POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Parameterization of robotic welding trajectories from a demonstration

Tesi di Laurea Magistrale in
Automation and Control Engineering - Ingegneria dell'automazione

Author: **Giovanni Fracassi**

Student ID: 10799851
Advisor: Prof. Paolo Rocco
Co-advisors: Prof. Andrea Maria Zanchettin, Ing. Isacco Zappa
Academic Year: 2021-22

Abstract

The rise of mass customization in the industrial scenario is leading to the necessity of important cooperation between humans and robots. In this context, the ability to program the robot easily, quickly, and reliably is bringing new programming frameworks, like the so-called Programming By Demonstration (PbD). PbD requires the extraction from demonstration of suitable skill parameters, for the execution of the task, but its deployment in industrial scenarios presents a series of drawbacks. In fact, the selection of the movements of the robot is defined by the operator through suitable interfaces, provided by robot manufacturers. The operator will define the specific robot movements saving waypoints, while the robot is moved in freedrive. As a consequence, a robot program is automatically generated. Anyway, the robot program is constrained to the position and orientation of the objects on the scene when the demonstration is performed. In this Thesis, we target the problem of programming a cobot in an easy and intuitive way, for a welding task. The method is able to parameterize the welding trajectory taught by the operator with a single kinesthetic demonstration, extracting target positions, orientations, and velocities of the end effector. The parameterized trajectory is then translated into robot code, executed through a combination of linear and circular movements, and suitable tool reorientations. Furthermore, the parameterized trajectory will be extended for the welding of multiple workpieces of the same shape on the welding table, adapting the skill parameters according to the situation.
The method, named ABB GoFa SmartWeld, is validated on ABB GoFa CRB 15000.

Keywords: Programming by Demonstration, Kinesthetic demonstration, Trajectory parameterization, Welding task

Abstract in lingua italiana

L'ascesa della personalizzazione di massa nello scenario industriale ha portato alla necessità di un'importante cooperazione fra umani e robot. Da questo punto di vista, l'abilità di programmare un robot in maniera facile, veloce e affidabile ha reso possibile lo sviluppo di nuove tipologie di programmazione, come la cosiddetta Programmazione per Dimostrazione (PbD). La dimostrazione effettuata in PbD necessita la definizione di opportuni parametri utili nella fase di esecuzione dell'azione complessa. Tuttavia, l'implementazione della PbD nello scenario industriale presenta degli svantaggi. Difatti, la selezione di movimenti del robot viene definita dall'operatore, attraverso specifiche interfacce fornite dai produttori di robot. L'operatore, mentre muove il robot in freedrive, ne definirà i movimenti salvando opportunamente dei waypoint. Di conseguenza, si genera automaticamente un programma robot. Tuttavia, questo programma è vincolato alla posizione e all'orientamento degli oggetti presenti sulla scena al momento della dimostrazione. In questa Tesi, affrontiamo il problema di programmare un robot collaborativo in maniera facile e intuitiva, in una operazione di saldatura. Il metodo è in grado di parametrizzare, attraverso una sola dimostrazione cinestetica, una traiettoria di saldatura eseguita da un operatore, estraendo parametri di posizioni di destinazione, orientamenti e velocità dell'end effector del robot. La traiettoria parametrizzata viene poi tradotta in codice robot, ed eseguita attraverso una combinazione di movimenti lineari, circolari, e re-orientamenti del tool. In aggiunta, la traiettoria parametrizzata verrà estesa alla saldatura di più oggetti della stessa forma presenti sul tavolo di saldatura, adattando i parametri estratti a seconda della situazione.
Il metodo, chiamato ABB GoFa SmartWeld, viene validato tramite un ABB GoFa CRB 15000.

Parole chiave: Programmazione per Dimostrazione, Dimostrazione cinestetica, Parametrizzazione della traiettoria, Operazione di saldatura

Contents

Introduction

The new framework of Industry 4.0 has led to new important shifts in the industrial scenario. In terms of production, the variability of the customers' demand is increased, resulting in an important passage from mass production to mass customization. The key enabling technologies required to support this shift include smart sensors, cloud computing, augmented reality, cybersecurity, big data analytics, and collaborative robotics.
In Small-Medium Enterprises, characterized by a high mix and low volumes of products, the integration of Industry 4.0 can be enhanced by a correct deployment of collaborative robots, thanks to their capability to adapt to flexible production processes. Anyway, this integration is limited by numerous factors. The cobot is affordable only if the number of tasks that it will be able to manage inside the factory is relevant. Otherwise, their application to a limited number of tasks makes the cobot less affordable than human workers. In addition, product customization leads to the need for fast programming and re-programming of the cobot. Nonetheless, the poor presence of a workforce with advanced skills in terms of cobot programming leads to the necessity to find new intuitive techniques to program the cobot. In this sense, the necessity to program the cobot intuitively and fast is leading to the development of new programming techniques, like the so-called Programming By Demonstration (PbD).
The popularity of PbD relies on the capability to transfer new skills to a robot through a demonstration of the task, without the need for any code-based knowledge of the user. The demonstration in robot PbD can take various forms. Kinesthetic demonstration aims to physically move the cobot's joints in order to execute the demonstration, while the robot records the relevant data through its sensors. Instead, Learning by Observation is focused on the cobot that observes, through a vision system, an operator performing the demonstration. In both cases, the robot is able to reproduce the demonstrated task, once the suitable skills parameters, required for the specific skill, are provided to the robot's skill set. However, PbD implementation in industrial scenarios presents some challenging aspects. In order to avoid time-consuming procedures, the cobot should be able to extract all the required features of the task from just a single demonstration, the so-called one-shot-learning. Furthermore, when the operator is performing a demonstra-

tion in kinesthetic teaching, specific waypoints must be manually saved. These waypoints are referred to specific positions and orientations of the end effector defined during the demonstration, and they are fundamental in order to execute the task. Once the demonstration has been performed, a robot program that contains robot instructions related to the specific waypoints is generated. Unfortunately, the robot program generated is only valid in the same scene of the demonstration, in terms of positions and orientations of the objects. So, if the position and orientation of the objects on the scene change, the robot program will be no longer valid.

0.1. Thesis purpose

The aim of this thesis is to find a feasible method to:

- Extract skill parameters from a single, kinesthetic demonstration of a specific continuous industrial task
- Adapt those parameters in different situations

In particular, we are looking for a method that is able to parameterize a suitable welding task, for a given workpiece of arbitrary shape. So, the method should be able to learn the velocity, position, and orientation of the end effector along the shape of the workpiece, from one demonstration, without the need to save specific waypoints during the demonstration. Furthermore, the method should highlight accurately the most important aspects of the demonstration, and select the parameters consequently, neglecting the errors performed by the teacher, like fluctuations of the end effector or discontinuous movements.

0.2. Thesis achievements

The results of the developed algorithm are here briefly synthesized. The algorithm is able to:

- discriminates between a linear movement of the end effector, a circular one, or its re-orientation, or arbitrary combinations of them, and defines a parameterization procedure of the demonstrated trajectory, i.e., extract automatically the skill parameters suitable for the execution of welding task in a plane. In particular, the end effector target that corresponds to a variation of the geometry of the considered workpiece, or a tool re-orientation, and the mean velocity of the demonstration will

be extracted.

- Adapt the extracted parameters, and consequently the move commands, in a scene composed of multiple workpieces of the same shape, in arbitrary numbers, positions, and orientations.

In addition, the method is able to send the collected parameters in suitable commands that the robot is able to interpret and execute, i.e., the so-called Move commands. The method will cope with the fluctuation typical of a kinesthetically taught trajectory and adjust the overall trajectory, maintaining the end effector at a given speed, offset, and orientation with respect to the workpiece, to grant the correct execution of the task. The execution of the task will be performed by evaluating the mean value of the speed defined by the operator in the demonstration, because of its importance in the final quality of the welded workpiece. The method manages the welding of a trajectory on a plane, i.e., in two dimensions, utilizing as z coordinate just its value in the starting point of the demonstration.

0.3. Thesis structure

The rest of this thesis is organized as follows. Chapter 1 provides an extended description of the major challenges addressed in this thesis project, and the literature review. In Chapter 2, an overview of the welding system is presented, including the data preprocessing steps required for the implementation of the algorithm.
In Chapter 3 the parameterization algorithm is described, while a possible extension of the method, for the parameterization of multiple workpieces on the scene, is described in Chapter 4. Chapter 5 presents the validation stage of the algorithm. Finally, Chapter 6 outlines the conclusions of the thesis project and future studies.

1 | Collaborative robots in Industry 4.0

1.1. Industry 4.0

The evolution of industry up to nowadays has gone through three big steps, also called industrial revolutions. From the use of machines introduced by the invention of the steam engine in the late 80s of the eighteenth century to the introduction of mass production and electricity, up to the integration of computers and automation, the evolution of the factory has reached a new important milestone [1].
The new era of modern factories relies on the so-called Industry 4.0, characterized by deep intercourse between the systems that run the production processes of the industry, aiming for their real-time synergy through a strong usage of sensors and networks. The new elements of this paradigm include collaboration between machines, tools, and operators, support of informatic systems, and new goals in terms of efficiency and energy saving, with the introduction of the model of the smart factory. Consequently, the future goals of Industry 4.0 are radically different from the ones of the previous industrial era. The main pillars of this new revolution can be summarized in Table 1.1 [2].

Table 1.1: Comparison between industry 3.0 and industry 4.0

Characteristic	Industry 3.0	Industry 4.0
Processes	Automation	Autonomous decision making
Industry defining technology	Industrial robotics	Collaborative robots
Production planning	Demand forecasting	On-Demand manufacturing
Alignment	Interconnection of production processes	Interconnection of the whole value chain
Variation	Delimited variation	Individually unique products
Goal	Efficiency	Flexibility
Base for revenue model	Selling products	Servitization

The main shift from the production point of view is the passage from mass production to mass customization, defined by an increase in the amount of variability of customers' demands. As a result, the manufacturing processes are personalized to meet these various requirements. To enable this new concept of industry, some supporting technologies must

be introduced, commonly known as key enabling technologies, which include collaborative robotics [3].
In principle, the integration of the Industry 4.0 framework is independent of the size of the company, but its implementation in small and medium-sized enterprises (SMEs) is constrained by numerous factors. The importance of the SMEs-Industry 4.0 integration is driven by the fact that SMEs represent 99,8 % of the European economy today [4]. The natural difficulties of the aforementioned integration derive from the complexity of the overall system, the specificity of the resources needed to interconnect the entire company, the limited financial resources to invest in new technologies, and the lack of ability in terms of human resources to implement new concepts, skills, knowledge, and innovation. The SMEs-Industry 4.0 integration can be speeded up with the introduction of collaborative robots, which are able to easily interoperate with several kinds of devices and systems in broader environments, harvest and transfer big amounts of heterogeneous data, and increase the efficiency of the overall processes. Lastly, their ability to adapt to flexible production processes is consistent with the high mix and low volume scenario, which is the backbone of the SME's business idea [5]. In conclusion, the challenges of the correct integration between Industry 4.0 and SMEs pass through collaborative robotics, and its correct deployment in modern industrial scenarios, which is not as straightforward as it may seem.

1.2. Collaborative Robotics

The important shift in the industrial framework has encouraged the seek for the correct balance between automation and flexibility, and between efficiency in repetitive tasks and the capability of process adaption in the mass customization era. All these conditions have brought to the focus on the Human-Robot Collaboration (HRC) discipline, where humans and robots cooperate jointly in solving a collaborative task [2]. Collaborative robots (cobots) are designed to interact actively with a human operator in a given workspace, and they are characterized, compared to industrial robots, by:

- Repeatability of the same order of magnitude. Industrial robots and cobots present repeatability in the order of tenths of a millimeter
- Lower price in terms of commissioning and equipment, thanks to the absence of fences, required instead in the case of industrial robots

Instructive, in this sense, is Table 1.2, which shows a comparison between four different tasks (assembly, placement, handling, picking), executed respectively by humans, cobots,

and industrial robots. Cobots represent a feasible bridge between the dexterity and flexibility of humans, and the high repeatability of industrial robots, especially in those tasks where a high payload is not mandatory, like assembly. Instead, for traditional pick and place tasks, traditional industrial robots may be preferred in terms of speed of execution, precision, and payload [6].

Table 1.2: Comparison between humans and robots in typical industrial tasks

	Human operator	Collaborative system	Traditional robot
Assembly	High dexterity and flexibility	Combines human dexterity with robot capabilities	Dexterity/Flexibility could be unreachable
Placement	High dexterity	Commercial cobots have lower repeatability	High repeatability and payload
Handling	Product weight restricted	Typical cobots have low payload	High payload and speed
Picking	Product weight restricted	Typical cobots have low payload	High payload and repeatability

The flexibility of the processes introduced by the mass customization scenario may require a workforce with advanced skills, in order to meet the rapidly evolving requests from the customers. Cobots are able to fill the "skill gap" caused by the poor presence of this kind of worker, thanks to the lower skill level required to program them. [7]. Even if the above description of the cobot capabilities seems exactly in line with the SMEs' needs, their integration is still challenging, because of two main factors. The first one is the differential cost of the cobot in relation to the number of tasks that it will be able to execute inside the factory. In fact, even if the cobots' cost is definitely lower than the one of the industrial robots, their application to a small number of tasks makes the cobot less affordable than human workers. The second factor is the level of skills required by the worker, in particular in terms of cobot programming. Product customization leads to the need for fast programming and re-programming of the cobot, so new cobot programming techniques are required, in order to accelerate their integration in the industrial scenario and lower the skill gap of the entry-level workers. Furthermore, cobot programming requires online programming techniques, because of the unpredictability and stochasticity introduced by the human's behavior in the same workspace; instead, the industrial robot requires traditional off-line programming techniques, because no human interaction is taking place [2].

1.3. Robot programming ways for non-expert end-users

The common ways to program the cobots can be roughly divided into two macro-categories. The first one includes all the communication-based methods between the operator and the cobot, like gesture, speech, and particularly user interfaces (UI). The second one relies

on learning methods, where the cobot actually learns from a user demonstration or via trial, error, and feedback. In particular, through the UI, the user can program all the possible cobot's actions offline. Between the UI methods, we can list:

- Programming via teach pendant, a classic programming method via waypoints through a device called teach pendant, that comes together with the robot. This kind of programming method is cobot-specific, that is every different cobot has its own programming language, based on the manufacturer
- Task Level Programming (TLP), where the UI allows a user to compose, and parameterize robot skills, in order to compose a sequence of skills, called task. Anyway, this hierarchy-based approach can be time-consuming [8].
- Offline programming, where manufacturers' simulation environments, like ABB Robotstudio, or V-REP, can be used to test programs and methods.

Of the learning-based programming techniques, the most interesting in terms of possibilities, capabilities, and challenges is the so-called Programming by Demonstration (PbD), where a cobot can actively learn and execute a demonstrated task by a human teacher. This method is highly popular, due to the promising intuitiveness and convenience, and to the low level of skills required for the programming step.
Programming by Demonstration (PbD), also known as Learning from Demonstration (LfD), is a robot programming method based on the robot observing, or physically executing through teaching, the demonstration of a task. The attraction of this method is manifold. First, PbD is a very intuitive method and aims to reduce or remove at all the robot programming of a task. In fact, executing a demonstration instead of writing complex codes increase the accessibility of the end-user to the technology. So, overall, the process will be faster and more flexible, because there's no need to call an external solution integrator every time the cobot needs re-programming. Then, the implementation of PbD on a robot can help its understanding of a task, especially using correct and wrong demonstrations to obtain a good solution for the accomplishment of the demonstrated task [9]. Finally, the robot can understand how to execute a task by extracting hints from the operator. [10].
The demonstration in robot PbD can take various forms. One option, called kinesthetic demonstration is to physically guide the cobot's joints in compliant mode, to execute the demonstration, while the robot records the relevant data coming from its sensors. Kinesthetic teaching is very fast and very easy to implement, because of the built-in integration of the compliant mode in the cobot software, and the user has always complete knowledge of where the robot end effector is located. However, it's not suitable for very precise or

highly dynamic demonstrations, due to the physical constraints of the robot's workspace and the difficulty to move around voluminous cobots. Another option to perform the demonstration is the so-called Learning by Observation, where the cobot observes the demonstrated task performed by a human teacher, through a vision system. It's the easiest way from the teacher's point of view, but the mapping from the user to the cobot's actions is uncertain, because of the different dynamic and kinematic properties of the cobot with respect to the human teacher [11]. In literature, the aforementioned issue is the so-called correspondence problem, which is very present in learning by observation, while is simplified in kinesthetic teaching [2]. The different kinds of demonstrations and the correspondence problem are pictured in Figure 1.1 [11]. However, some challenging

Figure 1.1: From left to right: observational learning, where the cobot uses a vision system to watch the demonstration; kinesthetic teaching, where the cobot is physically moved by the operator; correspondence problem, for cobots with different embodiments and boxes of different sizes

aspects of the PbD must be underlined. Central in this sense is the number of demonstrations, i.e., the quantity of data to provide to the cobot. The user should, in principle, supply different demonstrations, in terms of features, precision, and variation, in order to let the cobot discriminate between the quantitative and qualitative aspects of the task. This job can be sometimes tedious, and time-consuming, and always has a drawback in terms of cost, especially in industrial practice. Therefore, the research is focusing on the so-called one-shot learning, or one-shot demonstration, where the cobot is able to extract all the relevant features from just a single demonstration [12]. One-shot learning in PbD framework allows to save important amounts of time every time that a new task must be demonstrated. Implementing a method that requires too many demonstrations can represent an important disadvantage for on-site factories, which may cause an increase in costs and downtime of the plant. Once the demonstration is given and the robot under-

stands what actions are required, the robot should execute the task. Here lies another important aspect of the PbD: the operator will save specific waypoints when performing a demonstration in kinesthetic teaching. Anyway, the code generated will be strictly related to the demonstration, in terms of position and orientation of objects on the scene, and position and orientations of the robot end effector. For execution, it is then fundamental that the robot understands "how" the task has been executed, through the extraction, from demonstration, of the correct set of parameters needed for each action. One of the most important aspects of parameterization is that if the cobot fails to perform a task after parametrization, the parameter may be the cause of this failure, and so it needs to be changed [2]. To this purpose, the following set of examples from Figure 1.2 will clarify the so-called parameterization problem [13]. The first example describes a robot that should place an object in the box, a classic pick-and-place task. In order to correctly accomplish the task, the correct orientation of the end effector (i.e., our parameter) should be a "top grasp". The second example instead shows a robot that should place the object into a shelf with a ceiling. In this case, the suitable parameter is a "side grasp", because the "top grasp" can cause a failure of the performed task. The concept of parameter is broader

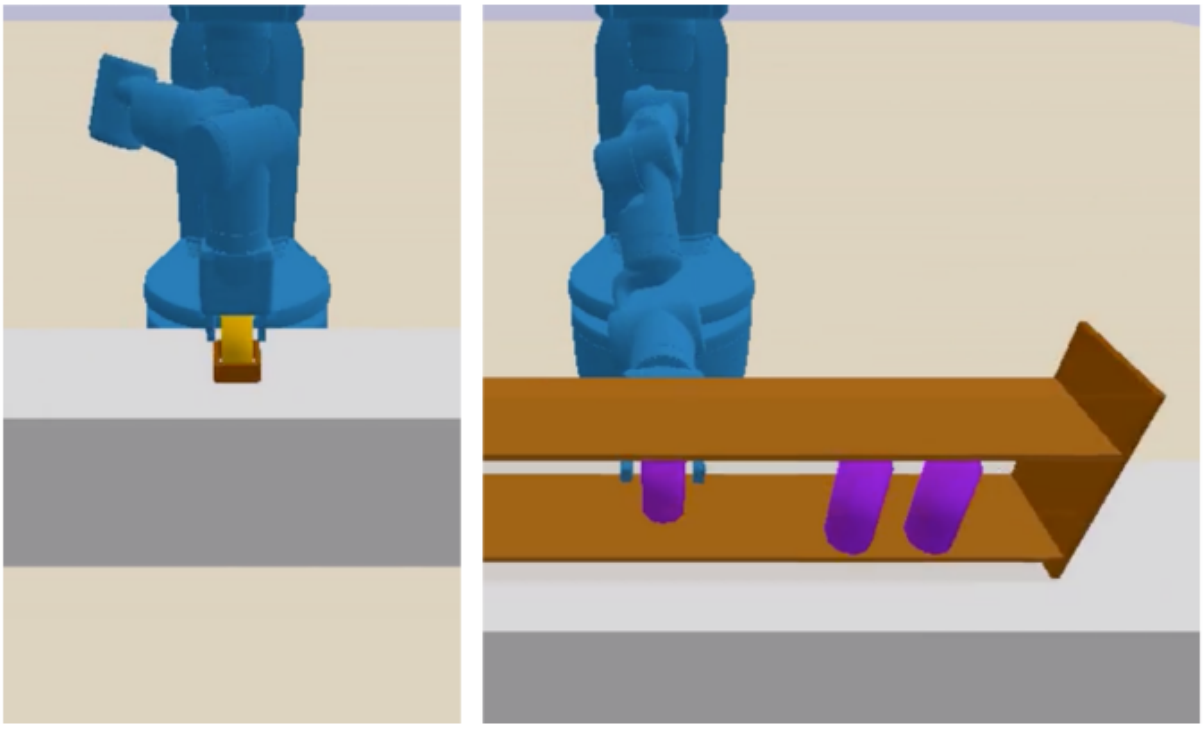


Figure 1.2: The parameterization problem for different tasks

than just the pose selection for a pick and place task of the aforementioned example. In relation to the given task, these parameters can assume a large variety of meanings, like objects with specific features on the scene (color, shape, dimension), kinematic entities

like position, speeds, or accelerations, or even forces, according to the required situation.

1.3.1. Skill Model

Once the overall picture of the cobot's programming methods, especially on the PbD, has been presented, it is finally possible to formalize rigorously the following essential concepts:

- Task, is a complex human action, that the robot learns from the demonstration, and wants to replicate. Each task is composed of a sequence of skills. More formally, a task is a complete robot program, and it is defined as a goal that the robot wants to achieve in its environment [14].
- Skill, is the high-level building block of the task. They are software blocks that can be composed to allow the robot to complete the task, and their most important feature is generality, i.e., they must be re-usable in most of all the scenarios that the robot can encounter. Examples of skills are actions like "pick", "place", "weld", "insert" [15].
- Skill primitives, are the low-level building blocks of the single skill. They refer to the real-time control loops of the robot and are characterized by a single operation, like "open the gripper", or "move the robotic arm to a given pose" [16].
- Skill parameters, the parameters needed in input to the skill. The generality of the skill is not casual, in fact, the specificity of each skill to the related task is possible thanks to the correct choice of the skill parameters' set. Beside the specificity, the choice of the skill parameters is also important to execute correctly the task [15].

The representation of these concepts is synthesized in the following skill model, in Figure 1.3.

1.3.2. Welding task

It's worth analyzing the possible cobot applications and in particular the related set of suitable parameters needed for the parameterization procedure. We can distinguish between discrete tasks, composed of distinct manipulation-type skills with a specific beginning and end, and continuous tasks, related to cyclic and repetitive continuous movements, with arbitrary beginning and end.
Starting from discrete tasks, cobots can exploit, with respect to human labor, their high

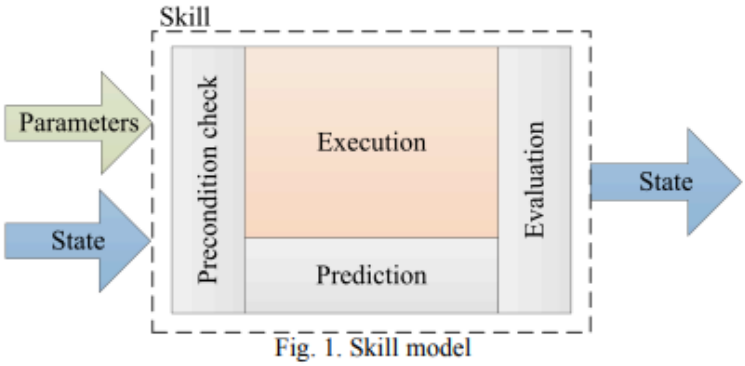


Figure 1.3: Representation of the skill model

speed, and repeatability, to execute highly intensive and time-consuming actions. For tasks like pick and place, palletizing, packing, machine tending, and assembly, the required parameters are typically specific objects and suitable positions and orientations of the cobot's end-effector [17]. In the field of continuous tasks, cobots are able to control adequately parameters like speed or force, respectively in welding and polishing tasks. In particular, in welding tasks, the relevant variables for the correct execution of the task are the following. The welding speed is fundamental to ensure good weld penetration of the welding torch and a low presence of defects on the welded workpiece. Too low speeds can cause excessively wide welds and the presence of large areas of molten material, called weld pool. Too high speed instead can bring thin and narrow welds, and poor weld penetration in the workpiece [18]. Another fundamental parameter is the welding torch angle, as depicted in Figure 1.4. Actually, it is possible to refer to two different angles; the first

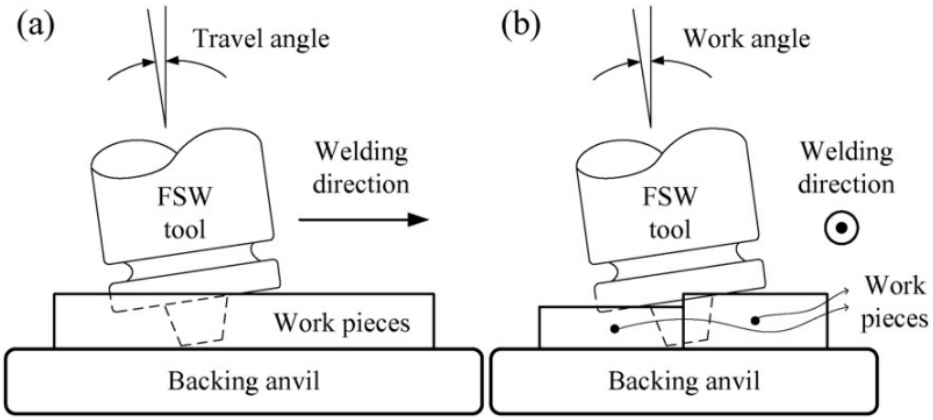


Figure 1.4: Travel angle on the left, work angle on the right

one is the angle between the torch and the line of travel, i.e., the direction against which the welding is executed, which can be seen as a backward tilting of the torch. This angle

is called travel angle. The second one is the angle between the torch and the workpiece, called work angle, which can be seen as a sideway tilting of the torch [19]. Of course, from the cobot's perspective, both two angles refer to the orientation of the end effector of the robot during the execution of the task. Ideally, a defect-free weld zone outcome of the welding process is fostered by a zero travel angle, i.e., a perpendicular welding [20]. So this set of parameters constitutes the skill parameters required for the execution of the task. Table 1.3 explains the different cobot applications in relation to the referred skill parameters set.

Table 1.3: Skill parameters referred to the related industrial task

Task	Skill parameters
Welding	End effector speed, orientation, and trajectory
Press tending	End effector orientation and target position
Quality inspection	End effector position and orientation
Pick and place	Object feature and position
Polishing	End effector force, orientation, and trajectory

1.4. State of the art

The parameterization problem has been addressed with various approaches. One popular approach to the problem is the Dynamic Movement Primitives (DMPs), introduced by Ijspeert et al [21]. DMPs model the demonstrated trajectory with non-linear, mechanical, second-order systems, that converge to an attractor, defined by the DMP's parameter g. Thus, the DMPs dynamic is considered stable. In particular, each aspect of the demonstration is mapped into a suitable DMP parameter, like positions, speeds, spring constants, damping constants, and scaling factors.
The assigned value of a non-linear function in the DMP's model is able to represent arbitrary complex movements executed in the demonstration step, and the related DMP's parameters will capture its various skill parameters. DMPs are a method that grants stable dynamics and capabilities to model complex motions, but the extraction of the meaning of their parameters is sometimes cumbersome and very difficult to transfer to the robot side. The next studies describe the parameterization problem through the use of the hybrid impedance-force control, extracting suitable parameters from continuous skills, to design the reference for each controller, and servings of the DMP to model the demonstrated trajectory. Origanti et al. [22] have developed an extraction procedure based on the extraction matrix defined on a compliant frame, that decouples the var-

ious dimensions of the demonstration in terms of force and motion parameters. Then each dimension is modeled through DMP and coupled with the relative control strategy, defined by a hybrid force-position control, for the execution of the task. This method does not present a satisfactory mapping procedure of the parameters, and it's run only in simulation mode. Pastor et al. [23] have defined a linear regression model to evaluate the non-linear function of the DMPs, in order to adapt the skills and the skill parameters to the required situation. However, the method does not explain how those DMP parameters are selected, and how the change of the non-linear function affects the DMP parameter variation. Kober et al. [24] proposed a DMP modeling of the demonstration in order to drive, with each force and position trajectory reference retrieved from DMP, a hybrid impedance-force controller. The method extracts the suitable parameters defining a reference frame through a scoring system but requires multiple demonstrations, thus is not suitable for our purpose. Gao et al. [25] proposed to extract force skill-based parameters through the constraints of the task, and the learning procedure is based on Gaussian Mixture Model. However, the method requires multiple demonstrations, and it's not task-specific. Conkey et al. [26] instead use DMP to model a dynamic reference frame, aligned with the desired force required by the specific skill, through task constraints. Once the parameters are extracted, a suitable hybrid controller is defined.
A second approach is represented by Differentiable Programming, a method useful for a given program to optimize its parameters, to enforce the execution of a given task. In particular, these parameters can be calculated and optimized via gradient descent-based techniques, through automatic differentiation (AD). Alt et al.[27] define a method that executes a gradient-based neural network inversion technique to infer skill parameters directly from data. Anyway, even if this method is suitable to optimize the given parameters of the task, it requires frequent execution of the task during the optimization phase, and the entire process must be re-executed when the task changes. So, the method defines an approach to optimize skill parameters, while the extraction and adaptation procedure is not examined. Other approaches are evaluated, considering also manipulation-based tasks, sometimes even in non-industrial scenarios. Ude et al. [28] defined a trajectory-based approach, where each trajectory of the demonstration is related to the related set of skill parameters, stored in a database. For new situations, the new parameter will be used as a query point in the aforementioned database, in order to find the suitable trajectory for the specified task. This method unfortunately does not report a robust procedure to assign each parameter to the related task and requires a multiple set of demonstrations to build a useful database. Wahrburg et al. [29] present the Robotic Assembly Skill method (RAS), centered on the decomposition of an assembly task through an assembly tree, evaluating the skill parameters in suitable reference frames, each one assigned to a

specific object on the scene. The parameterization procedure is not applied in relevant scenarios, relegating the skill parameters just to the related object's material. A different approach is proposed by Ramirez-Amaro et al. [30], where a series of semantic-based rules are developed in order to build a decision tree. Each action of the demonstration is described by the nodes of the tree, while its context is described by the attributes on the edges of the tree. A possible extension of this tree to the related skill parameters for each task-context combination has been evaluated, resulting in a hard coding problem, that may lead to stalled configurations of the robot.
A completely different field of research is proposed by Kober et al. [31], based on Reinforcement Learning. The paper models the trajectory of the demonstration through DMPs, mapping their meta-parameters to different states of the environment. Even if the aim of the work is more centered on our purpose, the high computational effort, the low number of samples available in industrial tasks, and the substantial number of trials needed by the agent make this approach unfeasible. Bullard et al. [32] work on a classifier able to define suitable object parameters and semantic locations of the pick and place task. Since the training phase is defined by different demonstrations, and the testing phase is validated in a non-industrial setting, the paper is out of scope. Kramberger et al. [33] proposed a method for generalizing orientation and force/torque trajectories, and their parameters, in a varying environment, based on Locally Weighted Regression. The fitting problem needs a substantial number of demonstrations in different environmental conditions. For this reason, this method is not suitable for our work. Mitrevski et al. [34] instead sampled skill parameters through different demonstrations, evaluating the success of the execution of the task with that specific skill parameter. Then a Gaussian Process is used for the fitting procedure. The main problem of this approach is the population of the dataset, which needs different demonstrations, and the suitable choice of effective rules that determines if the execution is correct or not. Manschitz et al. [35] instead define a method that demonstrated a task with varying object positions and orientations, generalizing to novel poses through weighting movements in multiple reference frames (Mixture of Attractors). However, the method requires multiple demonstrations. The field of research relative to the welding task executed by collaborative robots is also investigated. Ferraguti et al [36] have developed MyWelder, a collaborative robotic system useful for robot-assisted welding, able to identify movement primitives to reconstruct complex robot movements. Those primitives will be available in the GUI, and the suitable welding points are extracted by physically moving the robot to that specific waypoints. A procedure that extracts and adapts automatically the positions and orientations of the end effector is absent. Furthermore, parameters like speed or tool re-orientations are not automatically retrieved. Hollman et al [37] present a method that, through manual

guidance, can learn through Hidden Markov Model the position and orientation of the robot, as well as its speed and offset with respect to the workpiece. Unfortunately, the method requires the active participation of the user, who inputs external parameters when necessary, supporting the GUI through the whole process. Takarics et al [38] designed a vision system-based approach, centered on the edge detection of the workpieces. Once the edges are detected, a cubic spline will fit these points, to build the entire welding path and retrieve the respective position coordinates. Anyway, the method does not manage the extraction of the welding velocity, which is a fundamental parameter in the execution of the welding task.

2 | Setup

2.1. Welding system overview

The ABB GoFa CRB 15000 is the collaborative robot enabled in the welding system. It's a six-axis articulated robot with a payload of 5 kg and reachability equal to 0.95 meters. A sketch of the GoFa is provided in Figure 2.1, where the left side refers to its dimensions, while the right one refers to its axis of rotations. The GoFa product specifications include

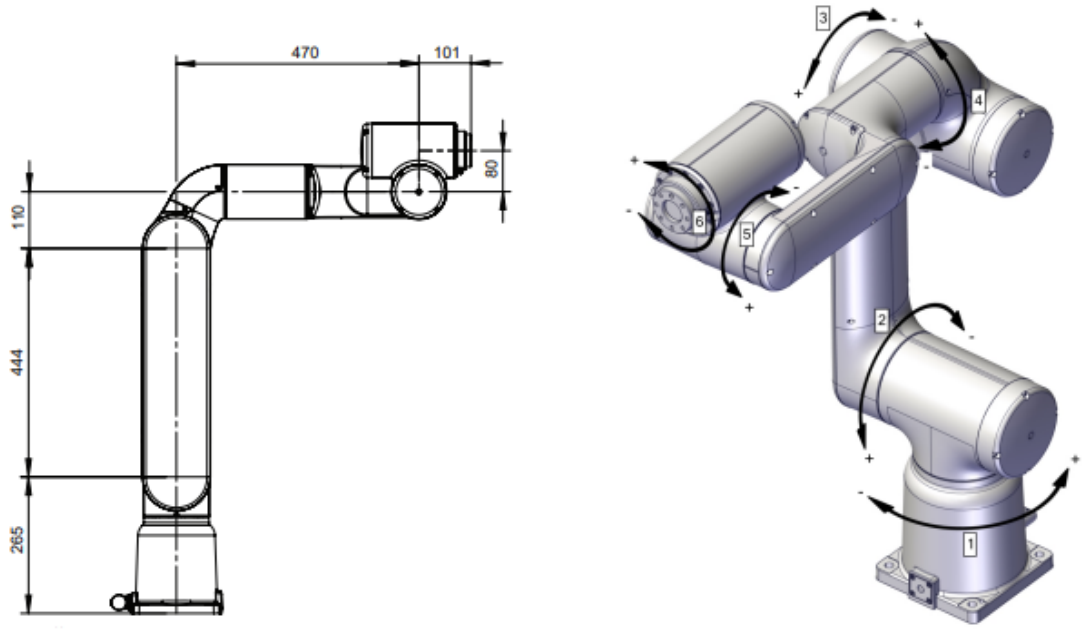
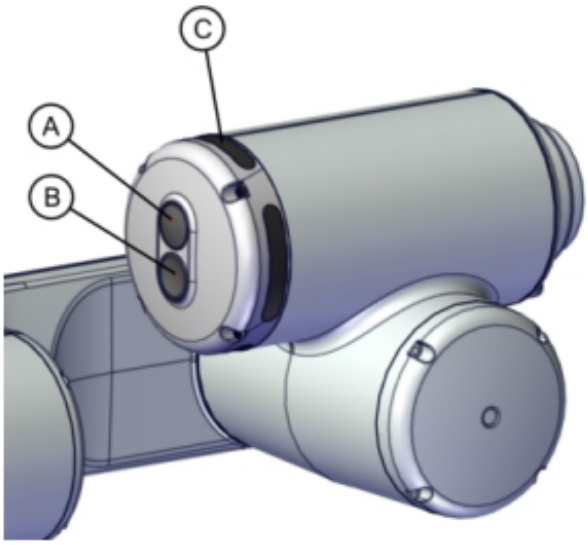


Figure 2.1: GoFa dimensions and axis of rotations

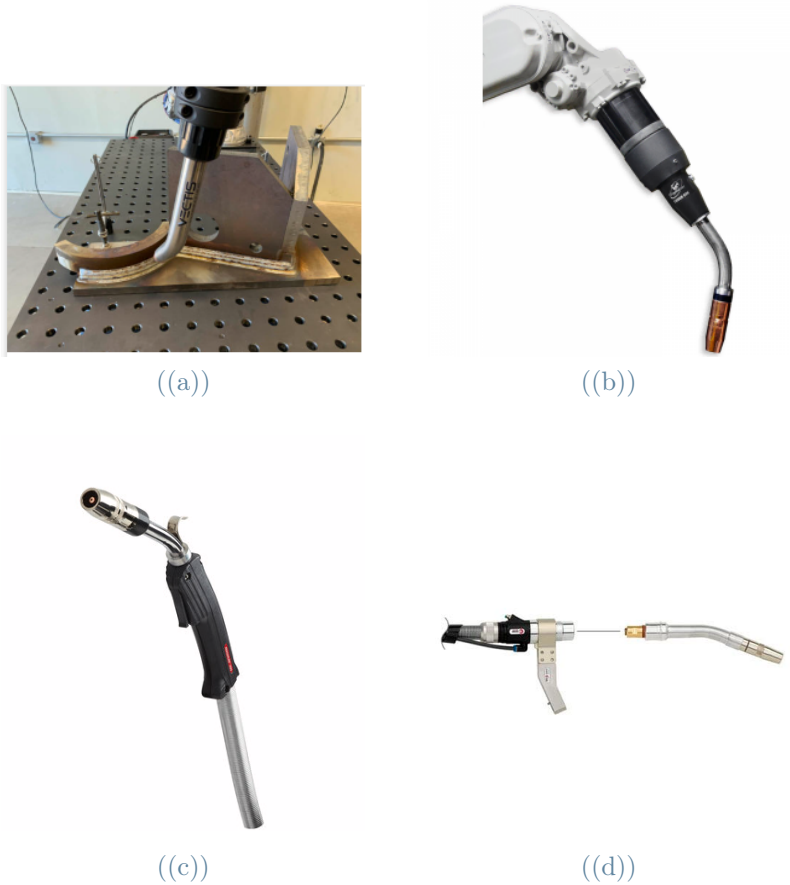
six torque sensors, one for each joint, an accuracy of 6 micrometers, and repeatability of 50 micrometers. The GoFa is equipped also with an Arm Side Interface (ASI), located on axis 5 opposite to the tool flange, that provides a configurable HMI through its buttons. The ASI is represented in 2.2. The GoFa is physically mounted on a fixturing surface and centered concerning the length of the welding table, which measures 1 x 0.5 meters. Finally, on the tool flange a welding tool mockup is mounted. GoFa functionalities include



A	Up button (convex button)
B	Down button (concave button)
C	Light ring

Figure 2.2: GoFa Arm Side Interface

lead through, accessible from the Flex Pendant, which constitutes the basic method to move a robot in kinesthetic teaching; in particular, it allows the free or constrained movement, concerning particular axes, of the joints of the robot. The activation and de-activation of lead-through can be implemented also via ASI, through suitable buttons. In our application, the up button, or convex button, will be responsible of lead through management. Further details of this procedure are presented in Chapter 3. The robot can be moved also in jogging mode, selecting suitable positions on Flex Pendant.
Welding tools for collaborative robots vary in dimension and applicability. For example, Figure 2.3(a) shows an XL welding tool from Vectis Automation, useful for multi-pass welds applications. Figure 2.3(b) shows ABIROB A 360, a welding tool air-cooled with different gas nozzles, in order to ensure a suitable gas coverage of the welded arc. Instead Figure 2.3(c) describes the Tregaskiss Tough Gun MIG welding gun, enforced with copper and equipped with a cooling system and Figure 2.3(d) shows the welding gun Magnum PRO from Lincoln Electric, designed for high heat applications in arc welding.



((a)) ((b))

((c)) ((d))

Figure 2.3: Different types of collaborative robot for welding tools

2.2. Data Preprocessing

This subsection provides insight on the preliminary procedures of data collection and processing. The data collection is performed through socket communication with the robot, which sends values about the current joint position and velocities of the robot during the demonstration of the trajectory. These data constitute the input dataset for the algorithm. The procedure of data collection is managed by the end user through the ASI and a Graphical User Interface. In order to execute the parameterized trajectory, the skill parameters must be organized in a suitable data type, called robtarget, encoding:

- the position of the tool center point (TCP) of the end effector, expressed in millimeters, in a suitable reference frame, which, in our application, will be the robot frame
- the orientation of the end effector expressed in quaternion
- the robot configuration, defined as confdata data type, that expresses the axis con-

figuration of the robot through four-axis values, and for each axis, these values are used to identify the correct quadrant of the robot axis

The last value required is the position of the external axes of the robot, and their value is defined for each correspondent robot axes. Since in our application, external axes are not needed, it will be enough to set the value 9e-9 for all six values of the external axis array.

2.2.1. Forward kinematics

First of all, the forward kinematics must be evaluated, in order to find, starting from the recorded joint values of the demonstration, the correspondent position, and orientation of the end effector. For the GoFa, a suitable sketch of the Denavit-Hartenberg frames is reported in Figure 2.3. Instead, the correspondent Denavit-Hartenberg parameters are

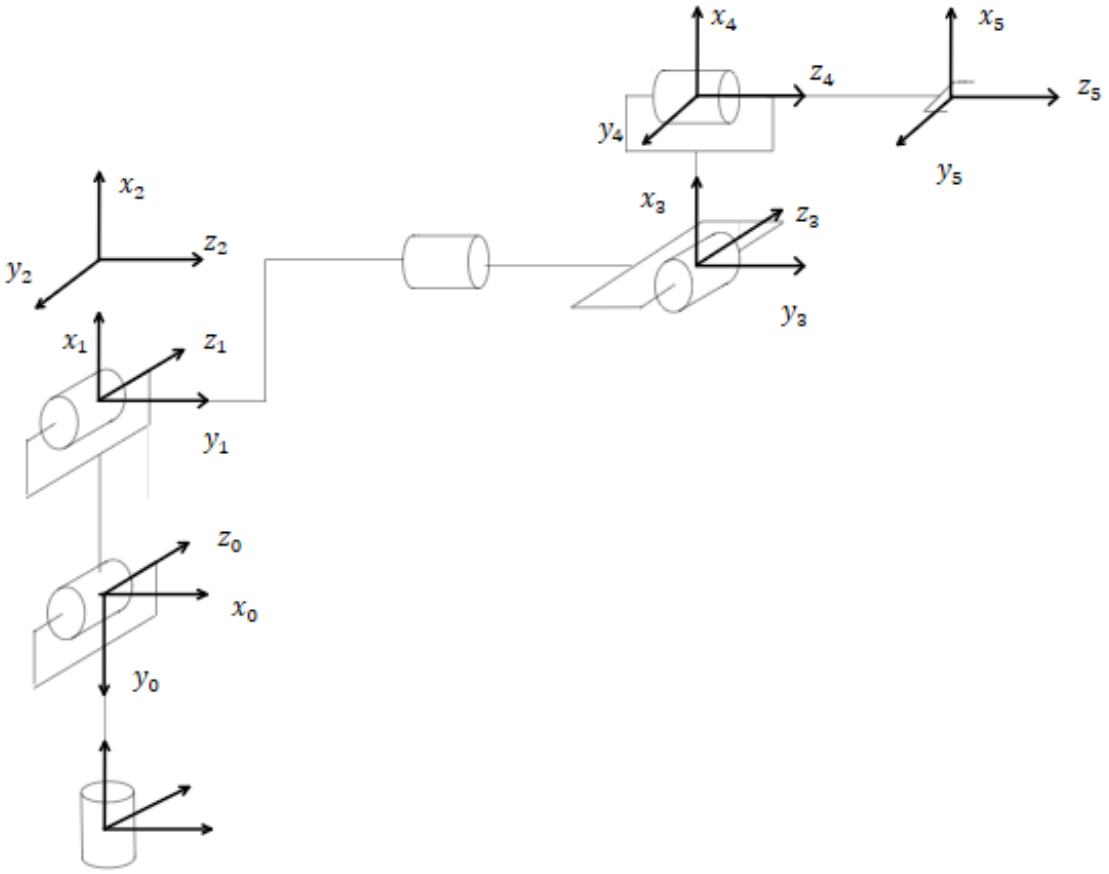


Figure 2.4: Denavit-Hartenberg reference frames

reported in Table 2.1 The Homogeneous Transformation Matrix can be retrieved from the

Table 2.1: DH parameters

Reference frame	θ	α	d	a
0	θ_0	-90°	0,265	0
1	$\theta_1 - 90$	0	0	0,44
2	θ_2	-90°	0	0,11
3	θ_3	90°	0,47	0
4	θ_4	-90°	0	0,08
5	θ_5	0°	0,101	0

following matrix multiplication:

$$A = A_0 \times A_1 \times A_2 \times A_3 \times A_4 \times A_5 \tag{2.1}$$

However, since the tool flange is connected to the welding tool, a suitable tool reference frame must be set. In particular, this reference frame will be translated, with respect to the flange reference frame, of 32.7 mm along axis x, and 165.54 mm along axis z, while its orientation remains equal to the one of the flange reference frame. So, this further matrix multiplication must be added. The orientation values extracted from this representation are in the rotation matrix form, so it is required a conversion in quaternion.

2.2.2. Filtering

The last preliminary task is to select suitable filtering techniques, to smooth the data about the recorded trajectory of the demonstration. In our algorithm development, the data of the recorded trajectory will be filtered with the Centered Moving Average (CMA) filter. CMA filter is relative to the family of Moving Average filters, and produces an output y averaging a number of points M from the input signal x: [39]

$$y = \frac{1}{M} \sum_{j=0}^{M-1} x[i+j] \tag{2.2}$$

In particular, in CMA, the group of points from the input will be symmetrical around the output point, so the new form of the equation is:

$$y = \frac{1}{M} \sum_{j=\frac{-(M-1)}{2}}^{\frac{M-1}{2}} x[i+j] \tag{2.3}$$

The number of points considered in the interval, M, is also called window size, and since the CMA will place the output value at the center of the window size, it must be an odd number.
Moving Average filters are very easy to understand, and widely used thanks to their simplicity, intuitive code implementation, optimal noise reduction, and ability to keep a sharp step response. Even if their performance in the time domain is advantageous, their behavior in the frequency domain is not optimal.[39]

3 | GoFa Smart Weld

3.1. Training Stage

The following Chapter will describe the core of the method developed in the Thesis work, explaining all the necessary steps required to correctly parameterize the trajectory. The process requires three different stages. In the first stage, called the training stage, the method manages the communication with the robot during the demonstration of the task. This stage will manage the activation and de-activation of lead through from the ASI button, the collection of joint position, and speed trajectory from the demonstration. In particular, once the connection with the robot has been established, the Graphical User Interface will show an introductory message. When the upper button of the ASI is pressed, the lead-through is enabled. Anyway, the recording of the trajectory is still not in progress, so the GUI will be the one figured in Figure 3.1. Then, when the tool of the

Figure 3.1: User Interface before the training stage

robot is positioned on the first point of the workpiece to be welded, the lower button of the ASI is pressed, and the recording of the joint position and speed trajectories starts. The GUI will have the aspect in Figure 3.2. Once the demonstration is over, we can stop recording by just pressing again the lower button of the ASI. Finally, the lead through must be deactivated, by pressing the upper button of the ASI a second time, and the robot will move to the home configuration. It's worth mentioning that the robot can go in two different home positions, relative to the verse of the demonstration. At the beginning of the training stage, the robot will show a configuration, with the sixth joint, equal to -180°. If the demonstration is performed by the operator clockwise, the reach of the home position after the training must preserve this configuration, to avoid the sixth joint going out of range. So, the first home configuration will have, for the sixth joint, a value equal to -180 degrees. For the same reason, if the demonstration is performed counter-clockwise, the first value after the start of the training, for joint six, will be equal to 180°, so the home position in this case will have a sixth joint placed at 180 degrees. The main reason for this choice is that the workpiece, after the training stage, will be welded in the same verse as the demonstration. The other five joints instead will have the same values for both home configurations. The operator should decide the verse of the demonstration, reorienting the sixth joint accordingly, before starting the recording of the trajectory.

Figure 3.2: User Interface during the training stage

3.2. Processing stage

The second stage of the procedure is the processing stage, where the analysis of the recorded trajectory is performed; this stage represents the core part of the algorithm. In particular, this stage is focused on the division of the filtered trajectory into segments, each one related to one particular movement for the execution of the trajectory, and the association of every segment with its own skill parameters. Finally, a list that contains all the moves with their parameters in the correct order will be generated. Each move command will be sent to the robot for the execution of the trajectory. When the processing stage terminates, the GUI will show three kinds of plots. First, the plot of the recorded trajectory versus the filtered one, then the plot of the parameterized trajectory, and finally the plot of the parameterization of multiple workpieces present on the scene. The move commands are generated by the robot, through the suitable ABB RAPID functions. In particular, we will have three kinds of commands, called respectively 'MoveL', used to move the tool center point (TCP) of the robot linearly to a given destination, 'MoveJ', used in this task to reorient the TCP at corners, and 'MoveC', used to move TCP circularly to a given target.
The move list is generated so that the correct identification of the move command must be codified with a numeric label that the robot is able to interpret, in order to associate the correct move command. In particular, the MoveL command will be associated with integer number 1, the MoveC command will be associated with integer number 2 and the MoveJ command will be associated with integer number 3.

3.2.1. Trajectory discretization

The first step of the Processing stage is trajectory discretization. The initial dataset is composed of the collection of the position of the end effector during the demonstration, filtered by a Centered Moving Average filter, and defined in the robot reference frame. These data have been retrieved in the Preprocessing stage. This dataset will form an ordered sequence of points in the cartesian space and so the overall path will be discretized into multiple oriented segments, each one delimited by an initial and a final point. The method is able to evaluate how the position of the considered point is varying with respect to the next one, for the entire dataset. Figure 3.3 depict the overall situation. This evaluation can be performed by quantifying the angle γ_i, defined through the oriented segment s_i that starts from the considered point p_i and reaches the next one, p_{i+1}. The angle γ_i will define the orientation of the considered segment with respect to the x-axis of the robot reference frame, and it is positive for counter-clockwise rotations, and

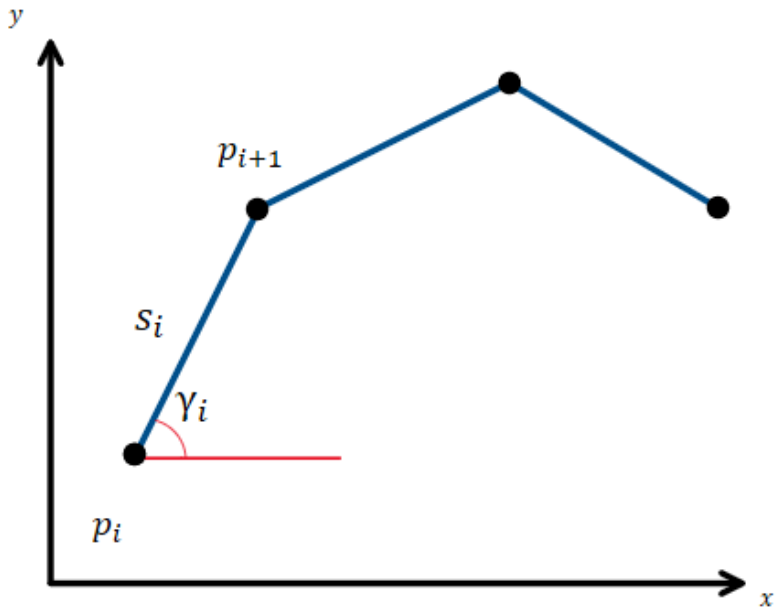


Figure 3.3: Representation of angle γ_i

negative for clockwise rotations. This procedure will allow us to collect the vector γ for the complete dataset. It is worth underlying that the evaluation of γ for each point will be referred to the next one for the entire dataset, except for the last point (forward difference approximation). In the latter case, γ will be referred to the previous point of the dataset (backward difference approximation).
From the practical point of view, γ can be calculated through the slope of the oriented segment in the considered point of dataset x:

$$\gamma = atan(f'(x)) \tag{3.1}$$

The derivative part of the formula can be approximated numerically via the finite difference method, a numerical method very useful to solve ordinary differential equations (ODE) that approximate the differential operator in the equation with the differential quotients, that is the difference of the value of the function in two points.
For the sake of completeness, it's possible to derive the difference quotient through Tylor expansion, expanding the value of the function f in the point of x-coordinate $(x+h)$ and truncating the series at first order:

$$f(x+h) = f(x) + hf'(x) + \mathcal{O}(h^2) \tag{3.2}$$

The approximated value of the first derivative of the function f in point x will be:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \tag{3.3}$$

This is the formulation of the so-called forward difference approximation. The formulation of the backward difference approximation will be:

$$f'(x) \approx \frac{f(x) - f(x-h)}{h} \tag{3.4}$$

Graphically we can refer to the situation depicted in Figure 3.4: The evaluation of the

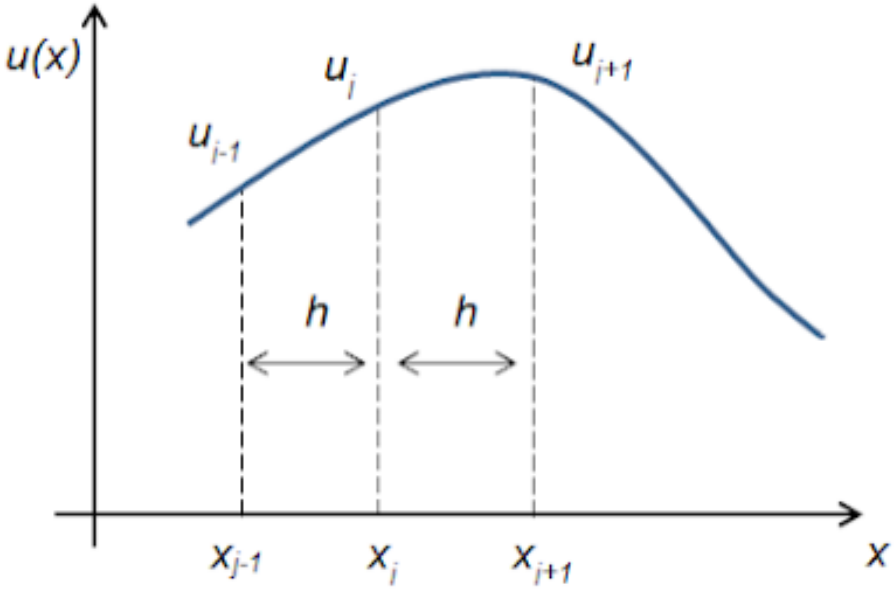


Figure 3.4: Structure of the forward and backward approximation

equation 3.1 can be performed through the 3.3 for the first n-1 points of the dataset, and through the equation 3.4 for the last point of the dataset. In the first case, for dataset point i of coordinates (x_i, y_i), γ will be evaluated as:

$$\gamma = atan\frac{y_{i+1} - y_i}{x_{i+1} - x_i} \tag{3.5}$$

In the latter case, for dataset point i of coordinates (x_i, y_i) , γ will be evaluated as:

$$\gamma = atan\frac{y_i - y_{i-1}}{x_i - x_{i-1}} \tag{3.6}$$

The finite differences method is a numerical method, and so it introduces an error with respect to the exact solution of the ODE. In particular, the truncation of the Taylor series introduced for evaluating the difference operator will quantify this error with respect to the original differential operator of the ODE. This error is called discretization error, or truncation error. Furthermore, the length of the vector γ is exactly the same as the dataset. The discretization of the trajectory and move recognition described in the next chapters will be focused entirely on the vector γ, especially in the evaluation of the position of the suitable points of welding for each move command. Each index of the vector γ is related to a move index, related to a specific segment of the trajectory. Trace back the value of the position in the dataset will be straightforward, thanks to the index-wise correspondence between the vector γ and the dataset itself.

3.2.2. MoveL detection

The first approach to the correct parameterization of the demonstrated task is the evaluation of the so-called MoveL command, which is the command used to move the tool center point (TCP) of the robot linearly, to a given target. This command is suitable when the geometry of a segment to be welded is linear. The correct structure of the MoveL command required by the robot is also needed. This command will require the target point of the MoveL command, with a robtarget structure, composed of:

- the position of the target point, i.e., the coordinates of the point in the robot reference frame;
- the orientation of the tool in the target point, expressed in quaternions;
- the speed and the TCP reference frame used for the commands.

For the current method, we are interested just in the position of the target point. The management of quaternion orientation will be managed separately.
Starting from the complete dataset, this method will evaluate only the portion of the filtered dataset related to a MoveL command, labeling the complementary part as “not MoveL”. The move type named “not MoveL” will be evaluated in the further steps of the algorithm.
To understand how “MoveL” and “not MoveL” classification works, we can refer to Figure 3.5. In particular, we will extract the correct index of the vector γ, which, thanks to its index-wise correspondence with the dataset, will output the position of the last target point of the MoveL command. We can refer to points A, C, and E in Figure 3.5. For the “not MoveL” commands, the index of the vector γ will refer to the position of the point

Figure 3.5: "MoveL" and "not MoveL" classification

where the "not MoveL" command will end. To this purpose, we can refer to points B and D in Figure 3.5. The orientation coupled with these points will be managed later in the Chapter.

From the implementation point of view, we can refer to the pseudo-code 3.1. We can divide the approach into two parts. The first one is described in lines 4-14 of pseudo-code 3.1, and it is related to the detection of the MoveL commands. In particular, the method analyzes how much the values of vector γ will differ with respect to a reference one, for all the elements of γ. The reference value, in line 3, will be initialized as the first value of the vector γ, and it is called n. To quantify this difference the parameter ϵ_{out} must be introduced, which defines the threshold value for the MoveL classification in line 6. In particular, every time two consecutive values of the vector γ, with respect to the reference one, are below this threshold value, they can be classified as part of a linear segment of the trajectory. The last point recognized in the linear trajectory will be the target point of the MoveL. Anyway, the simple definition of this parameter is not sufficient to fully understand if we are in a MoveL regime, because the teacher can perform the demonstration in an irregular way, resulting in vibrations of the end effector and thus, an irregular trajectory. Thus, is not so easy to discriminate between a local variation of the vector γ, and a variation of vector γ caused by a proper change of shape of the workpiece of the demonstration.

So the introduction of the parameter $min_{moveLength}$ is required, which represents the minimum number of indexes, with respect to the index of the reference value, called n_j, needed to classify the portion of the trajectory as MoveL. So, if the condition on $min_{moveLength}$ in line 9 is satisfied, the algorithm detects the segment as MoveL. As soon as the condition on ϵ_{out} is no longer satisfied, the current index of the vector γ exits the part of the detection of the MoveL. The reference value will be updated with the current value of the vector γ, in line 10, and that index is saved together with an integer number equal to 1. Then, the current index of the vector γ enters a second part of the code, described by lines 15-25 of the pseudocode 3.1. Every index of vector γ in this part will belong to a

Algorithm 3.1 MoveL detection

$\gamma_{len} \leftarrow$ length of γ array
$MoveL = True$
$n = \gamma_0$
for i `in` γ_{len} **do**
    if $MoveL = True$ **then**
        if $(\gamma_{i+1} - n) < \epsilon_{out} \wedge (\gamma_{i+1} - n) < \epsilon_{out}$ **then**
            continue
        else
            if $(i - n_j) > min_{moveLlength}$ **then**
                $n = \gamma_i$
                $MoveL = False$
            end if
        end if
    end if
    if $MoveL = False$ **then**
        if $(\gamma_{i+1} - n) < \epsilon_{in} \wedge (\gamma_{i+1} - n) < \epsilon_{in}$ **then**
            continue
        else
            if $(i - n_j) > min_{moveLlength}$ **then**
                $n = \gamma_i$
                $MoveL = True$
            end if
        end if
    end if
end for

"not MoveL" classification. Thus, the portion of the trajectory until the current γ index will be represented as "notMoveL", identified by the integer number 2. To understand if the current index of γ can be classified again as MoveL, the parameter ϵ_{in} must be introduced. All the elements considered, with respect to a reference value, must be lower than this parameter to be classified again as MoveL, as shown in line 16.
Here we still need that the part of the trajectory classified as linear will be satisfied for long enough, in particular for a number of indexes of the vector γ equal to the $min_{moveLlength}$ parameter, as shown in line 19. When the ϵ_{in} condition is satisfied, the current index of the vector γ exits the part of the detection of the "not MoveL", and can be classified again as MoveL. The reference value will be updated with the first value of the vector γ for which the ϵ_{in} condition holds, and an integer number in correspondence of that index is placed equal to 2.
The final output of the method is composed of two vectors: the first one, called segment indexes, is the collection of the indexes of the reference points defined above, and the second one, called recognized move segments, contains the corresponding values of the integer numbers.
Concluding, it's possible to define some final considerations. First of all, the nature of the vector γ can be discontinuous in -π. So, the method must be robust when the current values of γ reach or overcome that value, because the overall behavior of γ can present multiple discontinuities. To avoid problems, it's enough to adjust the current values of γ by -2π, whenever they reach this discontinuity. In addition, the comparison on ϵ_{in} and ϵ_{out} parameters must be done considering the absolute value of the difference, to avoid problems of sign in the comparison with the aforementioned parameters.

3.2.3. MoveJ detection

This section will be devoted to detecting which type of move is performed in the "not MoveL" segments. In particular, the method will spot the presence of MoveJ commands. For our purpose, the MoveJ command is requested whenever the tool is re-oriented, as a result of a change of shape of the workpiece.
The structure of the MoveJ command is the same of the MoveL. For the current method, we are interested just in the position of the target point. The quaternion orientation will be managed separately. As a reference, it's possible to refer to Figure 3.6. In particular, the method is able to discriminate if the "not MoveL" segment is entirely a MoveJ segment, is composed of a mix of MoveJ and MoveC segments, or if it's entirely a single/multiple MoveC. The main idea behind this method is the fact that a re-orientation of the tool,

Figure 3.6: "MoveJ" are represented in red, and "MoveC" are represented in green

consequent to a variation in the shape of the workpiece, will result in a group of consequent dataset points all close to each other. The method is described in the pseudo-code 3.2. Practically, as shown in line 5 of the pseudo-code 3.2, the consecutive physical x and

Algorithm 3.2 MoveJ detection

1: $p_{len} \leftarrow$ length of dataset array
2: $x_i \leftarrow x$ coordinate of position of point p_i
3: $y_i \leftarrow y$ coordinate of position of point p_i
4: **for** i `in` p_{len} **do**
5: **if** $(x_{i+1} - x_i) < neighb_{radius} \wedge (y_{i+1} - y_i) < neighb_{radius}$ **then**
6: $n_j = i$
7: **end if**
8: **end for**

y coordinates of the position p of the end effector must be enclosed in a neighbor of a certain radius, expressed by the $neighb_{radius}$ parameter, in order to be classified as MoveJ. The index of the vector p to which the condition in line 5 holds, will be the index of the target position for the MoveJ command, assigned in line 6. Target positions of MoveJ are represented in Figure 3.6 by points B, D, and F. All the other evaluated points of the original "notMoveL" segment, that do not belong to a MoveJ, in this step, are classified in the MoveC segment. They will be represented by the last point of the MoveC, that in the recognized move segments array corresponds to the integer number 2. We can refer to the point C in Figure 3.6.

However, the results found in this stage of the method still need to be refined. In fact, the tuning of the $neighb_{radius}$ does not guarantee that all the points will belong exactly to their correct segment in just one step. If the parameter is set too high, some points can be classified in the MoveJ, instead of a MoveC, and some important segments of the trajectory of the MoveCs will be lost. Instead, if the parameter is too low, the points that should belong to the MoveJ, are instead classified as MoveC. So a minimum length for the

MoveC is set through the parameter ϵ_{movec}, which defines the minimum length requested for the MoveC segment. All MoveC whose length is lower than ϵ_{movec} will be merged with the adjacent MoveJ segment, if present.
The method will output the updated values of the segment indexes array and the recognized move segments array, that now contains also, respectively, the indexes of the vector γ for the MoveJ segments, and the correspondent integer number equal to 3.
Concluding, for sure, at the end of this step, all the segments labeled with an integer number equal to 2 are MoveCs, but one or multiple MoveCs can be present, so further analysis is required. The next section will address this problem.

3.2.4. MoveC detection

The algorithm is capable to define all the segments that contain just MoveCs, that at this point will be characterized by an integer number equal to 2. Unfortunately, we are still not able to know either if every aforementioned segment contains just one or multiple MoveC, and the value of their concavity. In particular, the concavity, and its variation, will be very important to understand how many MoveC are taking place. First of all, let's define the concept of concavity. The concavity of a function f can be defined by the sign of its second derivative. If, for all x in the interval I:

$$f'' > 0 \tag{3.7}$$

then $f(x)$ is concave up on I. Instead, if:

$$f'' < 0 \tag{3.8}$$

then $f(x)$ is concave down on I.
Once defined the concavity, two or more consecutive MoveC will be spotted if there is a variation of the sign of the concavity in the MoveC segment. Figure 3.7 shows two consecutive MoveC with opposite concavity on the right side of the workpiece. For the MoveC detection, we can refer to the pseudo-code 3.3. The method requires a parameter to distinguish between the local change of concavity due to the irregularities and vibrations of the end effector during the demonstration, and the effective change of concavity due to a proper change of shape of the workpiece. So, the parameter $min_{concavitychange}$, defines the number of indexes for which the sign of the concavity must be preserved.
From the code implementation point of view, for the current MoveC segment, a concavity vector must be defined, evaluating the sign of the difference between two consecutive

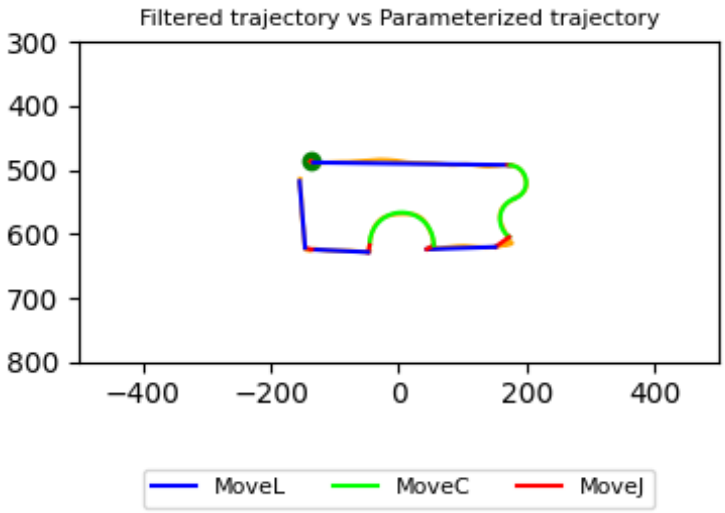


Figure 3.7: Two consecutive MoveCs

values of the vector γ.

The current concavity vector is then divided into two different parts. The first one, defined in line 3, is devoted to evaluating the current value of the concavity for the MoveC segment. So, in the first $min_{concavitychange}$ indexes of the concavity vector, the higher number of occurrences between -1 and 1, represented respectively as n_{-1} and n_{+1} will define the current value of the concavity, as shown in line 4. The second part of the concavity vector, from the $(min_{concavitychange} + 1)$ index to the last one, will be analyzed to spot one or multiple changes of concavity with respect to its current value, as shown in line 10. In particular, every time the value of concavity is different from the start value, and it's different from the next value of concavity in the concavity array, the change of concavity is verified, as shown in line 11. So, the index of the vector γ related to this change of concavity, n_j, is inserted in the segment indexes array, with the correspondent integer number equal to 2. We can refer to line 12 of the pseudo-code. Thus, if the presence of multiple consecutive MoveC is verified, the original MoveC segment is broken down into sub-segments, one for each MoveC. If there is not change of concavity, the considered segment will be composed of just a single MoveC. The method will output the updated values of the segment indexes array and the recognized move segments array, that now contains also, respectively, the indexes of the vector γ for all the effective MoveC segments, and the correspondent integer number equal to 2.

Concluding, it's worth mentioning two important facts for the further development of the algorithm. The structure of the MoveC command is slightly different from the MoveL and MoveJ. In particular, the MoveC command requires the circle point, or CirPoint, i.e., the position and orientation of the mid-point of the arc that defines the MoveC command,

Algorithm 3.3 MoveC detection

$c_{len} \leftarrow$ length of concavity array c
$f_c \leftarrow$ start value of concavity
for $i = 0, .., min_{concavitychange}$ in c_{len} **do**
    if $n_{-1} > n_{+1}$ **then**
        $f_c = -1$
    else
        $f_c = 1$
    end if
end for
for $i = min_{concavitychange}, .., c_{len}$ **do**
    if $c_i \neq c_{i+1} \wedge c_i \neq f_c$ **then**
        $n_j = i$
    end if
end for

with a robtarget structure; the endpoint, or ToPoint, i.e., the position and orientation of the destination point of the MoveC command, with a robtarget structure. Figure 3.8 describes the MoveC command. The method developed in this paragraph is able to detect,

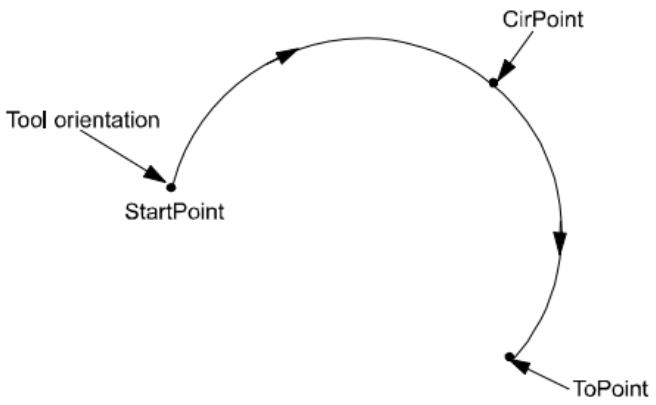


Figure 3.8: robtarget required for MoveC

for each MoveC, just the position of the endpoint of the arc, or ToPoint.
One last final remark must be highlighted on the values of concavity for each MoveC, because this term will be crucial in the definition of the correct orientation of the tool during a MoveC command. MoveC related to a decreasing slope will have a sign of concavity equal to -1, because the gamma vector will decrease in its absolute value. Instead, all the MoveC related to an increasing slope will be related to the sign of concavity equal to 1, because the gamma vector will increase in its absolute value. Anyway, the direction of the demonstration is here fundamental. Let's take for example the following situation in 3.9. If the demonstration, from the starting point in green, is executed

in a clockwise direction, respectively the first and second MoveC will be related to a decreasing slope, and a sign of concavity equal to -1, and the third MoveC will be related to an increasing slope, and a sign of concavity equal to 1. Instead, if the demonstration, from the starting point in green, is executed in a counter clock-wise direction, respectively the first MoveC will be related to a decreasing slope, and a sign of concavity equal to -1, and the second and third MoveC will be related to an increasing slope, and a sign of concavity equal to 1. The dependency between the sign of concavity and the direction of demonstration must be addressed in the management of the orientation of the tool during the MoveC command, which will be developed further in the Chapter.

3.2.5. Circular trajectories parameterization

So far, the algorithm is able to segment the overall filtered trajectory in the start and end point of every single move, utilizing exclusively the position of the points recorded during the demonstration. This approach is sufficient in the cases of MoveL and MoveJ, but for a MoveC command, the situation is slightly different. The command requires also the precise position of the mid-point of the circumference, which is not easily detectable along the workpiece. Furthermore, the teacher can have difficulties moving correctly the robot, obtaining a recorded dataset not reliable for defining a proper circumference. For these reasons, all the MoveC segments of the dataset analyzed up to this point, must be mapped to a proper circumference, through a circumference parameterization procedure. From the analytical point of view, for the single portion of the dataset that refers to a

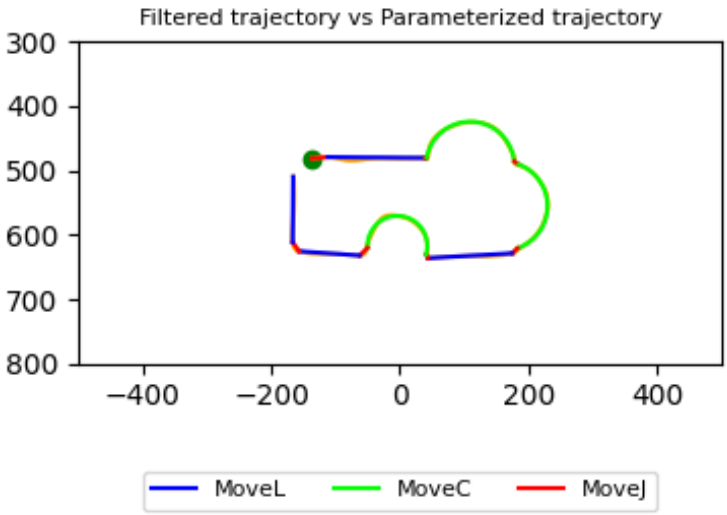


Figure 3.9: Workpiece for the concavity evaluation

MoveC, we can find the circumference that minimizes the least squares error, through Ordinary Least Squares method. However, this optimization problem is a non-linear least squares problem, that must be adequately linearized, rearranging opportunely the circle equation. The overall procedure will output the coordinates of the center of the circle, and its radius.
Starting from the circle equation for the generic point of the MoveC segment i:

$$r^2 = (x_i - x_c)^2 + (y_i - y_c)^2 \tag{3.9}$$

This equation can be re-arranged as follows:

$$x_i^2 + y_i^2 = ax_i + by_i + c \tag{3.10}$$

Where:

$$a = 2x_c \tag{3.11}$$

$$b = 2y_c \tag{3.12}$$

$$c = r^2 - x_c^2 - y_c^2 \tag{3.13}$$

Extending this approach to the entire considered dataset, the system takes the matrix form:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ \dots & \dots & \dots \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} x_1^2 + y_1^2 \\ \dots \\ x_n^2 + y_n^2 \end{bmatrix} \tag{3.14}$$

Now the problem can be treated as a classical OLS optimization procedure, where the values of a, b, and c can be retrieved. The coordinates of the center of the circumference, ad its radius will be:

$$x_c = \frac{a}{2} \tag{3.15}$$

$$y_c = \frac{b}{2} \tag{3.16}$$

$$r = \frac{\sqrt{4c + a^2 + b^2}}{2} \tag{3.17}$$

The procedure described above is performed for all the MoveC segments individuated by the algorithm. Given the starting and ending points of the arc, and its center, it's possible to formulate the parameterization of the arc between p and q, through the parameters s

and t:

$$s = 2atan\frac{p_y - c_y}{p_x - c_x + r} \tag{3.18}$$

$$t = 2atan\frac{q_y - c_y}{q_x - c_x + r} \tag{3.19}$$

The coordinates of the arc mid-point can be retrieved:

$$x_m = x_c + r\cos(\frac{s+t}{2}) \tag{3.20}$$

$$y_m = y_c + r\sin(\frac{s+t}{2}) \tag{3.21}$$

One major issue of this procedure is the fact that, for a given starting point A and an ending point B of the arc, two possible arcs exist, that is the one that goes from A to B, and the one that goes from B to A. In particular, if the parameterization will output:

$$s < t \tag{3.22}$$

The arc will be counterclockwise from A to B; instead, in the opposite case:

$$s > t \tag{3.23}$$

The arc will be clockwise from A to B. Picking the wrong arc will be equivalent to a change in the concavity for the considered MoveC, because the mid-point will be 180 degrees shifted with respect to the desired one. To make the algorithm robust, the $\epsilon_{wrongarc}$ parameter is introduced, that is the distance between the candidate arc mid-point, and its closest recorded point during the demonstration. If the threshold introduced by $\epsilon_{wrongarc}$ is overcome, the algorithm will pick automatically the mid-point of the complementary arc with respect to the one selected. The midpoint of the correct circumference is found by adding 2π to s or t if the arc is labeled as counterclockwise or clockwise respectively.
Once the coordinates of the arc mid-point are evaluated, they will be used further in the algorithm for the MoveC command generation. It's important to underline that the arc starting, ending, and mid-point are "generated" by the algorithm, and different from the ones present in the original dataset of the demonstration, nor in the filtered one.
In conclusion, this method will output, for each MoveC, the starting, ending, and mid-point of the arc, and the respective parameters about the center and the radius of the fitted circumference.

3.2.6. Quaternion management

The algorithm will evaluate in the first place the suitable positions for the parameterization of the workpiece, and separately the respective orientations, in quaternions. The management of the orientation will be different with respect to the specific move command. For MoveL and MoveJ commands, the orientation of the tool recorded during the demonstration is retrieved. This approach introduces a small approximation error because actually the retrieved orientations are referred to the original data points of the demonstration, while the matched positions are referred to the demonstrated trajectory filtered by a Centered Moving Average filter. Once these orientations are extracted, a further rotation of the respective tool reference frame will be necessary, in order to align it in the direction of the z axes of the base (or robot) reference frame. This alignment is needed to accomplish the requirement of the zero-travel angle goal.
For the MoveC command instead, the orientation of the tool in the start, end, and midpoint of the fitted arc is required. In particular, the tool must be oriented perpendicular to the circumference, in direction of its center of circumference. These orientation values will be generated from scratch, because the likelihood that the demonstration of the teacher will have the same orientation is very low since usually performing a demonstration along a circular path is very difficult, especially for novice users. The generation of the orientation for the MoveC command follows this general idea: the final orientation, i.e., the orientation of the tool reference frame in the considered point alongside the arc, is composed of subsequent axis-angle rotations of the robot reference frame. To this purpose, at first, the algorithm must evaluate the suitable axes and angles that compose the final orientation of the tool, and then express this orientation in a suitable quaternion value, that will be inserted in the MoveC command. The axis-angle representation is a common method useful to represent an orientation in a three-dimensional Euclidean space. In particular, a given orientation is represented by a rotation of the considered reference frame along a specific axis, with a magnitude of a particular angle about the axis. The advantage of using an axis-angle representation lies in the intuitiveness of its mathematical formulation, and its direct conversion to the quaternion notation, without any form of ambiguity.
The orientation of the tool reference frame in a MoveC is composed of two subsequent axis-angle rotations of the robot reference frame. The first one goes along its x-axis, 90 degrees in clockwise direction, obtaining as a final result the intermediate reference frame in Figure 3.10. The second one is directed along the z-axis of the latter reference frame, by the angle that orients the x-axis of the tool reference frame with the center of the circumference. This last orientation is depicted in the last reference frame in Figure 3.10.

The algorithm will quantify this angle for the start, mid and end points of the arc of the

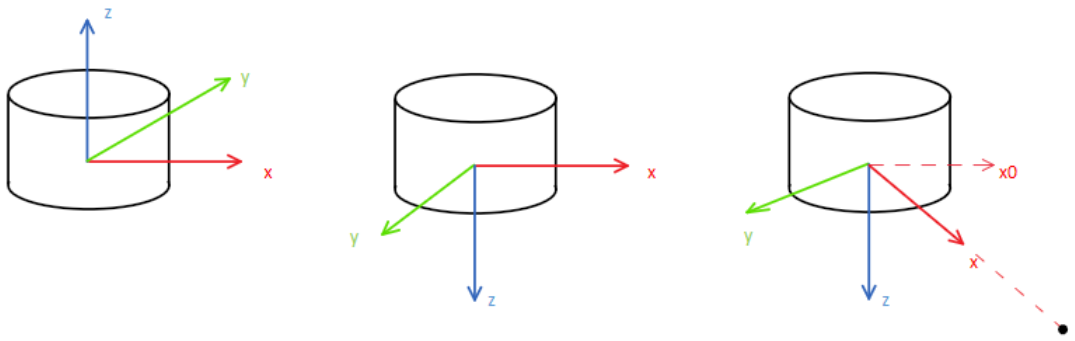


Figure 3.10: First rotation around x by 90°, second one around z by x0-x angle

considered MoveC.
Once the orientations, in axis-angle representation (through angle θ and unit versors of the axis), are found, each one will be converted in the correspondent quaternion notation:

$$q_0 = \cos\frac{\theta}{2} \tag{3.24}$$

$$q_1 = \hat{x}\sin\frac{\theta}{2} \tag{3.25}$$

$$q_2 = \hat{y}\sin\frac{\theta}{2} \tag{3.26}$$

$$q_3 = \hat{z}\sin\frac{\theta}{2} \tag{3.27}$$

where the conversion equation must maintain the correct order of the four elements of the quaternion, with the first one that will be scalar, and the other three that will be the vectorial part. The desired orientation of the reference frame can be found by multiplying the values of the quaternion for every single rotation, from the last one to the first one. The orientation of the tool reference frame in MoveL or MoveJ, instead, is evaluated starting from the orientation during the demonstration, already in quaternion form, depicted in the center of Figure 3.11. To align this reference frame with the z direction of the robot reference frame, depicted on the left of Figure 3.11, one additional rotation is enough. The axis of rotation is the axis perpendicular to both the z-axis of the tool reference frame of the demonstration, and the z-axis, in negative direction, of the robot reference frame. The final orientation of the tool reference frame is depicted on the right of Figure 3.11. The angle instead is the one that the former z axis form with the latter one. The

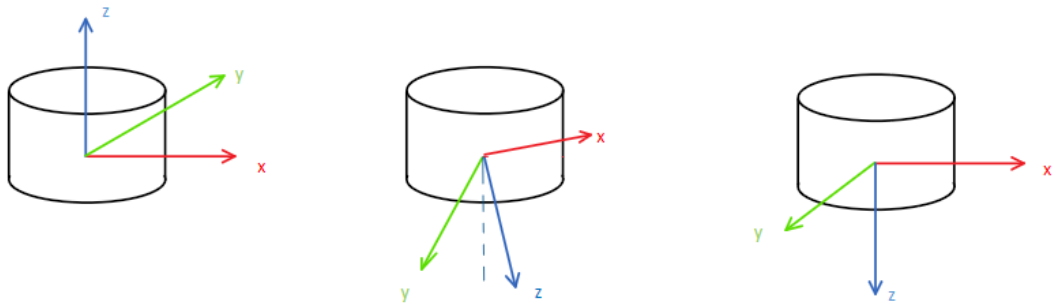


Figure 3.11: Alignment procedure of the tool reference frame

procedure of identification of the final desired orientation remains the same. First, the axis-angle rotation is converted to quaternion notation; then, the desired orientation is found by multiplying the obtained quaternion q with the one of the demonstration (q_{dem}), in this specific order:

$$q_{fin} = q q_{dem} \tag{3.28}$$

However, the procedure of evaluation of the orientation during the MoveC command, must keep track of another important variable, the concavity. Even if the tool is always in the direction of the center of the circumference, the sign of its direction depends on the nature of concavity of that specific MoveC, and furthermore, the registered value of concavity depends on another variable, which is the verse of the demonstration. In fact, the algorithm has been designed in order to parameterize the entire closed geometry of the workpiece, assuming the demonstration will follow an almost closed path. This aspect is fundamental in order to retrieve the exact direction of the demonstration, and consequently to assign correctly the orientation of the tool along the arc of the MoveC command. In particular, the start and end values of the sixth joint recorded during the demonstration will be examined. If the former is greater than the latter, the demonstration is performed in counter-clockwise direction. Instead, if the start value of the sixth joint is lower than its end value, the demonstration is executed in clockwise direction. So, the algorithm has to distinguish between the following cases. If the demonstration is executed in clockwise direction, the MoveCs related to a decreasing slope, and a sign of concavity equal to -1, will have the tool perpendicular to circumference, directed in its center. The MoveCs related to an increasing slope, and a sign of concavity equal to 1, instead, will have the tool perpendicular to circumference, but rotated by 180° about the z-axis of the tool reference frame. Instead, if the demonstration is performed in counter-clockwise direction, the situation is the opposite. The MoveCs related to a decreasing slope, and a

sign of concavity equal to -1, will have the tool perpendicular to circumference, but rotated by 180° about the z-axis of the tool reference frame. The MoveCs related to an increasing slope, and a sign of concavity equal to 1, instead, will have the tool perpendicular to circumference, directed to its center. As a result, demonstrations of just a portion of the geometry of workpieces, will not guarantee that the verse of the demonstration is assigned correctly, and thus the orientation of the tool reference frame can be directed in the opposite direction to the one desired during the execution of a MoveC.
For the MoveC, this method will output the cartesian positions of the point along the arc, and its tool orientation, expressed in quaternions, for the start, end, and mid-point of each MoveC. For MoveJ and MoveL, the method will output the values of the quaternion of the demonstration, aligned properly along the negative z direction of the robot reference frame.

3.2.7. Command list generation

The previous methods have shown how all the parameters about positions, and orientations have been analyzed and collected. The velocity instead is calculated through a mean calculation on the overall array that contains the end effector velocities, for all the points of the demonstration. The resulting target velocity will be input into the move list.
The next task, in this sense, is the organization of these parameters in a specific sequence, that will be sent to the robot in the execution stage. In addition, the value required to let the robot understand at which command associate the given parameters will be represented by an integer number, where respectively 1/2/3 stands for MoveL/MoveC/MoveJ. The structure of the parameters of the move command is the same for MoveL and MoveJ commands, and is the following:

- integer number required for the Move command
- target value for the velocity of the end effector
- x-axis coordinate of the target point in the robot reference frame
- y-axis coordinate of the target point in the robot reference frame
- z-axis coordinate of the target point in the robot reference frame
- orientation of the end effector in the target point, in quaternions

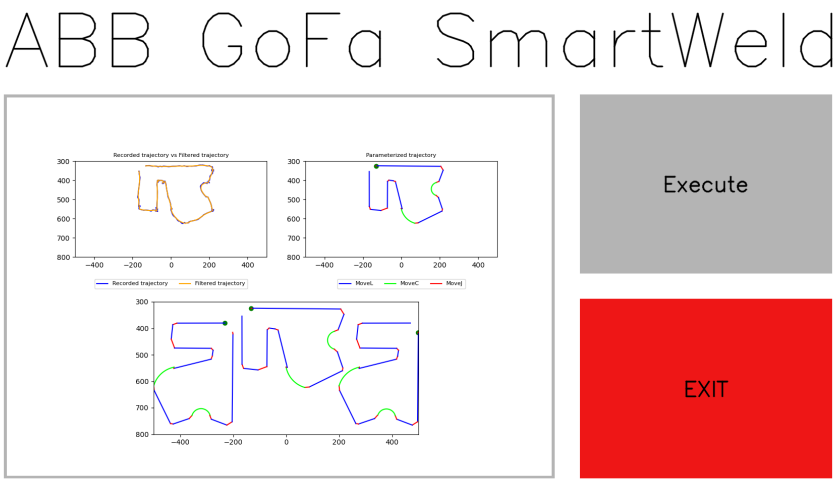
The parameter of the z coordinate will be composed of two terms. The first one is the value of the z coordinate, in the robot reference frame, of the first point recorded during the demonstration. The first point is the most reliable of the entire trajectory in terms

of z position, due to the precise positioning of the end effector by the teacher when the demonstration starts. The second term will be an offset along the z direction of the robot reference frame, for the engage and disengage movement done by the robot to approach the workpiece. The structure of the MoveC command is very similar, with one fundamental difference, which is the presence of the mid-point of the arc. So the MoveC command will present in addition the x-axis coordinate and y-axis coordinate of the midpoint, the z-axis coordinate, and the orientation of the end effector, in quaternions, in the midpoint. Some important peculiarities must be underlined. As easy to understand, the z axes coordinate and the target velocity will be the same for each command. Furthermore, in the first place, the MoveC uses only the parameters about the end and mid-point of the arc, neglecting the ones of the starting point, for which position and tool orientation have been evaluated properly. The parameters about the start point of the arc will be used only in one particular case, that is when a MoveL, or a MoveJ, is directly followed by a MoveC. In this case, instead of using the target point evaluated in MoveL (or MoveJ, respectively), we will use the start point of the arc of MoveC, with its proper tool orientation. The move list defined up to this point is related, obviously, to the recording phase of the training stage. As a consequence, the movements of the robot when it's approaching the workpiece from the home position, and when it is leaving the workpiece to come back to home position, must be defined and inserted in the move list. The approach movements are constituted by a MoveJ followed by a MoveL command. The first one goes from the home configuration to the initial point of the filtered trajectory, with a z coordinate constituted by the offset along the z-axis of the robot reference frame. The orientation of the tool will be the one retrieved from the demonstration, modified by the alignment procedure. The MoveL command will follow the same components of the MoveJ, but the z coordinate will not have the offset term. The disengage movement will be specular with respect to the approach, moving the robot end effector from the last point of the demonstration, along the z-axis of the robot reference frame of the same offset of the approach movement, but in opposite direction.
The method will output the move list that contains the parameters for the commands, needed to execute the parameterized welding trajectory.

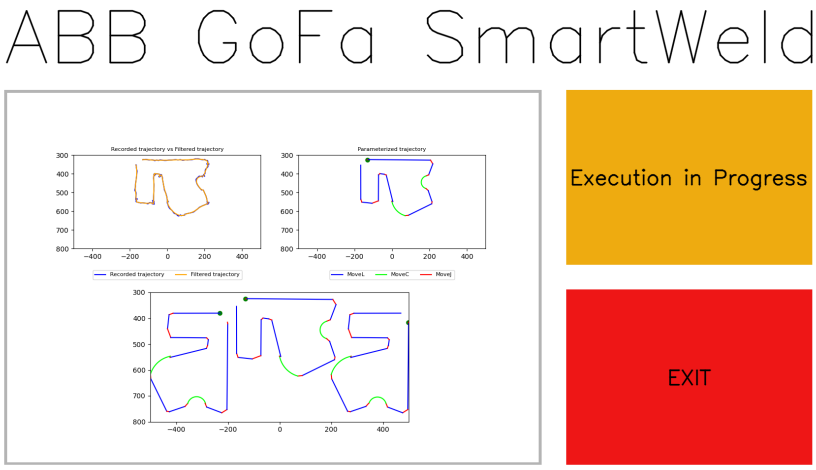
3.3. Execution Stage

After the processing stage, the GUI is shown in Figure 3.12(a), where the parameterized trajectory is displayed. The input of the operator is needed, in order to start the execution of the welding task that has been parameterized, by simply pressing the execute button.

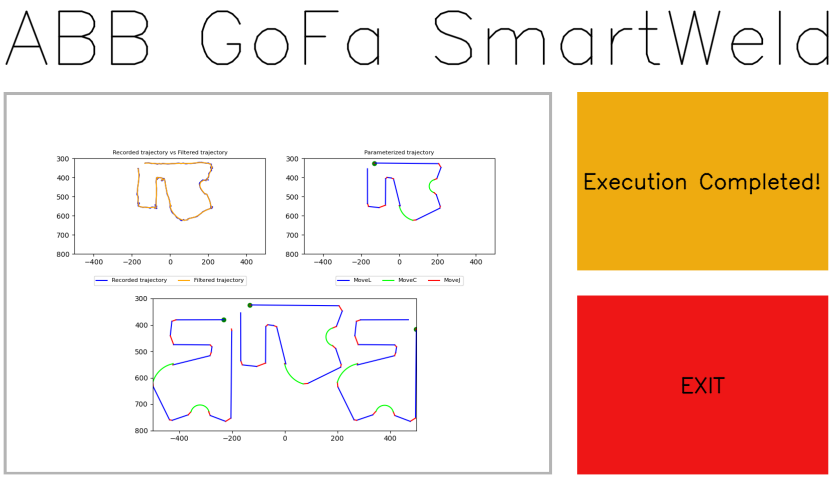
The data about the move command list will be sent to the robot. The GUI will display the status of the system either when the execution is in progress, or when it is terminated, respectively in Figures 3.12(b) and 3.12(c). As a last point, the robot will come back to the home configuration, and the execution stage terminates by pressing the exit button on the GUI.
Concluding, the Chapter has described deeply the overall procedure of parameterization and execution of the trajectory, but, for now, it's related just to the execution of the welding task on the demonstrated trajectory. Further extensions of this procedure on more complex scenarios will be discussed in the following Chapter.



((a)) GUI after the processing stage



((b)) GUI during the execution stage



((c)) GUI after the execution stage

Figure 3.12: Graphical User Interfaces

4 | Extension to multiple part welding

4.1. Multiple welding method

The presented method is able to parameterize the trajectory of the demonstration, and to execute the welding, just on the workpiece of the demonstration. So, the skill parameters extracted, are relative only to that specific workpiece. Anyway, the growing request for products with high variability and low volumes requires fast re-programming techniques, and the necessity to weld multiple workpieces of the same shape. Thus, is possible to extend the method of the parameterization of the trajectory to a scene where are present multiple workpieces, of the same shape as the demonstrated one, but in different orientations and positions. Since the shape of the workpieces on the scene is the same, it will be enough to extend the parameterized welding trajectory move targets in the reference frame of each workpiece. The method must be able to adapt the skill parameters extracted from the demonstration, to all the workpieces on the scene, understanding how the positions of the welding move targets will change from one workpiece to another, and how the tool will be oriented along the trajectory on the different workpieces. Instead, the management of the target velocity and z-axis coordinate will remain the same, granting an execution performed with the same velocity and distance between the tool and the workpieces, on all the workpieces of the scene. The velocity must be the same because its role is fundamental for the correct execution of the welding task. So, overall, the list of commands defined in the previous steps of the algorithm will be extended, adding the sequence of moves, one for each workpiece with the respective skill parameters. The relative position and orientation of the different workpieces, i.e., the pattern, must be provided in input to the algorithm. Furthermore, the execution stage must be performed by reproducing the welding on the workpieces in a specific order, decided a priori. So, it's possible to assign a number to each workpiece, that will represent the order in which the execution will take place.

Let's describe the roto-translation of one specific robtarget of the move command list, in

terms of the position and orientation of the robot end effector, as an example. To map the robtarget of the green point, defined for workpiece number one (i.e., the one of the demonstration), in the reference frame of workpiece number two, multiple homogeneous transformation matrices must be defined. The overall scene is depicted in Figure 4.1. The

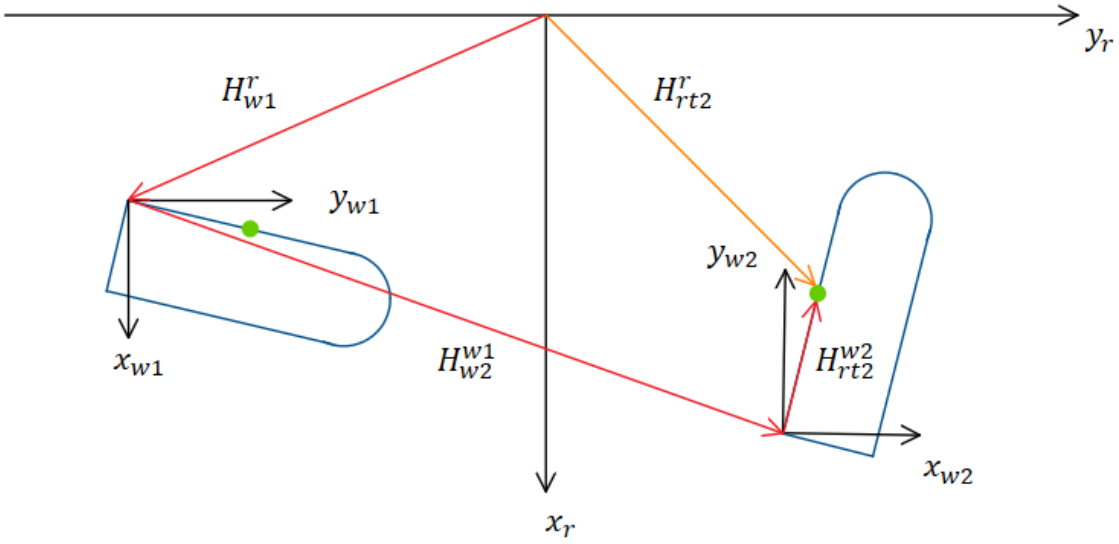


Figure 4.1: Multiple workpieces

first homogeneous transformation matrix goes from the robot reference frame, (x_r, y_r, z_r), to the first workpiece, (x_{w1}, y_{w1}, z_{w1}), and it's called H^R_{W1}. In particular, the reference frame of the first workpiece will have always the same orientation as the robot reference frame. Instead, its origin will be located at the starting point of the demonstration along the geometry of the workpiece, which is saved during the training stage. So the form of this matrix will be:

$$H^R_{W1} = \begin{bmatrix} 1 & 0 & 0 & o_x \\ 0 & 1 & 0 & o_y \\ 0 & 0 & 1 & o_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

where the rotation matrix is just the identity matrix, due to the same orientation of the two reference frames, and the position of the origin is evaluated in the robot reference frame. The second homogeneous transformation matrix required is the one that defines the rotation and translation of the workpiece number two, (x_{w2}, y_{w2}, z_{w2}), with respect to the reference frame of the workpiece number one, and it's named H^{W1}_{W2}. The reference frame of the workpiece number two in Figure 4.1, will be described by the following

homogeneous transformation matrix:

$$H_{W2}^{W1} = \left[\begin{array}{c|c} R_{w2}^{w1} & O_{w2}^{w1} \\ \hline \underline{0}^T & 1 \end{array}\right] \tag{4.2}$$

where the rotation matrix is the orientation of the reference frame of workpiece two, with respect to workpiece one, and the position of the origin is evaluated in relation to the origin of the reference frame (x_{w1}, y_{w1}, z_{w1}). The last homogeneous transformation matrix required is the one that defines the roto-translation from the reference frame of the workpiece number two (x_{w2}, y_{w2}, z_{w2}), to the robtarget for which the roto-translation must be evaluated. This matrix is called H_{RT2}^{W2}. The value of this matrix will be the same as the one that goes from the origin of the reference frame of the demonstrated workpiece (x_{w1}, y_{w1}, z_{w1}) to the correspondent robtarget that needs to be roto-translated, called H_{RT1}^{W1}. The obvious reason is that the workpieces are of the same shape, so in each object reference frame the position and orientation of the robtarget will be conserved. The situation is depicted in Figure 4.2. So the homogeneous transformation matrix H_{RT2}^{W2} will be equal to:

$$H_{RT2}^{W2} = H_{RT1}^{W1} \tag{4.3}$$

Furthermore, H_{RT1}^{W1} can be evaluated also as:

$$H_{RT1}^{R} = H_{W1}^{R} \times H_{RT1}^{W1} \tag{4.4}$$

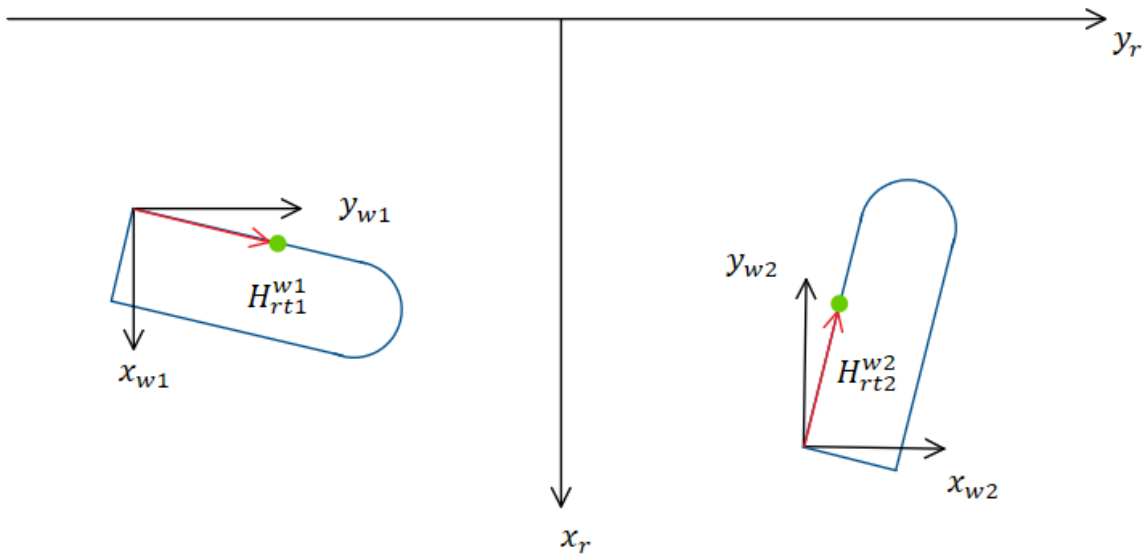


Figure 4.2: Object reference frames

$$H_{RT1}^{W1} = (H_{W1}^{R})^{-1} \times H_{RT1}^{R} \tag{4.5}$$

The reason for this additional computation is the fact that, from the list of commands retrieved by the algorithm, the position and orientation of the robtarget are referred to the robot reference frame (x_r, y_r, z_r). A matrix that manages these parameters is the homogeneous transformation matrix H_{RT1}^{R}. The situation is depicted in 4.3. So, the H_{RT1}^{R} matrix will have a rotation matrix composed by the orientation of the tool reference frame in the robtarget, with respect to the robot reference frame, and the position of the robtarget evaluated also in relation to the origin of the robot reference frame (x_r, y_r, z_r). The value of the robtarget rotated and translated in the reference frame of workpiece number two will be:

$$H_{RT2}^{R} = H_{W1}^{R} \times H_{W2}^{W1} \times H_{RT2}^{W2} \tag{4.6}$$

All the matrices represented in equation 4.6 are depicted in Figure 4.1 Once this matrix has been evaluated, it is possible to obtain the roto-translated position and orientation of the tool, the rotation matrix, and the last column of the H_{RT2}^{R}. This procedure must be repeated for all the robtarget present in the list of move commands evaluated for the demonstrated trajectory, and so the robot will be able to parameterize and execute the welding task to all the workpieces on the scene. One particular variation of this method is when the move list presents a MoveC command, that intrinsically contains two robtarget, the first one for the middle point, and the second one for the endpoint of the arc. In this case, the algorithm will evaluate the roto-translation matrix for both the robtarget

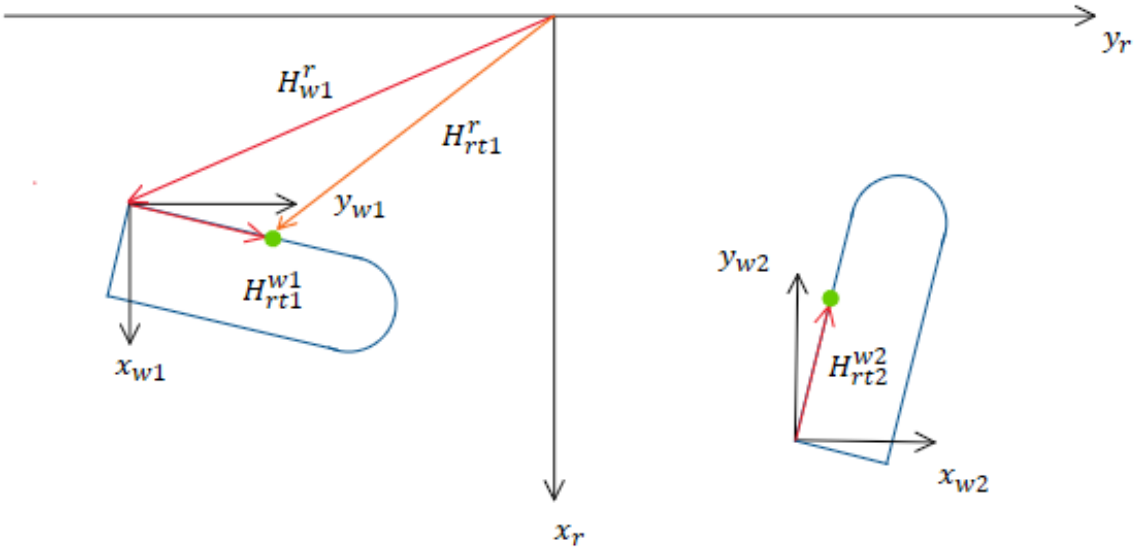


Figure 4.3: Representation of matrix H_{RT1}^{R}

of the MoveC in the same step. As stated in 3.11, and in 4.1, the orientation of the reference frame of the demonstrated workpiece, (x_{w1}, y_{w1}, z_{w1}), is the same of the robot reference frame, despite of the actual orientation of that workpiece. Thus, the parameters that the user gives in input to the algorithm, about the orientation of the other reference frames, will correspond to the actual physical workpiece orientation on the scene, only if the demonstrated workpiece is aligned with the robot reference frame.
Concluding, the method takes as input the pattern of the workpiece, i.e., the values of the orientation and position of each workpiece on the scene, and the command move list obtained from the parameterization of the trajectory of the demonstration, and will output an extended move list, that contains, in addition, the skill parameters for each additional workpiece on the scene.

4.2. Joints limit management

From the practical point of view, in the execution step, the welding of multiple workpieces on the scene can bring the robot joints out-of-range. In fact, the parameterization of closed geometries implies the presence of important variations in the configuration of the sixth joint, especially when multiple workpieces are on the scene. To avoid this case, an additional procedure has been designed. The move list evaluated in the previous step will be modified. For workpieces on the scene that are numbered as even, in order to invert the sense of rotation of the sixth joint, and avoid the reach of the joint limits, is possible to execute the welding procedure backward, with respect to the direction of welding of the demonstrated workpiece. For example, for three workpieces on the scene, if for the first workpiece the demonstration has been executed in clockwise direction, the second workpiece will be welded in counter-clockwise direction, while the third one will be welded again in clockwise direction. To implement this idea, the move list that is in output to the multiple workpiece method will be modified, just for workpieces numbered as even. For sure the order of the points to follow the shape of the workpiece is opposite, so the move list must be reversed. To achieve the final move list, some basic rules can be defined. For reference, an example can be described in Figure 4.4 and the movelists in 4.7, 4.8, 4.9.

$$\begin{aligned} OriginalMovelist = & [[1, robtarget_B], [3, robtarget_C], [2, robtarget_D, robtarget_E], \\ & [3, robtarget_F], [1, robtarget_G], [3, robtarget_H], \\ & [1, robtarget_I]] \end{aligned} \tag{4.7}$$

$$\begin{aligned} ReversedMovelist = &[[1, robtarget_I], [3, robtarget_H], [1, robtarget_G], \\ &[3, robtarget_F], [2, robtarget_D, robtarget_E], \\ &[3, robtarget_C], [1, robtarget_B]] \end{aligned} \tag{4.8}$$

$$\begin{aligned} FinalMovelist = &[[1, robtarget_H], [3, robtarget_G], [1, robtarget_F], \\ &[3, robtarget_E], [2, robtarget_D, robtarget_C], \\ &[3, robtarget_B], [1, robtarget_A]] \end{aligned} \tag{4.9}$$

First, whenever it's presented a sequence of MoveJ and MoveL, it's enough to assign to the

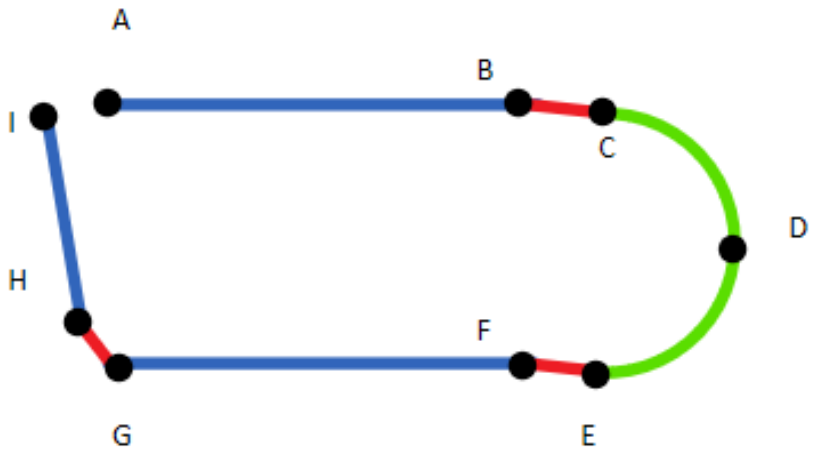


Figure 4.4: Parameterization of the workpiece, where blue lines correspond to a MoveL, the red ones to a MoveJ, and the green ones to the MoveC

new move command of the final move list, the integer number that refers to the previous move command in the reversed move list. This is because these commands require the end robtarget of the move segment, which corresponds to the robtarget of the start point of the same segment (or the robtarget of the endpoint of the previous segment) when the direction of the demonstration is inverted. This rule remains the same whenever a MoveJ or MoveL command is preceded by a MoveC command, with the only exception regarding the presence of the mid-point. In fact, the robtarget of the endpoint of the current move command in the reversed move list will be the robtarget of the endpoint of the MoveC in the final move list. Instead, the robtarget of the mid-point in the final move list will be the same as the one of the previous move command, the MoveC above, in the reversed move list, despite the direction of the demonstration. Finally, whenever there are two consecutive MoveCs, a second example can be useful, where Figure 4.5, can be described

by the movelists in 4.7, 4.8, 4.9.

$$\begin{aligned} OriginalMovelist = &[[1, robtarget_B], [3, robtarget_C], [2, robtarget_D, robtarget_E], \\ &[2, robtarget_F, robtarget_G], [3, robtarget_H], [1, robtarget_I], | \\ &[3, robtarget_L], [1, robtarget_M]] \end{aligned} \tag{4.10}$$

$$\begin{aligned} ReversedMovelist = &[[1, robtarget_M], [3, robtarget_L], [1, robtarget_I], \\ &[3, robtarget_H], [2, robtarget_F, robtarget_G], \\ &[2, robtarget_D, robtarget_E], [3, robtarget_C], [1, robtarget_B]] \end{aligned} \tag{4.11}$$

$$\begin{aligned} FinalMovelist = &[[1, robtarget_L], [3, robtarget_I], [1, robtarget_H], \\ &[3, robtarget_G], [2, robtarget_F, robtarget_E], \\ &[2, robtarget_D, robtarget_C], [3, robtarget_B], [1, robtarget_A]] \end{aligned} \tag{4.12}$$

The robtarget of the two mid-point is conserved, instead, the first MoveC in the final

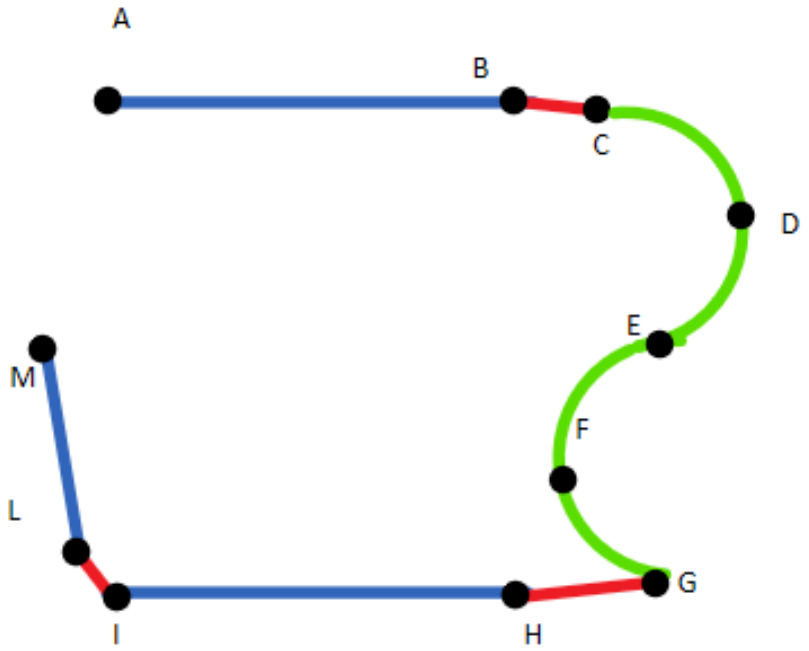


Figure 4.5: Parameterization of the workpiece, with two consecutive MoveC

movelist will have the end robtarget of the second MoveC of the reversed movelist. Furthermore, the second MoveC in the final movelist will have the end robtarget equal to the start robtarget of the first MoveC of the original movelist.
Concluding, the method will take as input the original move list, and gives as output the

final movelist.

5 | Experimental validation

5.1. Experimental Setup

The validation procedure setup is performed in three main steps, and each one of them is executed on a different workpiece. It's important to point out that the parameterization depends on the correctness of the demonstration performed by the operator. In fact, for example, if the operator moves the end effector linearly in a circular portion of the shape of the workpiece, the algorithm will parameterize this part of the trajectory as a linear segment. So, the performances of the algorithm will be conditioned on the "quality" of the demonstration. In particular, we can summarize three main claims:

- Claim one: correctness in the trajectory parameterization
- Claim two: correctness in the execution of the welding, also for a linear pattern of workpieces
- Claim three: correctness in the execution of the welding of multiple workpieces with different orientation

Each claim will be related to an experiment. Experiment one aims to validate claim one, and it's depicted in Figure 5.1.
This step is not directly focused on the execution of the parameterized trajectory, but more on its correct parameterization. The goal of this validation step is to evaluate the magnitude of possible parameterization errors like a wrong position identification of the arc mid-point of a specific MoveC, or a mismatch in the evaluation of the parameterized segment. Experiment two aims to validate claim two, and it's depicted in Figure 5.2.
This step of the validation is performed on a more complex workpiece that presents two consecutive MoveC, with opposite concavity, and five linear segments with different orientations. The scene is composed of two workpieces of the same shape, with the second one that presents the same orientation as the first one, The scope of this validation stage is to evaluate the execution of the parameterized trajectory on both the workpieces, analyzing the correctness of the orientation of the end effector along the trajectory, as

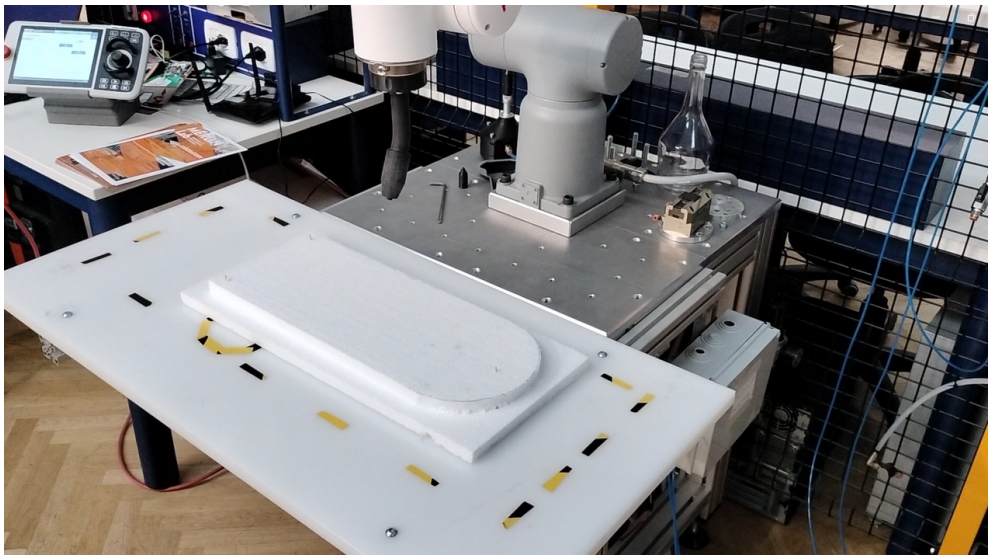


Figure 5.1: Workpiece number one

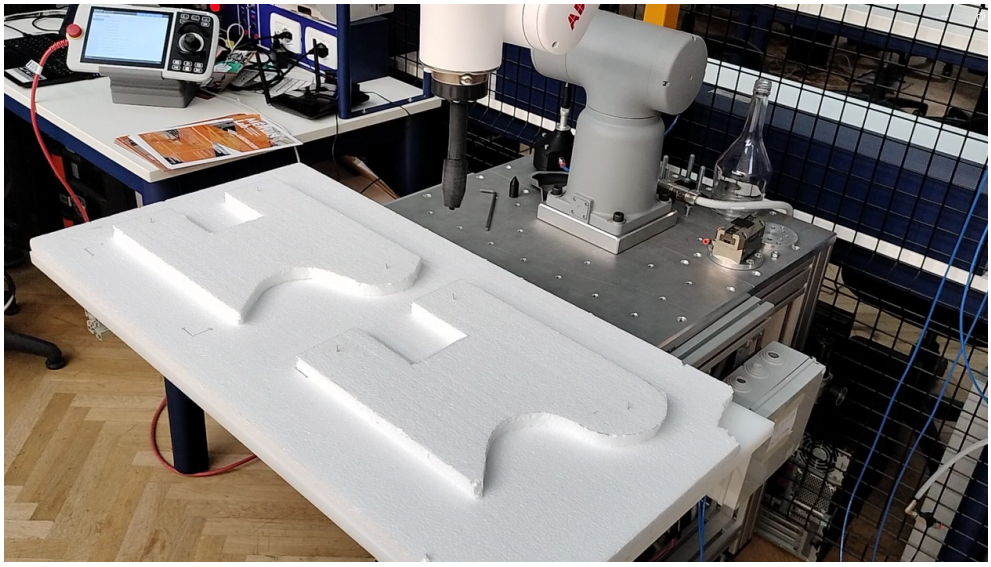


Figure 5.2: Workpiece number two

well as the parameterization of the shape of the workpiece. Furthermore, every time that the robot joint goes out-of-range on the execution of the demonstrated trajectory, even if during the demonstration this configuration is not present, this will be considered an error.

Experiment three aims to validate claim three, and it is depicted in figure 5.3.

The third step of the validation procedure is essentially equal to step 2, but the geometry of the workpiece is more complex. In fact, it presents a circular portion immediately followed by a linear one, with a sensibly small shape variation, some linear portions of geometry that are smaller with respect to workpiece 2, and another circular portion that is smaller

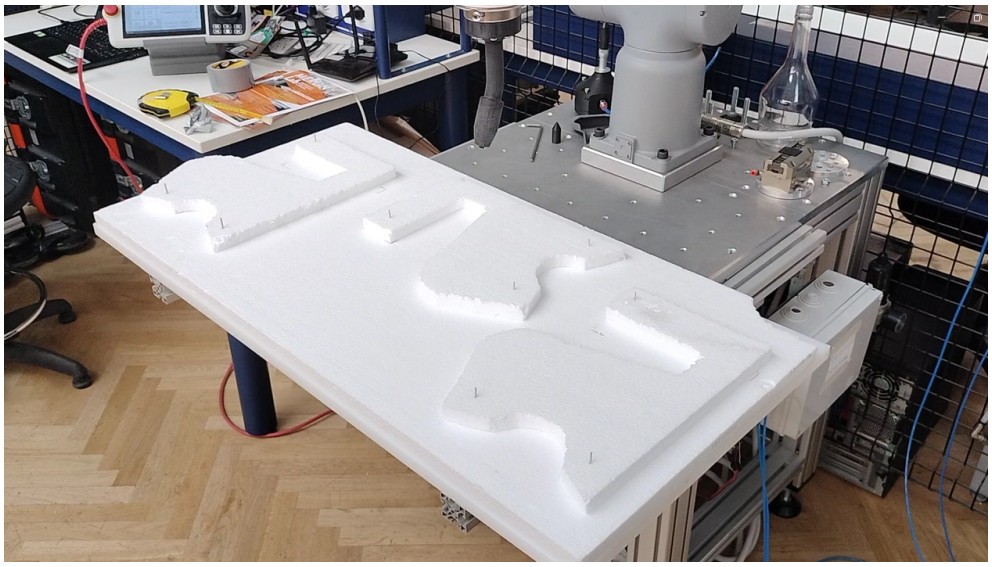


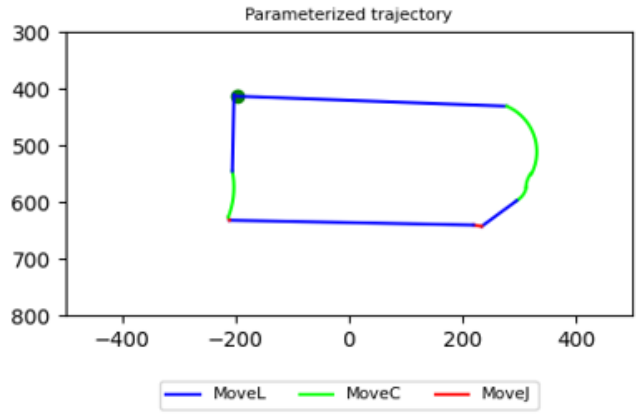
Figure 5.3: Workpiece number three

with respect to workpiece two. Furthermore, the scene is composed of three workpieces of the same shape. The second and third workpieces are rotated with respect to the first one by 90 degrees in the clockwise direction. The experimental validation involved five different non-expert candidates. Each one has performed 10 different demonstrations on each workpiece, resulting in 30 demonstrations for each candidate, and 50 demonstrations on each workpiece, for a total of 150 demonstrations.

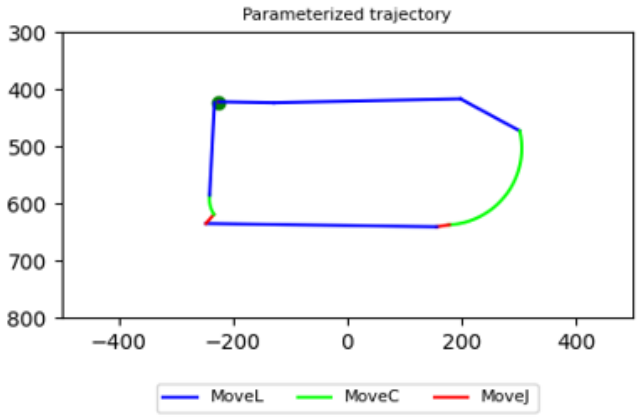
5.2. Results and discussion

5.2.1. Experiment one

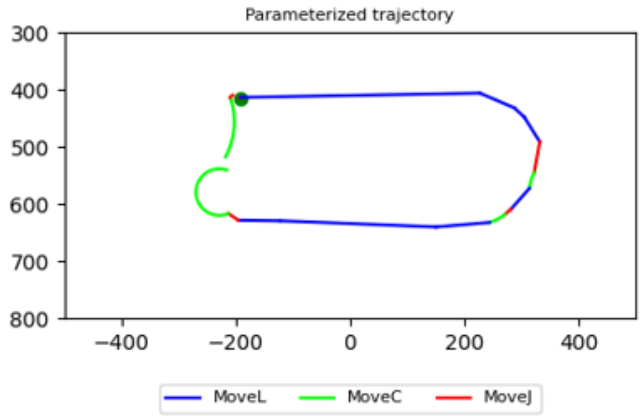
For workpiece 1, the algorithm parameterizes correctly the trajectory in 19/50 demonstrations. However, as mentioned above, these performances are dependent on the correctness of the demonstration. The results on workpiece 1 show that the algorithm misses the correct evaluation of the MoveL when there is a sudden change in the shape of the workpiece, characterized by a MoveJ segment followed by a MoveC segment, and the angle that describes γ is in the proximity of the discontinuity value -π. Examples of this situation are depicted in Figures 5.4(a),5.4(b), 5.4(c), where the area of interest is the bottom right part of each workpiece.



((a))



((b))



((c))

Figure 5.4: Examples of wrong parameterization in Stage One

5.2.2. Experiment two

For Stage Two, the results are shown in Figure 5.5.

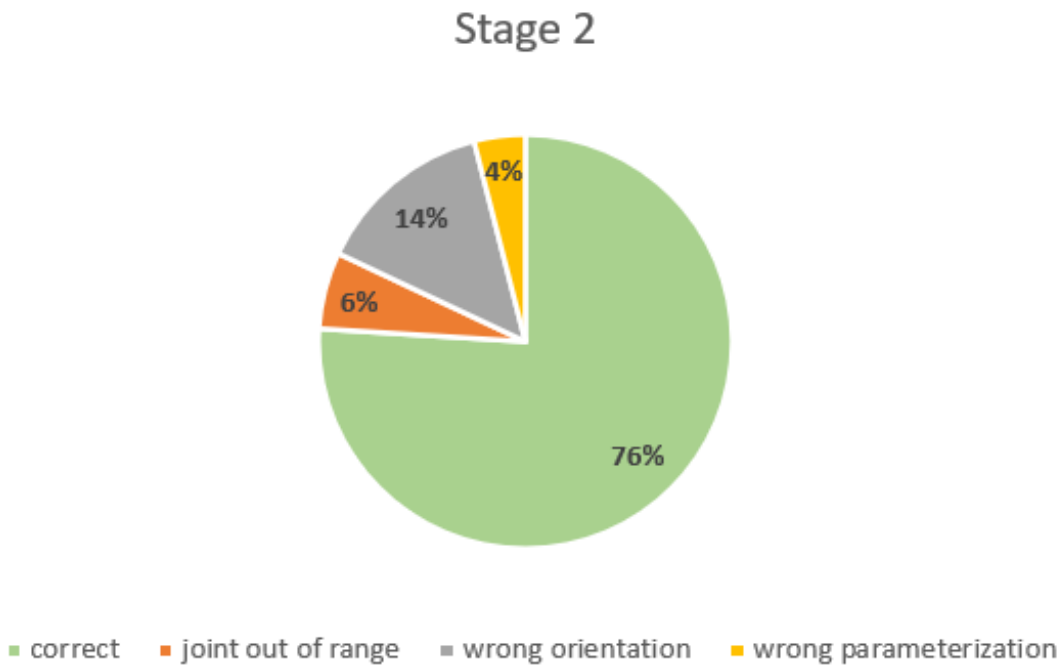
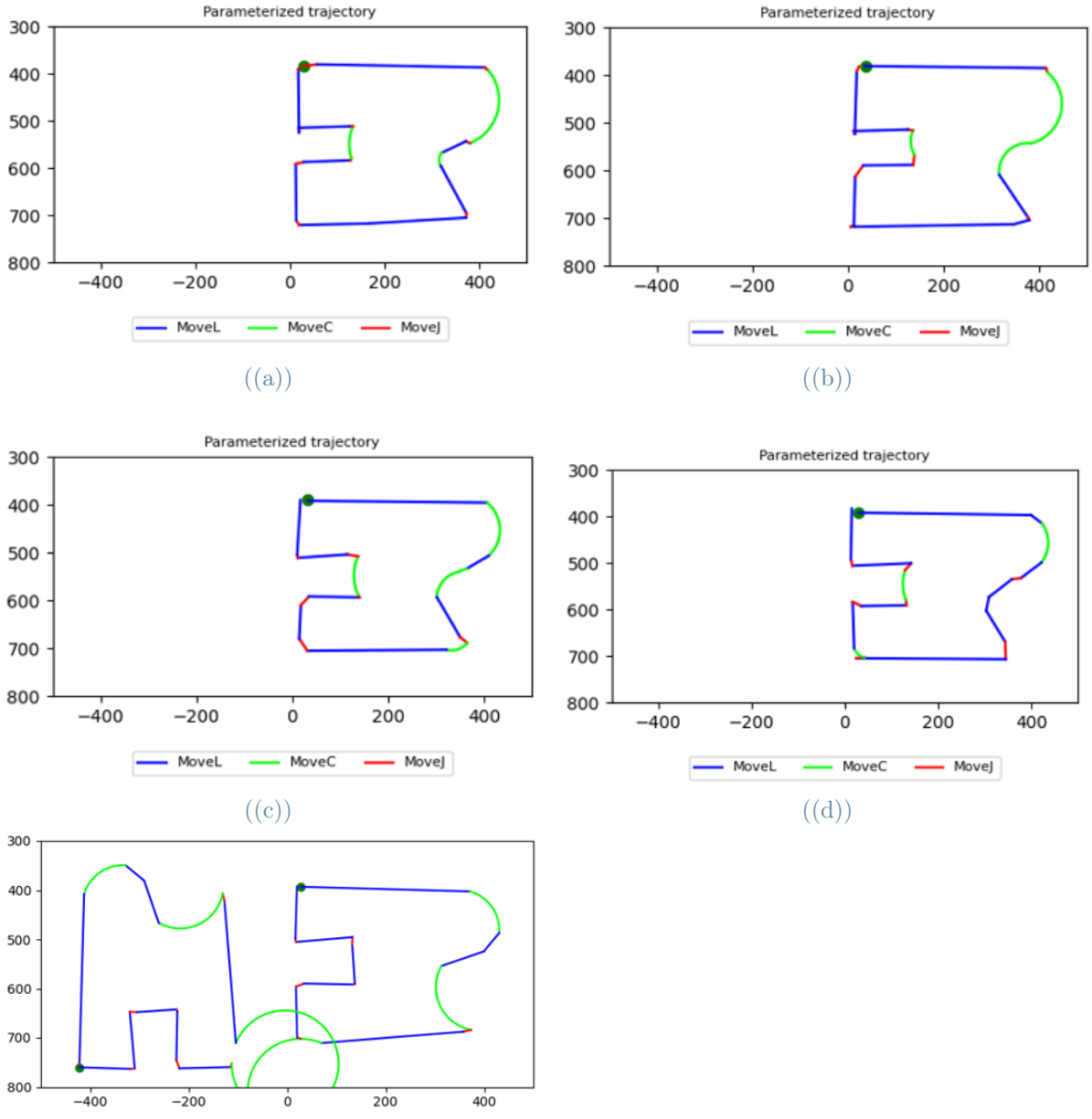


Figure 5.5: Validation Stage Two

In particular, 76% of the executions are correct, instead, 24% present multiple errors. Thus, 25% of the wrong executions are caused by the sixth joint going out-of-range, 58.3% is caused by a wrong orientation of the tool along the geometry of the workpiece, and 16.7% is caused by a wrong parameterization of the trajectory. The algorithm presents a lack of capability to execute correctly the MoveC whenever the geometry is such that the γ angle reaches its discontinuity value. This problem is underlined especially in the execution of the green portion of the trajectory in the bottom part of the workpieces, highlighted in green in Figures 5.6(a), 5.6(b). Furthermore, the algorithm is still not robust in the execution of small portions of circular geometry, as depicted in Figures 5.6(c), 5.6(d). Also, in this case, the algorithm chooses the wrong direction along which the tool is oriented, with respect to the center of the circumference of the MoveC. Possible causes are wrong management of the value of the concavity for that specific MoveC, or wrong evaluation of the sign of the angle that rotates the tool reference frame in the direction of the center of the circumference. There is also a case of wrong parameterization in Figure 5.6(e), resulting in a MoveC being too big to be executed by the robot. In this case, the algorithm selects the wrong arc, and thus, the wrong mid-point.



((a))

((b))

((c))

((d))

((e))

Figure 5.6: Examples of wrong execution in Stage Two

5.2.3. Experiment three

For Stage Three of the validation, the results are shown in Figure 5.7.
In particular, 80% of the executions are correct, and 20% present errors. In fact, 40% of

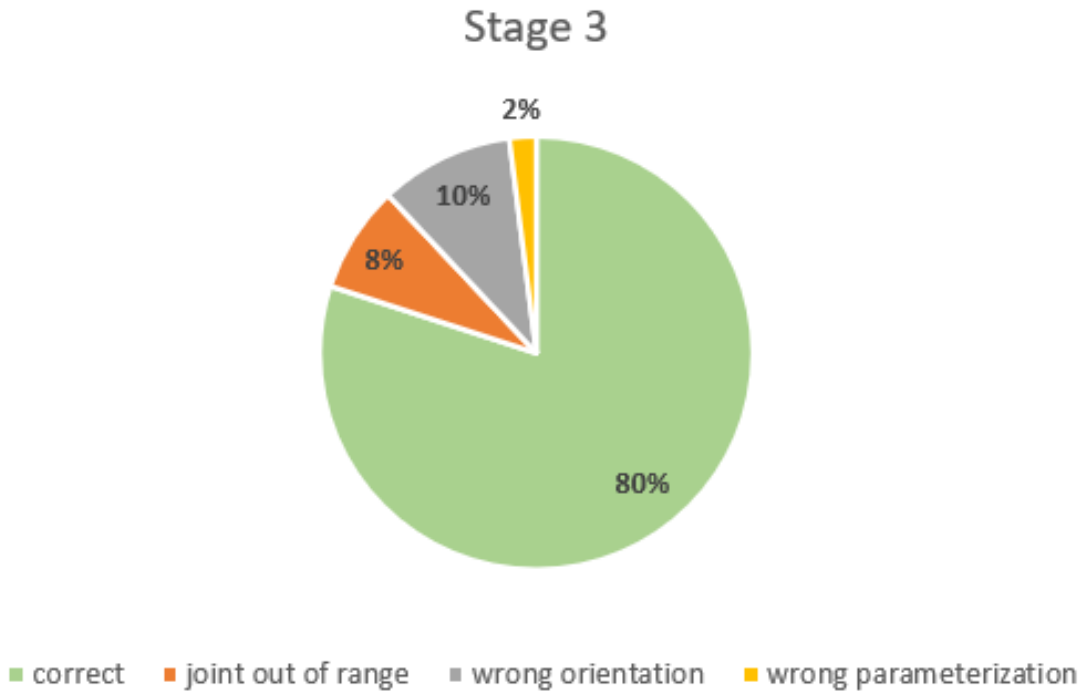
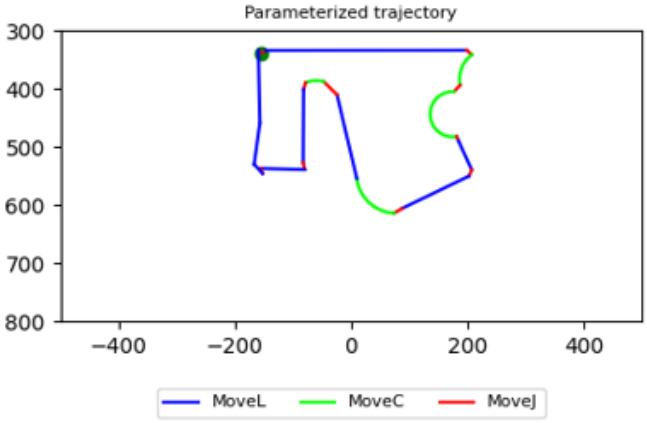
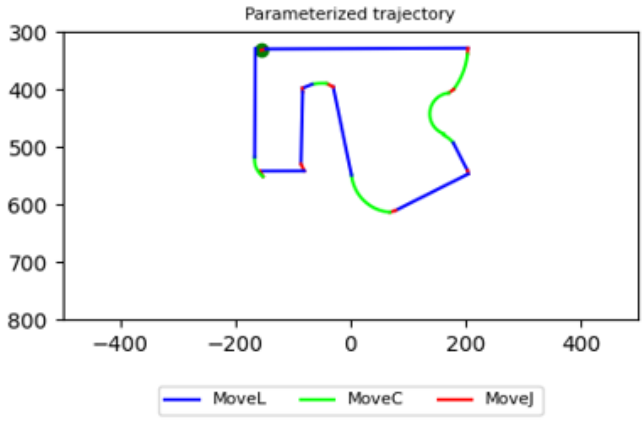


Figure 5.7: Validation Stage Three

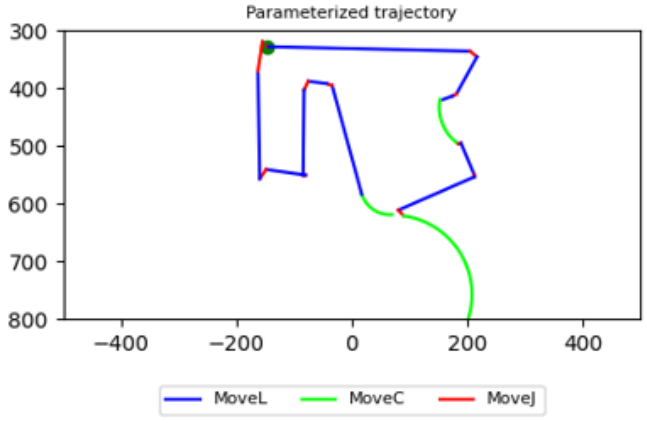
the wrong executions are caused by the sixth joint going out-of-range, 50% is caused by a wrong orientation of the tool along the geometry of the workpiece, and 10% is caused by a wrong parameterization of the trajectory. Stage 3 confirms the errors present in the other stages. The algorithm suffers the correct execution of small portions of geometry, like the first MoveC in clockwise direction for Figure 5.8(a),5.8(b) or parameterization errors, like Figure 5.8(c).



((a))



((b))



((c))

Figure 5.8: Examples of wrong execution in Stage Three

6 | Conclusions

6.1. Thesis overview

The objective of this thesis work is the development of a method for easy and quick programming of a welding task, called GoFa SmartWeld, that is able to overcome the main limitations of PbD, fostering its applicability in industrial scenarios. The methodology proposed aims to parameterize the trajectory recorded in one single kinesthetic demonstration of a welding task. In particular, since during the kinesthetic demonstration the operator has to save specific waypoints in order to execute the task, the objective of the method is to define an automatic procedure for the extraction of skill parameters from demonstration. For our purpose, the skill parameters of target position, end effector orientation, and target velocity are extracted. Furthermore, the robot program generated towards previously saved waypoints is valid only on the same scene of the demonstration, with the same objects in the same positions and orientations. Thus, if the number, positions, and orientations of the objects change, the robot program will be no longer valid. For this reason, our method aims to adapt the extracted parameters in different situations, where multiple workpieces of the same shape are present in the welding table, in different positions and orientations.
Related works in this field proved to be inadequate for the task of easily programming a cobot for a welding process through a one-shot kinesthetic demonstration.
The experimental validation aims to evaluate the correctness of the trajectory parameterization in experiment 1. Furthermore, the correctness of execution for a linear pattern of workpieces, and for multiple workpieces with different orientations, is evaluated respectively in experiment 2 and experiment 3. From the results point of view, our algorithm is able to execute correctly the parameterized trajectory in 76% of the cases for experiment 2, and 80% of the cases for experiment 3. Anyway, correct parameterization strongly relies on the correctness of the demonstration performed by the operator. In fact, experiment 1 shows that the correct parameterization takes place in 19/50 demonstrations.

6.2. Future studies

Possible future developments of the work include the management of the values of position and orientation of the different workpieces on the scene, that so far are provided in input to the method. The integration of a vision system on the robot may be evaluated, in order to detect automatically the different locations and orientations of the workpieces on the welding table.
Another possible development includes the extension of the proposed method to non-planar welding tasks.

Bibliography

[1] Rabab Benotsmane, L Dudás, and Gy Kovács. Collaborating robots in industry 4.0 conception. In *IOP Conference Series: Materials Science and Engineering*, volume 448, page 012023. IOP Publishing, 2018.

[2] Shirine El Zaatari, Mohamed Marei, Weidong Li, and Zahid Usman. Cobot programming for collaborative industrial tasks: An overview. *Robotics and Autonomous Systems*, 116:162–180, 2019.

[3] Anand Nayyar and Akshi Kumar. *A roadmap to industry 4.0: smart production, sharp business and sustainable development*. Springer, 2020.

[4] Afonso Amaral, Diogo Jorge, and Paulo Peças. Small medium enterprises and industry 4.0: current models'ineptitude and the proposal of a methodology to successfully implement industry 4.0 in small medium enterprises. *Procedia Manufacturing*, 41: 1103–1110, 2019.

[5] Aly M Eissa, Mostafa R Atia, and Magdy R Roman. An effective programming by demonstration method for smes' industrial robots. *Journal of Machine Engineering*, 20, 2020.

[6] Eloise Matheson, Riccardo Minto, Emanuele GG Zampieri, Maurizio Faccio, and Giulio Rosati. Human–robot collaboration in manufacturing applications: A review. *Robotics*, 8(4):100, 2019.

[7] Joseph E Michaelis, Amanda Siebert-Evenstone, David Williamson Shaffer, and Bilge Mutlu. Collaborative or simply uncaged? understanding human-cobot interactions in automation. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2020.

[8] Franz Steinmetz, Verena Nitsch, and Freek Stulp. Intuitive task-level programming by demonstration through semantic skill recognition. *IEEE Robotics and Automation Letters*, 4(4):3742–3749, 2019.

[9] Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. Handbook of robotics, chapter robot programming by demonstration. number 59. *MIT Press*, 10: 978–3, 2007.

[10] Oliver Kroemer, Scott Niekum, and George Konidaris. A review of robot learning for manipulation: Challenges, representations, and algorithms. *The Journal of Machine Learning Research*, 22(1):1395–1476, 2021.

[11] Sylvain Calinon. Learning from demonstration (programming by demonstration). *Encyclopedia of robotics*, pages 1–8, 2018.

[12] Maj Stenmark and Elin A Topp. From demonstrations to skills for high-level programming of industrial robots. In *2016 AAAI fall symposium series*, 2016.

[13] Tom Silver, Rohan Chitnis, Joshua Tenenbaum, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning symbolic operators for task and motion planning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3182–3189. IEEE, 2021.

[14] Mikkel Rath Pedersen, Lazaros Nalpantidis, Aaron Bobick, and Volker Krüger. On the integration of hardware-abstracted robot skills for use in industrial scenarios. In *Proceedings of the IEEE/RSJ International Conference on Robots and Systems, Workshop on Cognitive Robotics Systems: Replicating Human Actions and Activities*, 2013.

[15] Mikkel Rath Pedersen and Volker Krüger. Gesture-based extraction of robot skill parameters for intuitive robot programming. *Journal of Intelligent & Robotic Systems*, 80:149–163, 2015.

[16] Mikkel Rath Pedersen, Lazaros Nalpantidis, Rasmus Skovgaard Andersen, Casper Schou, Simon Bøgh, Volker Krüger, and Ole Madsen. Robot skills for manufacturing: From concept to industrial deployment. *Robotics and Computer-Integrated Manufacturing*, 37:282–291, 2016.

[17] Samir Vojić. Applications of collaborative industrial robots. *Machines. Technologies. Materials.*, 14(3):96–99, 2020.

[18] Harish Kumar Arya and Kulwant Singh. Effect of current, voltage and travel speed on micro hardness of saw welded mild steel plate.

[19] Nuno Mendes, Pedro Neto, Altino Loureiro, and António Paulo Moreira. Machines and control systems for friction stir welding: a review. *Materials & Design*, 90: 256–265, 2016.

[20] Rahul Kanti Nath, Pabitra Maji, Atosh Kumar Sinha, Ranit Karmakar, and Pritam Paul. Effect of different electrode angles as well as weld direction on the bead geometry of submerge arc welding process. *Materials Today: Proceedings*, 49:1793–1798, 2022.

[21] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

[22] Vamsi Krishna Origanti, Thomas Eiband, and Dongheui Lee. Automatic parameterization of motion and force controlled robot skills. In *Robot Intelligence Technology and Applications 6: Results from the 9th International Conference on Robot Intelligence Technology and Applications*, pages 66–78. Springer, 2022.

[23] Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. Learning and generalization of motor skills by learning from demonstration. In *2009 IEEE International Conference on Robotics and Automation*, pages 763–768. IEEE, 2009.

[24] Jens Kober, Michael Gienger, and Jochen J Steil. Learning movement primitives for force interaction tasks. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3192–3199. IEEE, 2015.

[25] Xiao Gao, Jie Ling, Xiaohui Xiao, and Miao Li. Learning force-relevant skills from human demonstration. *Complexity*, 2019, 2019.

[26] Adam Conkey and Tucker Hermans. Learning task constraints from demonstration for hybrid force/position control. In *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, pages 162–169. IEEE, 2019.

[27] Benjamin Alt, Darko Katic, Rainer Jäkel, Asil Kaan Bozcuoglu, and Michael Beetz. Robot program parameter inference via differentiable shadow program inversion. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4672–4678. IEEE, 2021.

[28] Aleš Ude, Andrej Gams, Tamim Asfour, and Jun Morimoto. Task-specific generalization of discrete and periodic dynamic movement primitives. *IEEE Transactions on Robotics*, 26(5):800–815, 2010.

[29] Arne Wahrburg, Stefan Zeiss, Björn Matthias, Jan Peters, and Hao Ding. Combined pose-wrench and state machine representation for modeling robotic assembly skills. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 852–857. IEEE, 2015.

[30] Karinne Ramirez-Amaro, Michael Beetz, and Gordon Cheng. Transferring skills to humanoid robots by extracting semantic representations from observations of human activities. *Artificial Intelligence*, 247:95–118, 2017.

[31] Jens Kober, Andreas Wilhelm, Erhan Oztop, and Jan Peters. Reinforcement learning to adjust parametrized motor primitives to new situations. *Autonomous Robots*, 33: 361–379, 2012.

[32] Kalesha Bullard, Baris Akgun, Sonia Chernova, and Andrea L Thomaz. Grounding action parameters from demonstration. In *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 253–260. IEEE, 2016.

[33] Aljaž Kramberger, Andrej Gams, Bojan Nemec, Dimitrios Chrysostomou, Ole Madsen, and Aleš Ude. Generalization of orientation trajectories and force-torque profiles for robotic assembly. *Robotics and autonomous systems*, 98:333–346, 2017.

[34] Alex Mitrevski, Paul G Plöger, and Gerhard Lakemeyer. Representation and experience-based learning of explainable models for robot action execution. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5641–5647. IEEE, 2020.

[35] Simon Manschitz, Michael Gienger, Jens Kober, and Jan Peters. Mixture of attractors: A novel movement primitive representation for learning motor skills from demonstrations. *IEEE Robotics and Automation Letters*, 3(2):926–933, 2018.

[36] Federica Ferraguti, Valeria Villani, and Chiara Storchi. Mywelder: A collaborative system for intuitive robot-assisted welding. *Mechatronics*, 89:102920, 2023.

[37] Rebecca Hollmann, Arne Rost, Martin Hägele, and Alexander Verl. A hmm-based approach to learning probability models of programming strategies for industrial robots. In *2010 IEEE International Conference on Robotics and Automation*, pages 2965–2970. IEEE, 2010.

[38] Bela Takarics, Peter T Szemes, Gyula Németh, and Peter Korondi. Welding trajectory reconstruction based on the intelligent space concept. In *2008 Conference on Human System Interactions*, pages 791–796. IEEE, 2008.

[39] Steven W Smith et al. The scientist and engineer's guide to digital signal processing, 1997.

List of Figures

List of Tables

List of Symbols

Variable	Description	SI unit
γ_i	angle of the oriented segment	degrees
p_i	position of the point of dataset	mm
x_i	x-coordinate of the point of dataset	mm
y_i	y-coordinate of the point of dataset	mm

Acknowledgements

Desidero ringraziare tutti coloro che mi hanno aiutato nello sviluppo di questo lavoro. In particolare un ringraziamento è indirizzato ai professori Paolo Rocco e Andrea Maria Zanchettin, che mi hanno permesso di sviluppare il progetto, a Isacco, che mi ha aiutato nello sviluppo, e a tutti i ragazzi del MeRLIn Lab.
Dedico questo lavoro a chi mi è stato accanto, a chi mi ha aiutato e supportato in questi anni, e a chi avrebbe voluto farlo, ma non ha potuto. A Valentina, alla mia famiglia, a chi viaggia in direzione ostinata e contraria.