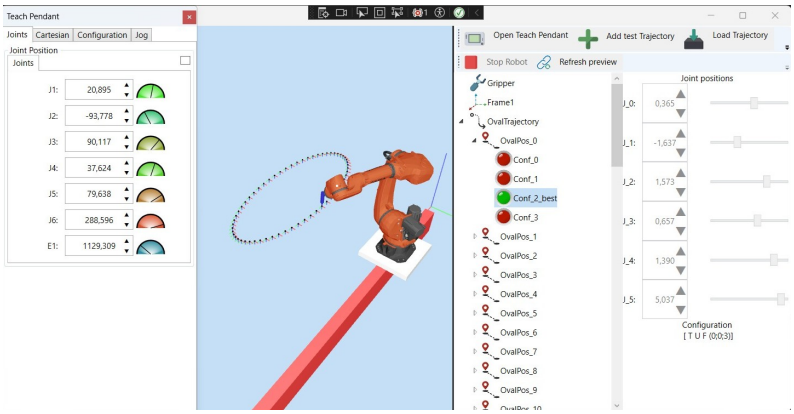


Figure 5.3: User interface with Teach Pendant

Within the scope of this thesis, the existing simulator framework was extended with additional functionalities to support a more in-depth comparison between the simulated robot model and its physical counterpart. These enhancements aimed to improve the fidelity of the simulation, integrate features to monitor deviations between real and virtual robots, and enable the evaluation of various trajectory optimization strategies. This extended tool-set proved essential for iteratively refining the proposed models and validating the theoretical assumptions against empirical observations.

In figure 5.4, it is possible to look at all the information and elements that can be used by importing a trajectory generated directly from the Marvin software.

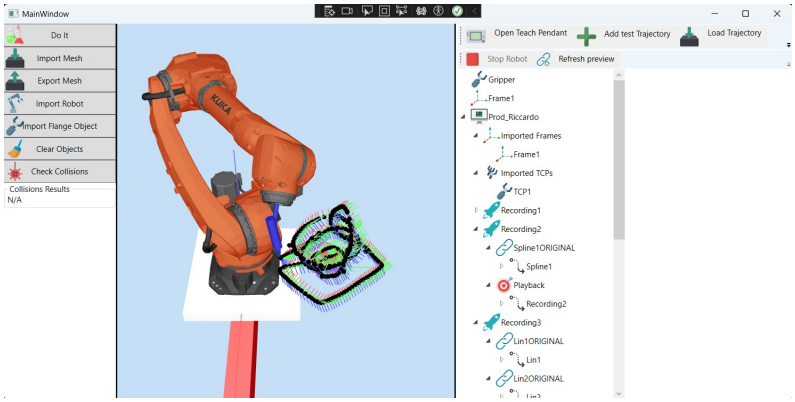


Figure 5.4: User interface with Marvin product

During the course of the dissertation, features were added so that the various trajectories and their necessary optimization methods could be selected to be performed during the various tests.

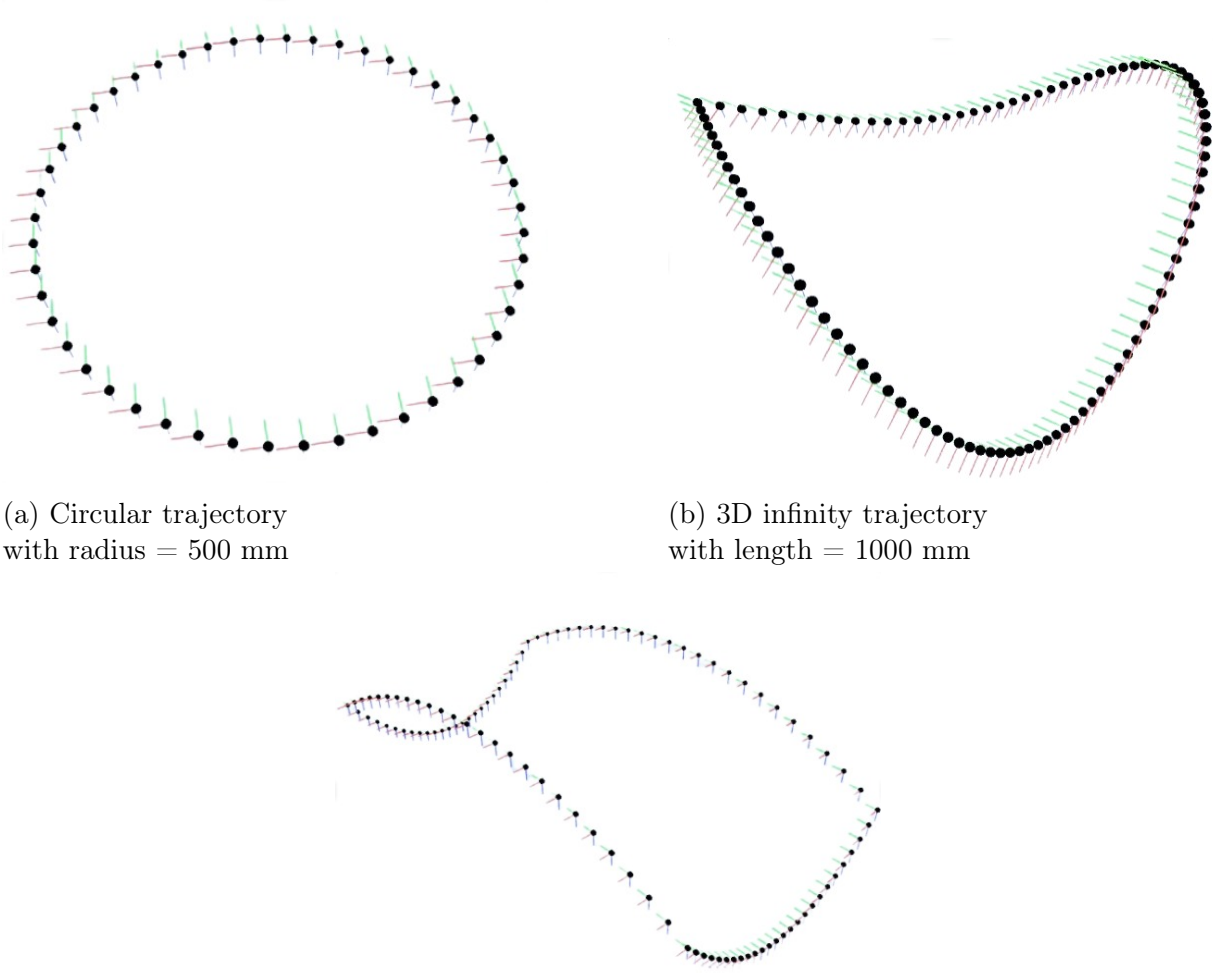
5.3 Trajectories

Clearly, for the simulations to work properly and in accordance with how the initial problem was formulated the trajectories needed to be executable entirely as linear interpolations between points, without any change in the robot's configuration. This requirement stems from the fact that, in practical applications, configuration changes can only be executed through point-to-point (PTP) or soft point-to-point (SPTP) motions, which introduce discontinuities and are not compatible with continuous linear trajectories. Therefore, ensuring a consistent configuration throughout each trajectory was a critical condition for the accuracy and feasibility of the simulated experiments.
However, when a configuration change is unavoidable and depending on how the algorithm is designed, the PTP motion is treated as the start of a new sub-trajectory. Although the robot performs the entire motion without interruption, the underlying system logic handles the change in motion type as a discrete transition. This approach ensures that the linearity constraint is preserved within each segment processed by the algorithm, while accommodating necessary configuration changes in a structured and predictable manner.

5.3.1 Training trajectories

The first trajectories were intended to highlight the correct operations and criticalities during different movements, such as the course of a circular trajectory, or orthogonal to

the outer axis with different heights, and finally with varying TCP orientations throughout the trajectory.



(a) Circular trajectory with radius = 500 mm

(b) 3D infinity trajectory with length = 1000 mm

(c) Paraboloid hyperbolic trajectory with length = 2000 mm and width = 400 mm

Figure 5.5: Training trajectories

5.3.2 Test trajectories

To effectively test the simulation system and validate the behavior of the external axis, two types of trajectories were selected. The first was a complex trajectory derived from Marvin program, which includes varied and articulated movements that challenge the full kinematic chain of the robot. The second trajectory, in contrast, was intentionally simpler and designed to cover a broader range along the central portion of the linear axis (external axis). This was done to observe the behavior of the axis without being constrained by physical end-of-travel limits or software-imposed restrictions. By focusing on the central span of the linear unit, it was possible to maximize movement freedom and

better assess the coordination between the robot's base and the external axis, ensuring reliable positioning and movement within an unconstrained operational window. This strategic choice of test trajectories allowed both stress testing of the control algorithms and detailed analysis under controlled conditions. The third trajectory is intended to highlight the different behavior among the three methods used, through optimization of the same algorithm, along a trajectory that allows the robot in any case to travel almost the entire outer axis and has sections that are perfectly orthogonal to the linear unit.

(a) Circular trajectory with length = 2000 mm

(b) Trajectory composed by Marvin

(c) Paraboloid hyperbolic trajectory with length = 4500 mm and width = 400 mm

Figure 5.6: Test trajectories

5.4 Results

All tests carried out on the trajectories described in the previous section were specifically designed to compare the performance and behavior of the three proposed methods under a range of controlled conditions. Each test scenario was carefully selected to evaluate the methods in distinct yet comparable situations, allowing for a thorough and consistent analysis. The overarching objective was to assess whether the methods produced similar results when subjected to the same input conditions, or whether notable differences emerged in their responses.

By systematically analyzing the outputs and performance metrics of each method, the tests aimed to highlight not only points of convergence, where the methods behaved similarly, but also areas of divergence, which could indicate underlying differences in their structure, assumptions or sensitivity to specific parameters. Such observations are crucial in determining the robustness and generalizability of each approach across different types of trajectory and constraint scenarios.

In addition, the results of these experiments served to validate or challenge the initial hypotheses posed at the beginning of the investigation. In cases where the methods performed in line with expectations, the results reinforced the theoretical foundations and assumptions underlying their design. In contrast, when discrepancies were observed, further investigation was conducted to identify possible causes, be they related to numerical stability, optimization characteristics, or context-specific limitations of a given method.

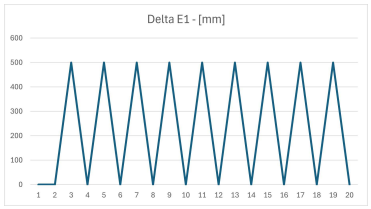
Ultimately, this comprehensive comparison provides a clearer understanding of the strengths and weaknesses of each approach, offering valuable insights into their applicability in real-world settings and laying the groundwork for future improvements or adaptations.

5.4.1 Algorithm comparison with external axis cost function

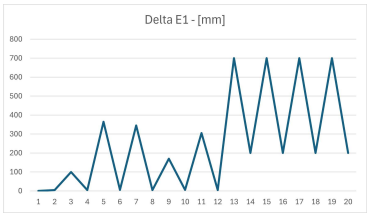
The first set of tests was carried out using the step trajectory, chosen specifically to evaluate the robot's behavior along the entire range of the external axis, including both segments that are parallel and orthogonal to the linear unit. This configuration provides an ideal scenario for assessing how each method manages the movement of the external axis under varying directional constraints.

A key observation can be made from Graph 5.7a and, similarly, from Graph 5.9a. In these cases, the external axis remains stationary during all segments orthogonal to its direction. This behavior occurs because the axis has already reached a position close to the optimal distance from the robot flange. As a result, any additional movement away from this position would increase the value of the cost function being minimized, particularly the one related to the distance from the ideal configuration, which cannot be further reduced within the given constraints.

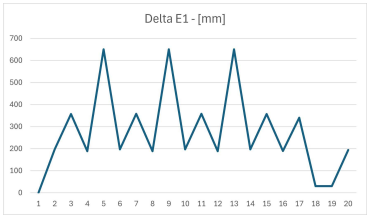
In contrast, the behavior exhibited in the other two cases (Figures 5.8a and 5.8b - 5.8c) reveals a more dynamic use of the external axis. Here, the trajectory can be conceptually divided into two distinct phases. In the first phase, the robot actively searches for an optimal configuration, primarily driven by the orientation requirements of the Tool Center Point (TCP). Once this configuration is reached, the system maintains it consistently for the remainder of the trajectory. This indicates that the method successfully stabilizes around a solution once the initial adjustment phase is complete.
An exception to this trend is observed during the test involving the third method. In this case, the robot initially reaches the limit imposed by the software during the range calculation phase. This constraint leads to a subsequent reduction in the displacement between consecutive points, limiting the movement of the external axis and slightly altering the expected behavior. Despite this limitation, the general strategy of exploiting the external axis to achieve an optimal configuration remains evident.
Overall, across all test cases, it is clear that the external axis is utilized effectively and, in most scenarios, to its full potential. Each method demonstrates a tendency to maximize the advantage offered by the linear unit in aligning the robot toward its optimal configuration, with only minor variations introduced by software limitations or method-specific characteristics.



(a) Delta of E1 during step trajectory and with method 1

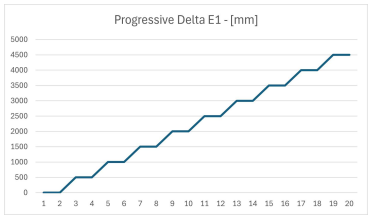


(b) Delta of E1 during step trajectory and with method 2

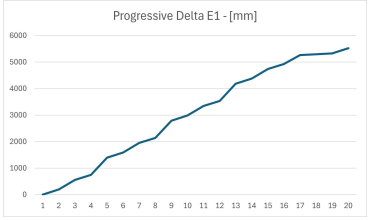


(c) Delta of E1 during step trajectory and with method 3

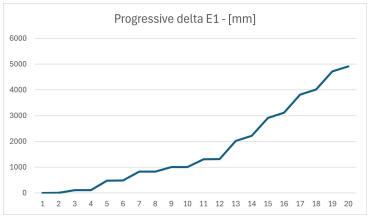
Figure 5.7: Simulated Annealing



(a) Progressive Delta of E1 during step trajectory and with method 1



(b) Progressive Delta of E1 during step trajectory and with method 2



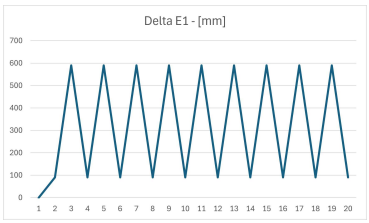
(c) Progressive Delta of E1 during step trajectory and with method 3

Figure 5.8: Simulated Annealing

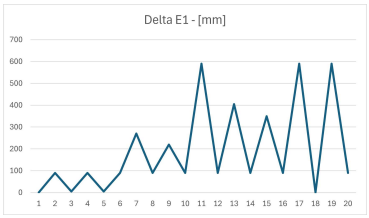
A notable outcome observed across the tests is that changing the type of algorithm does

not lead to substantial variations in the final result. This finding highlights the robustness and consistency of the cost functions employed, as they appear to guide the optimization process toward similar solutions, even when embedded within computational frameworks that differ significantly in nature. Specifically, the two algorithms under comparison, despite their fundamentally different optimization strategies and produce nearly identical results in terms of trajectory shaping and external axis behavior.

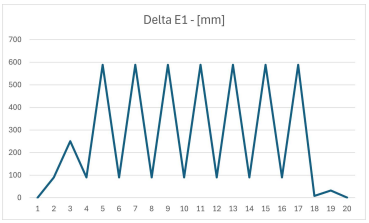
This consistency underscores an important point: the influence of the cost function dominates over the specific implementation details of the optimization method, at least within the tested scenarios. Whether the algorithm follows a deterministic or heuristic path, the formulation of the objective function imposes a strong directional influence on the solution space, effectively steering both methods toward configurations that minimize the same performance criteria.



(a) Delta of E1 during step trajectory and with method 1

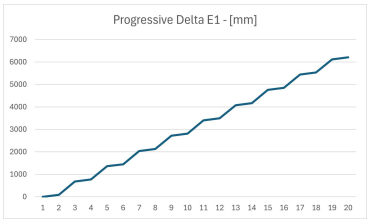


(b) Delta of E1 during step trajectory and with method 2

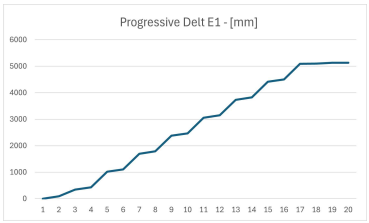


(c) Delta of E1 during step trajectory and with method 3

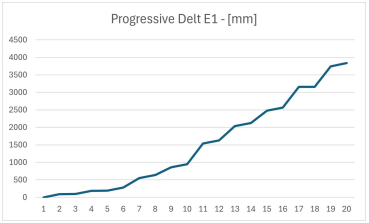
Figure 5.9: Interior Point



(a) Progressive Delta of E1 during step trajectory and with method 1



(b) Progressive Delta of E1 during step trajectory and with method 2



(c) Progressive Delta of E1 during step trajectory and with method 3

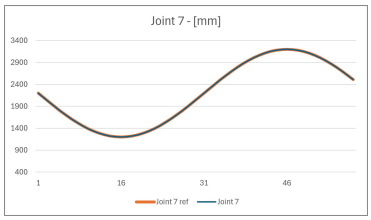
Figure 5.10: Interior Point

In particular, when examining the behavior of the first and third methods, it becomes evident that they exhibit highly stable and regular characteristics throughout the trajectory. The profiles generated by these methods are marked by long segments of constant or nearly constant values, suggesting that once the optimal configuration is found, the system maintains it without unnecessary adjustments. This stability can be interpreted as an indication of convergence toward a global or sufficiently optimal local minimum, where

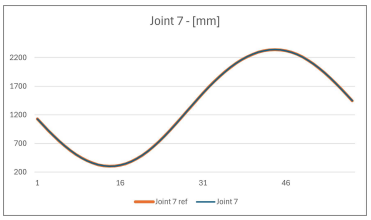
further improvements offer negligible gains or would require significant computational effort with little added value.
The results reinforce the effectiveness of the chosen objective formulations and provide strong evidence of their adaptability across different optimization paradigms. This lays a solid foundation for their use in more complex scenarios and in systems where computational resources or algorithmic constraints may vary.

5.4.2 Methods comparison with external axis cost function

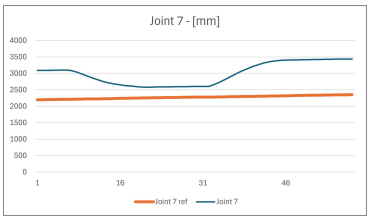
The second test was conducted on an oval trajectory positioned physically closer to the linear unit. This setup was intentionally designed to "force" the robot into adopting a diagonal posture relative to the direction of motion, preventing it from curling inward into mechanically inefficient or suboptimal configurations.
By placing the trajectory near the base, the robot was challenged to find a balance between maintaining proper TCP orientation and avoiding overly compressed joint positions. This scenario allowed us to evaluate how effectively each method handles constrained workspaces and trajectory induced limitations.
The results show that the more adaptive methods guided the robot toward diagonal configurations, particularly during extended segments of the path. This positioning helped avoid joint saturation, ensured smoother transitions, and made better use of the external axis. Overall, the test highlights the methods' ability to maintain efficient and stable behavior even under more restrictive spatial conditions.



(a) External Axis during Oval Trajectory with method 1, Simulated annealing



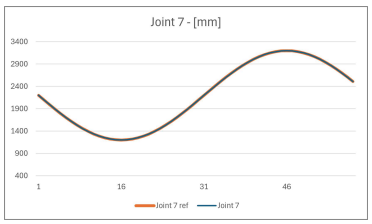
(b) External Axis during Oval Trajectory with method 2, Simulated annealing



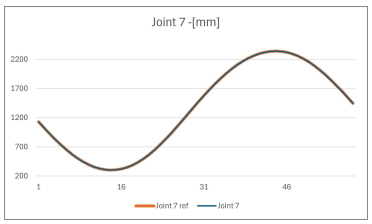
(c) External Axis during Oval Trajectory with method 3, Simulated annealing

Figure 5.11: Simulated annealing with cost function 3.1.1

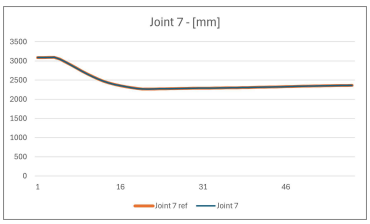
Once again, it can be observed that the first two algorithms behave in a closely aligned, almost symbiotic manner. This is evident from the results shown in Graphs 5.11a-5.11c, and 5.12a-5.12b, where both methods produce a similar trend. In contrast, the third method exhibits a slightly different behavior, as seen in Graphs 5.8c and 5.11c. However, its deviation remains relatively limited and does not significantly affect overall performance. The first and second methods generate an almost sinusoidal profile, which can be attributed to the trajectory starting near the external axis and appearing to return to that



(a) External Axis during Oval Trajectory with method 1, Interior point



(b) External Axis during Oval Trajectory with method 2, Interior point



(c) External Axis during Oval Trajectory with method 3, Interior point

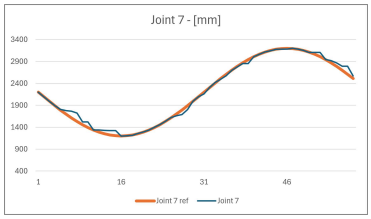
Figure 5.12: Interior point with cost function 3.1.1

initial configuration during repeated loops. This behavior stems from the methods' design, which fits each trajectory point independently of the available range of the external axis (E1). This contrasts to the third method, which takes the constraints of E1 more directly into account, influencing its motion pattern.

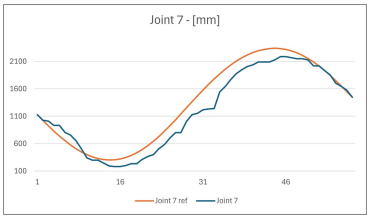
These results suggest that different methods exhibit varying levels of sensitivity to workspace constraints, particularly outer axis constraints. While some approaches prioritize trajectory fitting and maintain a uniform response regardless of system boundaries, others incorporate mechanical constraints more explicitly into their optimization process. This distinction reveals important differences in how each method balances trajectory accuracy with feasibility. These insights are invaluable in choosing an appropriate algorithm for applications where physical constraints or flow limits significantly affect performance.

5.4.3 Methods comparison with robot joints cost function

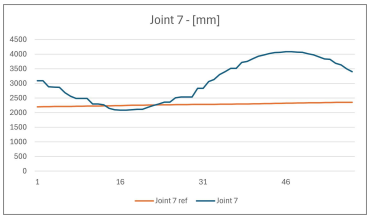
The analysis highlights how modifying the cost function influences the behavior of all methods, encouraging a convergence toward more efficient axis configurations, especially in the presence of curved trajectories. Interestingly, this change leads to a partial alignment in behavior across the methods, with trends resembling those of method 2. Variability in the external axis movement, particularly in stochastic approaches, reflects the algorithm's acceptance of near-optimal solutions, emphasizing the role of probabilistic mechanisms in refining local adjustments. Meanwhile, the intermediate response of method 3, illustrates its ability to balance between deterministic smoothness and adaptive flexibility, offering a compromise in both trajectory shape and axis usage. These observations underscore how the formulation of the objective function plays a critical role in guiding not only performance, but also the qualitative characteristics of the resulting motion.



(a) External Axis during Oval Trajectory with method 1, Simulated annealing



(b) External Axis during Oval Trajectory with method 2, Simulated annealing



(c) External Axis during Oval Trajectory with method 3, Simulated annealing

Figure 5.13: Simulated annealing with cost function 3.1.2

Chapter 6

Conclusion

The comprehensive evaluation and testing of the proposed robotic motion optimization methods have yielded significant insights that advance our understanding of trajectory optimization for industrial robots with external axes. The results presented in this chapter demonstrate the effectiveness of the developed approaches and provide clear guidance for their practical implementation.

Cost Function Supremacy

The most significant conclusion emerging from this research is the dominance of cost function design over algorithm selection in determining optimization outcomes. Across all test scenarios, both the Simulated Annealing and Interior Point methods converged to nearly identical solutions when employing the same objective function. This finding fundamentally challenges the conventional emphasis on algorithm sophistication, instead highlighting that well-formulated cost functions provide robust optimization guidance regardless of the underlying computational strategy.

This discovery has profound implications for industrial applications, suggesting that development efforts should prioritize the careful design of objective functions tailored to specific operational requirements rather than focusing primarily on algorithm selection. The consistency observed across different optimization paradigms demonstrates that the mathematical formulation of the problem constraints and objectives carries far more weight than the specific path taken to reach the solution.

The demonstrated equivalence in solution quality between more complex stochastic methods and deterministic approaches suggests that algorithm selection can be based primarily on implementation constraints, computational resources, and real-time requirements rather than concerns about optimization effectiveness.

Stability and Convergence Characteristics

Methods 1 and 3 consistently exhibited remarkable stability throughout all test trajectories, characterized by extended segments of constant or near-constant values once optimal configurations were achieved. This behavior indicates successful convergence toward global or sufficiently optimal local minima, with the systems maintaining these configurations without unnecessary adjustments that could introduce instability or inefficiency.
The observed stability patterns suggest that these methods effectively balance exploration and exploitation phases of optimization. Initially, the algorithms actively search for optimal configurations, particularly driven by Tool Center Point orientation requirements. Once suitable configurations are identified, the systems demonstrate excellent maintenance of these solutions throughout the remainder of the trajectory execution.

External Axis Optimization

All tested methods demonstrated exceptional utilization of the external linear axis, consistently leveraging this additional degree of freedom to achieve optimal robot configurations. The external axis was employed to its full potential in most scenarios, with clear evidence of the system actively seeking positions that minimize the overall cost function while maintaining trajectory accuracy.
Particularly noteworthy is the behavior observed during orthogonal trajectory segments, where the external axis often remained stationary after reaching positions close to optimal distance from the robot flange. This demonstrates intelligent optimization behavior, as any additional movement would increase the value of the cost function without providing the corresponding benefits. The system's ability to recognize and maintain these optimal positions throughout extended trajectory segments underscores the effectiveness of the proposed optimization framework.

Algorithm-Specific behavioral patterns

While the overall performance remained consistent across methods, subtle yet meaningful differences emerged in their behavioral characteristics. Methods 1 and 2 exhibited patterns of symbiotic behavior, particularly evident in their generation of nearly sinusoidal profiles for cyclic trajectories. This behavior stems from their independent optimization of each trajectory point without explicit consideration of external axis range constraints.
Method 3, in contrast, demonstrated greater sensitivity to workspace constraints, incorporating mechanical limitations more explicitly into its optimization process. This approach resulted in slightly different motion patterns but maintained overall performance quality, illustrating the method's ability to balance trajectory accuracy with feasibility constraints.

Adaptability under constraints

The methods showed remarkable adaptability when confronted with challenging spatial constraints, such as trajectories positioned near the linear unit that forced the robot into potentially suboptimal configurations. The algorithms successfully guided the robot toward diagonal postures, effectively avoiding joint saturation while maintaining proper Tool Center Point orientation and ensuring smooth transitions throughout the trajectory. This adaptability extends to various trajectory types, from simple geometric paths to complex industrial sequences generated by commercial software. The consistent performance across this diverse range of test cases demonstrates the robustness and generalizability of the proposed approaches for real-world industrial applications.

Cost Function impact on method convergence

The modification of cost functions from external axis optimization to robot joint optimization caused all methods to exhibit more similar behavioral patterns, particularly converging toward characteristics similar to those in Method 2. This observation reinforces the primary conclusion regarding the dominance of the cost function and demonstrates how the formulation of the objective function can be used to guide the overall behavior of the system toward the desired operational characteristics.

Final Synthesis

This research has successfully demonstrated that effective optimization of the robotic trajectory with external axes can be achieved through careful design of cost functions, with algorithm selection of secondary importance to the quality of the solution. The proposed methods exhibit excellent stability, adaptability, and use of external axes in various operational scenarios, providing strong validation of their industrial deployment. The comprehensive evaluation presented in this chapter establishes a clear framework for practical implementation of these optimization strategies in industrial robotics applications, with particular emphasis on the critical role of objective function formulation in achieving optimal system performance. These findings contribute significantly to the field of robotic motion planning and provide valuable guidance for future developments in industrial automation systems.

6.0.1 Future works

The comprehensive results presented in this thesis establish a solid foundation for robotic trajectory optimization with external axes, demonstrating the effectiveness of the proposed methods across various scenarios. However, the findings also illuminate several

promising directions for future research and development that could significantly expand the practical applicability and industrial impact of this work.

Industrial Software Integration

The most immediate next step is implementing the developed optimization methods within the Marvin software environment for real-world testing. While simulation results demonstrate promising theoretical performance, validation through actual industrial projects is essential to confirm practical effectiveness.
This implementation should begin with integrating a single method into the Marvin platform, followed by thorough testing on real industrial projects. This real-world validation will provide crucial insights into performance under actual manufacturing conditions, including robot dynamics, environmental disturbances, and system tolerances that cannot be fully captured in simulation. The feedback from industrial operators will enable parameter refinement and optimization framework fine-tuning to meet specific industry requirements.

Multi-Axis System Extensions

Independent External Axis Integration

A significant opportunity lies in extending the system to incorporate multiple external axes operating independently of the robot's base. The current work focuses on a linear unit moving synchronously with the robot, but many industrial applications would benefit from additional rotary tables, indexing units, or other positioning systems operating independently.
Integrating an additional non solidary external axis would introduce new degrees of freedom and complexity to the optimization problem. This extension requires enhanced cost functions that effectively balance multiple external axes utilization while maintaining trajectory accuracy and system stability. The coordination between independent axes presents challenges in synchronization, workspace management, and collision avoidance that warrant investigation.

Stationary Robot with External Axis Synchronization

Another promising direction involves investigating systems where the robot remains stationary while synchronizing with external positioning axes. This configuration is particularly relevant for applications such as welding along conveyor systems, painting operations on moving assemblies, or precision manufacturing tasks where the workpiece is manipulated by external positioning systems.

In such configurations, the optimization challenge shifts from managing robot mobility to coordinating internal joint movements with external axes carrying the workpiece. This approach could lead to improved precision, reduced robot component wear, and enhanced overall system efficiency through specialized cost functions and constraint handling methods.

CAD/CAM Integration and Object-Aware Optimization

The integration of the optimization framework with CAD/CAM systems represents a transformation opportunity for advancing industrial robotics applications. Current optimization methods primarily consider geometric and kinematic constraints, but real-world manufacturing scenarios involve complex interactions with physical objects, fixtures, and environmental obstacles.

Object-aware optimization algorithms would incorporate 3D workspace models, including workpieces, fixtures, tools, and safety barriers, directly into constraint formulation. This integration would enable automatic collision avoidance, safe clearance maintenance, and tool accessibility optimization during trajectory generation. The system would require sophisticated geometric reasoning capabilities, real-time collision detection, and the ability to handle dynamic workspace changes, ultimately leading to reduced programming time, improved safety, and automatic adaptation to production requirement changes.

Mixed Trajectory Optimization

Real-world industrial applications rarely consist of purely linear interpolation trajectories. Instead, they typically involve complex sequences combining linear motions, point-to-point movements, and circular interpolations, each with different kinematic and dynamic characteristics. Extending the optimization framework to handle such mixed trajectory types represents a crucial step toward practical industrial deployment.

The challenge lies in developing optimization methods that seamlessly handle transitions between different motion types while maintaining overall trajectory optimality. This requires sophisticated constraint handling for smooth transitions, appropriate velocity profiles, and consistent external axis utilization across motion type changes. Point-to-point motions present unique challenges as they allow robot reconfiguration but must be carefully coordinated with external axis positioning, while circular interpolations introduce complexity related to path accuracy and tool orientation management throughout curved trajectories.

Conclusion

The future directions outlined above represent natural extensions of the work presented in this thesis, building upon the established foundation of effective cost function design and robust optimization methods. The systematic pursuit of these research directions would significantly advance robotic motion planning and contribute to more capable, efficient, and adaptable industrial automation systems.

Real-world validation through Marvin software integration remains the most immediate priority, providing the empirical foundation necessary for pursuing more advanced extensions. The successful completion of these future research endeavors would establish a comprehensive framework for robotic trajectory optimization capable of addressing the full complexity of modern industrial manufacturing environments.

Bibliography

[1] Smart Factory. *Cobots vs robots: Which one should you choose?* Accessed: 2025-06-21. 2025. URL: https://smartfactory.konicaminolta.eu/cobots-vs-robots/?utm_source=chatgpt.com.

[2] DEVELOP LLC. *Cobots vs industrial robots in manufacturing*. Accessed: 2025-06-21. 2025. URL: https://develop-llc.com/insights/cobots-vs-industrial-robots-in-manufacturing/?utm_source=chatgpt.com.

[3] Inbolt. *Cobots 101: Understanding collaborative robotics*. Accessed: 2025-06-21. 2025. URL: https://www.inbolt.com/resources/cobots-101-understanding-collaborative-robotics?utm_source=chatgpt.com.

[4] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. 7th printing with corrections, 2009. Cambridge, UK: Cambridge University Press, 2004. ISBN: 978-0-521-83378-3. URL: https://www.cambridge.org/9780521833783.

[5] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. "Optimization by simulated annealing". In: *Science* 220.4598 (1983), pp. 671–680.

[6] KUKA Deutschland GmbH. *KR 50 R2100 – Dati tecnici e specifiche*. Version V3.1. Manuale tecnico per il robot industriale KR IONTEC. Augsburg, Germania, 2023. URL: https://www.kuka.com.

[7] KUKA Deutschland GmbH. *KL 4000 – Manuale tecnico e specifiche di installazione*. Version V1.1. Unità lineare per applicazioni con robot KUKA. Augsburg, Germania, 2024. URL: https://www.kuka.com.

Appendix A

Singular Value Decomposition

Given a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, the Singular Value Decomposition expresses it as:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{A.1}$$

where:

- $\mathbf{U} \in \mathbb{R}^{m \times m}$ is an orthogonal matrix containing the left singular vectors
- $\mathbf{V} \in \mathbb{R}^{n \times n}$ is an orthogonal matrix containing the right singular vectors
- $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ is a diagonal matrix with singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{\min(m,n)} \geq 0$

A.0.1 Processing Algorithm

Step 1: Compute SVD

Decompose the matrix $\mathbf{A}$ using a numerically stable SVD algorithm (e.g., Golub-Reinsch, divide-and-conquer):

$$[\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}] = \text{SVD}(\mathbf{A}) \tag{A.2}$$

Step 2: Rank Determination

Define a tolerance threshold τ based on machine precision and matrix dimensions:

$$\tau = \max(m, n) \cdot \sigma_1 \cdot \epsilon_{\text{machine}} \tag{A.3}$$

where $\epsilon_{\text{machine}}$ is the machine epsilon (typically 2.22×10^{-16} for double precision).
Determine the numerical rank:

$$r = \sum_{i=1}^{\min(m,n)} \mathbf{1}(\sigma_i > \tau) \tag{A.4}$$

where $\mathbf{1}(\cdot)$ is the indicator function.

Step 3: Construct Pseudoinverse of Singular Values

Create the pseudoinverse of the diagonal matrix:

$$\mathbf{\Sigma}^+ = \text{diag}\left(\sigma_1^+, \sigma_2^+, \ldots, \sigma_{\min(m,n)}^+\right) \tag{A.5}$$

where:

$$\sigma_i^+ = \begin{cases} \frac{1}{\sigma_i} & \text{if } \sigma_i > \tau \\ 0 & \text{if } \sigma_i \leq \tau \end{cases} \tag{A.6}$$

Step 4: Compute Matrix Pseudoinverse

The Moore-Penrose pseudoinverse is given by:

$$\mathbf{A}^+ = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^T \tag{A.7}$$

For square, full-rank matrices, this reduces to the standard matrix inverse.

A.0.2 Implementation Considerations

Condition Number Assessment

The condition number in the 2-norm is:

$$\kappa_2(\mathbf{A}) = \frac{\sigma_1}{\sigma_r} \tag{A.8}$$

where σ_r is the smallest non-zero singular value. A large condition number indicates potential numerical instability.

Regularization (Optional)

For ill-conditioned problems, Tikhonov regularization can be applied:

$$\sigma_i^{\text{reg}} = \frac{\sigma_i}{\sigma_i^2 + \lambda^2} \tag{A.9}$$

where $\lambda > 0$ is the regularization parameter.

A.0.3 Computational Complexity

The SVD computation has complexity $\mathcal{O}(\min(mn^2, m^2n))$ for an $m \times n$ matrix, which is higher than standard Gaussian elimination $\mathcal{O}(n^3)$ for square matrices, but provides

superior numerical stability and handles rank-deficient cases.

A.0.4 Verification

After computing $\mathbf{A}^+$, verify the pseudoinverse properties:

$$\mathbf{A}\mathbf{A}^+\mathbf{A} = \mathbf{A} \tag{A.10}$$

$$\mathbf{A}^+\mathbf{A}\mathbf{A}^+ = \mathbf{A}^+ \tag{A.11}$$

$$(\mathbf{A}\mathbf{A}^+)^T = \mathbf{A}\mathbf{A}^+ \tag{A.12}$$

$$(\mathbf{A}^+\mathbf{A})^T = \mathbf{A}^+\mathbf{A} \tag{A.13}$$