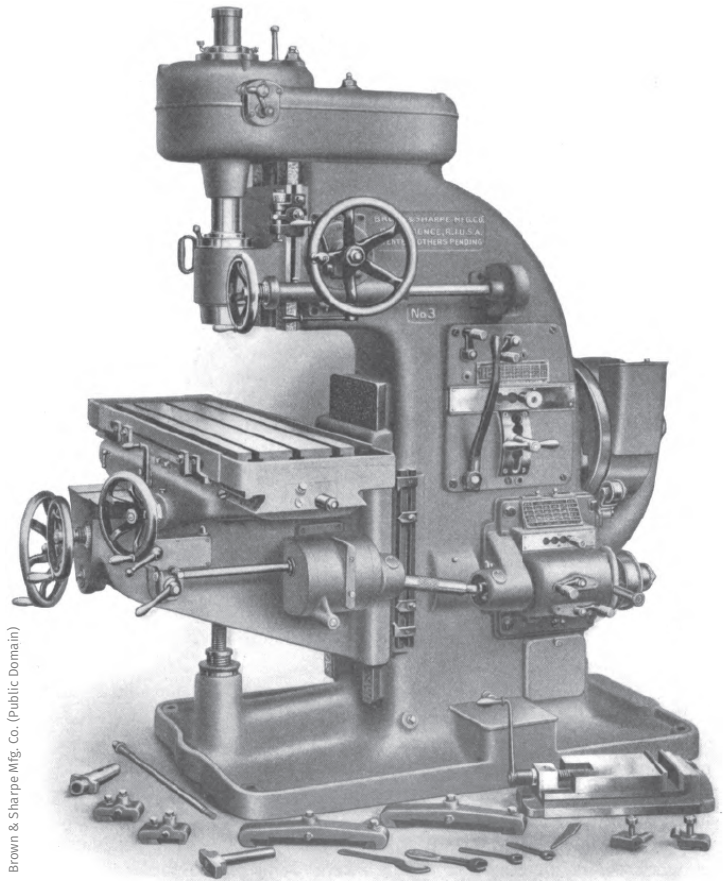
UNIVERSITY
OF TRENTO - Italy

# INTELLIGENT MANUFACTURING

## Engaging Industry 4.0 Challenges through Emerging Technologies




Brown & Sharpe Mfg. Co. (Public Domain)

**MATTEO RAGNI**

Advisor: **PAOLO BOSETTI**

2014 / 2017 — XXX Cycle

**Dept. Industrial Engineering**

School of Materials,
Mechatronics and
Systems Engineering

*Obriously for you, Laura*

*A lot of people should be thanked for the work presented in this thesis, and for the help they provided me. I want to start with my family, because without them such a result would have been simply impossible. I want to thanks Paolo Bosetti, a great advisor and teacher for my PhD journey, Francesco Biral and Enrico Bertolazzi for their support, suggestions and the always useful discussions. Amedeo, Matteo, Luca, Mirko, Andrea 1 and 2, Mattia, Sultan, Roberto, Alessandro and Silvia: you guys are the greatest minds I have ever met, and I'm very lucky for having shared this adventure with you. During my period abroad I've had the opportunity to explore some research path that kept alive my interest day and night, thanks to the suggestions of David Windridge and the company of Franco, Giuseppe, Lorenzo, Elisabetta and Michele. A special thanks to Amedeo, who has written all the source code of Artool, that is in this thesis, and a special thanks also to Matteo Cocetti, for his time, his consideration, his thoughtfulness, and his guidance.*

# Contents

# 1 Introduction

*The Industry 4.0 challenge is to exploit the synergy of different technologies in order to achieve the results required by its specifications. This chapter presents: (a) the state of the art in Augmented Reality applied to industrial engineering and manufacturing machines, (b) insights on the implementation of optimal feed-rate interpolation for computer numerical control machine tools, (c) an application of knowledge-based techniques such as computer algebra systems in the implementation of solvers for optimal control problems, and (d) challenges in the application of artificial neural networks to the massive amount of unlabeled data available in the industrial practice. It is shown how these topics, wich may appear as distant one from each other, play a central and correlated role in the Industry 4.0.*

## 1.1 INTELLIGENT MANUFACTURING: AN OVERVIEW

The industrial world is preparing itself, since 2011, for the fourth industrial revolution. It was in the Hannover Messe that the term *Industry 4.0* was coined after an initial investigation from the German Government.

Leaving aside the strong criticism arisen around the term *revolution*, the strategic plan considers four main topics that empower the general industrial hype around this topic.

**Interoperability** a strong emphasis is given to the evolution from

automated systems to cyber-physical systems that extend beyond the single machine, to embrace the whole plant, if not the entire set of productive assets of a geographical region. Cyber-physical systems operates on multiple levels: physical/mechanical, software, and communication technologies that characterize them are deeply intertwined and expose to the external world a series of behavior that reacts to a given context. When the cyber-physical system embodiment of an industrial system is extended way beyond the single machine, it is immediate to raise the term *interoperability* as one of the critical and challenging aspect of such heterogeneous systems. Main research topics focused on this aspect are Internet of Things (IoT), Industrial IoT, and their interaction with Internet of People (IoP).

**Information Transparency** the new cyber-physical systems may be formalized with the common paradigm of a perception—action agent, where the perception is actually the constellation of sensors typically employed in industrial contexts that produce a massive amount of raw data, as a projection of the physical environment observable by the system. The representation of this world is probably redundant, and one of the biggest challenge is the aggregation of data in a more compact space, where higher-value information (with respect to the context) are transparently available to the system itself (to perform or select the subsequent action) or to the other agents that interacts with such system.

**Technical Assistance** this topic tackles two main aspects of the interface between systems and human operator; the first aspect is strictly related with the information transparency topic, since it interests the presentation of the data to the human agents in a comprehensibly and contextualized form, to ease the decision process or the information gathering tasks. As for the second aspect, the technical assistance implies the capability of the system in physically performing some particular tasks that

are currently fulfilled by human operators. Those operations range from the critical, the unsafe, the unpleasant, and even the exhausting, in order to relief humans.

**Decentralized Decisions** the fourth topic regards the autonomous level of the cyber-physical system. While a single machine may be considered autonomous in performing a very specific task and capable of correct deviation from the requested operation (Autonomous level A4 *closed loops* [3]), the approach is to make the whole system capable of abstracting and deciding the strategy to achieve a specific goal (Autonomous level A7 *reasoning*), with the minimum waste of assets. The system may access different abstraction layers of the perceived reality, in order to tackle a particular problem.

Those aspects underline the stronger importance of software in the industrial world, in particular in the form of optimization and intelligent algorithms. There is no unique solution to the challenges that are raised by the Industry 4.0 plan. Like tiles of a very big puzzle, different methodologies should be employed in synergy, over existing hardware, to build the envisioned cyber-physical system. The long term inspiration is to detach the human expertise from the shop floors to obtain the *from art to part* paradigm, a result that is unfortunately still far from being achieved.

The following chapters will present different approaches and technologies (augmented reality, optimal control, knowledge based systems, and machine learning algorithms) that have been explored in order to tackle some of the four Industry 4.0 topics. The synergy of the different technologies allows to synthesize a framework to tighten the gap between human programmed machine and Computer Numerical Control (CNC) machines.

## 1.2 MACHINING ECONOMICS AND AR

The economics of machining operations considers different cost authorities that should be minimized to achieve an efficient process. For each machined product, the main factors to consider are [39, 53]:

- the cost of the effective machining operation, alongside with maintenance and man-hours costs;
- the cost for preparing the machine, which comprises testing of the part-program, fixing and aligning the blank material in the working area, and mounting the tools and the cutters on tool holders;
- the costs for loading the raw material and unloading the finished part;
- the cost of tooling.

One of the cost of strongest impact is related to maintenance and inactivity that directly correlates time and machining costs. In case of human operator involved in the process of loading and unloading material—e.g. in case case of shop-floor with limited automation and with small batches to be produced—optimizing the maintenance and the alignments procedures permits to reduce dramatically the costs.

The application of AR technologies to manufacturing can boost the efficiency [58], while reducing the error rate.

### 1.2.1 *Envisioning AR Technologies*

The manufacturing industry has always envisioned the application of AR related technologies, and the strong interest is underlined in the results of the survey conducted by the *Deutsche Forschungszentrum für Künstliche Intelligenz* during the Hannover Messe of 2010. On a

total of 54 industrial rappresentative, 77.8 % have every intention of deploying augmented solutions in their production lines [70].

In literature, Architecture is the first field that embraced the AR, enlarging the Building Information Modeling schemes in order to accomodate a data infrastructure for the Augmented Reality technologies [91].

Also the Cognitive Sciences inspected the application of AR technologies, evaluating the benefit from a cognitive workload point of view [48, 73].

In general, the proofs-of-benefit for AR as alternative training method, described in Refs. [35, 67], make educational and informational applications, such as augmented manuals and operators training, literally mainstream. In Ref. [59], the authors use a marker solution to build interactive lectures on machinery handling for completely inexperienced students, revealing once again the high acceptance of the methodology, and allowing a faster comprehension of programming caveats for complex paths [24]. Ref. [78] pushes towards the integration of AR for training and expert systems to support decision making for inexperienced operators.

The costs of integrating such a new technology in the process is not an easy decision. Few studies started to develop decision supporting tools *ex-ante* [31], for evaluating the effectiveness of the approach for a specific manufacturing process. Both Product Design and Planning (PDP) and Workplace Design and Planning (WDP) benefit from an AR developing environment [63], that aid designers and engineers in making better decision while designing new assembly lines. Lines include AR interfaces [12, 19] that guide the operator in the execution of a specific task — i.e. projecting welding spots on work-piece in [29]. The ergonomy of the technology is also evaluated in literature [87].

Refs. [48, 92] present first implementations of virtual assembly interfaces. Cameras are used to detect position of operator hands, that are the Human Computer Interface (HCI) for the augmented renderer.

Systems are desktop static prototypes, but usability is validated with respect to non-augmented real-case-scenario. Evidence of cognitive workload reduction for the operator are underlined, as also reduced time to complete tasks and reduced mean error rate.

For what concerns application on process machines, the manipulators programming and collision avoidance is for sure the most prolific field. And in fact the complex kinematic configurations during a program execution results more intuitive—e.g. programming [32] or visualizing [21, 33] end-effector pose and trajectory,—by the mean of different user interface—e.g. mobile, projection on half silvered glasses or head displays [51]. General survey can be found in Refs. [62, 60].

The applications of AR on machine tools are limited and may be referred as proof-of-concept prototypes rather than proof-of-benefit ones. In Ref. [77], an AR application is used to help operators during manual alignment in a pipe manufacturing machine. In Ref. [58], AR is used to develop a framework for dimensional validation of finished parts. The framework is marker based, one reliable solution that guarantees enough precision for manufacturing applications. The work also illustrates evidences of advantages, both economical and practical, induced by the use of AR applications in manufacturing. Another approach typically discussed in literature, is the use of superimposition of virtual image on work-space video recording for validation of complex paths [95]. Virtual images contain augmented information about the process, and are visualized through the use of different device, such as stereo-projector [61] or mobile devices. In general, the idea is to use the augmented visualization to give more insight to the operators about the process, usually before performing the actual machining operation [99]. Other applications focused on active maintenance, using OCR (Optical Character Recognition) in combination with localization markers [57], but real benefits of such implementations to users were not assessed. In Ref. [96], it is worth noting the use of handheld devices, with respect to the typical static

desktop setups seen in previous works.

### 1.2.2 *A Framework for AR in Manufacturing*

Chapter 2 introduces a framework that exploits Virtual, Augmented Reality and Optimal Control technologies to reduce unproductive times. The platform reduces errors induced by operators during procedures such as alignments of blank material and in touch-probe programming. In common practice, for avoiding collisions that may result in extended damages for both machine and work-piece, in-air test are performed—i.e. a complete execution of the part program with a constant safety offset between the tool and the raw material.

The Augmented Reality (AR) component of the framework, namely ARTool, uses the reference systems stored inside the machine controller to overlay a properly oriented simulation of the workpiece blank, alongside with fixtures, and machine moving peripherals on the scene of the working area captured by a camera. The simulation reflects exactly what the machine is programmed to perform, thus in-air test, which may require hours to be fully executed, is substituted by an augmented simulation where time can be scaled. The operator concentrates the attention only on the complicated passages, and effectively identify visually evident mistakes, in less time and with an higher accuracy.

The augmented component of the framework is built to run on a personal device, and the considered device is a tablet, which is relatively low-cost with respect to more exotic hardware—e.g. head-mounted displays. With a tablet, the operator explores the simulated scene from different perspectives. Moreover, the same framework can be easily employed to enhance the maintenance operations on a machine, and inexperienced operators largely benefit from the usage of augmented schematics and manuals.

An interesting application regards the automatic programming of touch-probes. Modern machine tools implement specific code blocks for touch-probes, based on the assumption that the operators already have a rough knowledge about positions and orientations of fixed workpieces in working space. At least, an approximate measurement is required, and tolerance parameters for approaching must be set. Some simulations can be seen on the CNC screen before execution, although no feedback is provided with respect to real objects in workspace. Those procedures require time and experience, since errors bring to catastrophic damages for machine, workpieces, and touch-probes.

To employ ARTool in this context, the framework has been expanded to become a complete input-output interface. Generation of part-programs for touch-probes aims at reducing setup time related to block form alignment. This application is extended to the measurement of different geometric primitives, which are the basic blocks for the identification of complex geometric features. Trajectory of the probe is generated starting from the information collected by the camera and the user input; the device is a mobile tablet, which allows to project on camera feed the simulation of the generated listing and send it to CNC for the actual execution.

To improve furthermore the performances, the generated trajectory is optimizable with respect to machine dynamics. Performance of machine tools depends on the algorithms that the Computer Numerical Control (CNC) implements for calculating feed rate profiles. This performance impacts both the accuracy of the tool movements (path tracking), and the efficiency in terms of process time (or average tool speed) [76].

### 1.2.3 *Optimization of Part Programs*

Modern CNC systems are based on *acceleration/deceleration control before interpolation*. This is a software implementation, where the feed rate profiles (i.e. profiles of tangential speed) are generated before executing the motion interpolation on individual machine tool axes. Path tracking error for these CNC systems is theoretically limited by the performance of axes drivers, provided that the feed rate profiles are generated within the velocity, acceleration, and jerk feasibility limits for each axis.

A relatively open issue, though, remains when considering the cornering performance. Any sharp corner along the tool path represents a point of discontinuity in the velocity vector, so that the only physically feasible trajectory along the tool path must get to a full stop in the corners. For tool paths made by a succession of short segments, though, this approach greatly reduces the average feed rate compared to its nominal (or programmed) value.

This issue has been partially addressed by CNC manufacturers by allowing the CNC programmer to relax the path-following tolerance thus sacrificing the accuracy in favor of speed. It is the case of `G61` (exact path mode) and `G64` (continuous mode) G-code instructions [76, 50], where the former requires an accurate positioning at nodes by forcing a full stop, whether the latter enable higher average speed by allowing some limited tracking error during the acceleration/deceleration phases at the expense of tracking accuracy and thanks to the look-ahead approach [76, 2].

It has been shown that these solutions provide non-optimal trajectories both in terms of path tracking error, and in terms of minimum time [72]. In fact, the calculation of the feed rate profile is performed by assuming that the maximum allowed tangential acceleration is that of the slowest axis, so that the part program can be executed with comparable results whichever is the orientation of the workpiece to

be machined with respect to the machine tool axes—and this is by definition, since the feed rate profiles are calculated *before interpolation*.

Consequently, it is interesting to investigate the possibility to improve the machine tool performance by designing more advanced algorithms for feed rate profiling. Recent literature reports a number of works in manufacturing and in robotics field, proposing algorithms for motion interpolation of multiple axes aiming at improving time efficiency and path tracking accuracy. A comprehensive literature review on this subject has been proposed by Refs. [38, 28, 26, 27]. It is worth noting that most of the available literature deals with minimum-time feedrate profiles generated for a pre-defined tool path (i.e. admitting zero path tracking error). Clever solutions have also been proposed on this matter [86, 74, 9, 28, 49, 1, 52, 72, 100] that *do not explicitly relate the path tracking accuracy to optimization parameters*. In other words, the approach followed in these works is to calculate the optimal velocity profile satisfying a given set of limits, jerk included, assuming that the nominal tool path is the reference, but *without* enabling the perspective user to explicitly set a limit or constraint on the maximum allowable path tracking error, which is actually *what really matters from the product quality* point-of-view.

Here a different approach is adopted: the whole trajectory is optimized, rather than the sole feed rate profile. This choice allows to define a lateral tolerance on the tool path—which can be physically related to the workpiece design tolerance. On this basis, the formulation of an optimal control problem (OCP) minimizing the time and yet respecting constraints on maximum jerks and accelerations, proved effective in comparison to the execution time for part programs run on standard CNCs.

After the preprocessing of the original part program it is possible to generate an optimized code, made by a sequence of short straight segments, whose track is a discretization of the optimized tool path,

and whose feed rate are set to copy the optimized speed profile. The effectiveness of this approach is evaluated by comparing the cycle-time and the acceleration profiles of a reference part program with those of its optimized version.

The optimal control problem has been tackled with a numerical approach, by using the custom solver PINS. PINS is able to solve problems in the form:

$$\text{Minimize} \quad \Phi(\mathbf{x}(a), \mathbf{x}(b)) + \int_a^b J(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t)\mathrm{d}t \tag{1.1}$$

$$\text{subject to:} \quad \mathbf{M}(\mathbf{x}(t), \mathbf{p}, t)\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t), \tag{1.2}$$

$$\mathbf{b}(\mathbf{x}(a), \mathbf{x}(b), \mathbf{p}) = \mathbf{0} \tag{1.3}$$

It has an interface in *Ruby* while the core solver is a `C++` library. *Ruby* [36] is a purely object-oriented scripting language designed in the mid-1990s by Yukihiro Matsumoto, internationally standardized since 2012 as ISO/IEC 30170.

With the advent of the *Internet of Things*, a compact version of the *Ruby* interpreter called *mRuby* (*eMbedded Ruby*) [80] was published on *GitHub* by Matsumoto, in 2014. The new interpreter is a lightweight implementation, aimed at both low power devices and personal computers, and complies with the standard [81]. *mRuby* has a completely new API, and it is designed to be embedded in complex projects as a front-end interface—for example, PINS uses *mRuby* for problem definition.

The *Ruby* code-base exposes a large set of utilities in core and standard libraries, that can be furthermore expanded through third party libraries, or *gems*. Among the large number of available gems, *Ruby* still lacks an Automatic and Symbolic Differentiation (ASD) [83] engine that handles basic computer algebra routines, compatible with all different *Ruby* interpreters.

Nowadays *Ruby* is mainly known thanks to the web-oriented *Rails*

framework. Its expressiveness and elegance make it interesting for use in the scientific and technical field. An ASD-capable gem would prove a fundamental step in this direction, including the support for flexible code generation for high-level software—for example, IPOPT [90, 89].

*Mr.CAS*[1] is a gem—i.e. a dinamically loadable *Ruby* library—implemented in pure *Ruby* that supports symbolic differentiation (SD) and fundamentals computer algebra operations [88]. The library aims at supporting programmers in rapid prototyping of numerical algorithms and in code generation, for different target languages. It permits to implement mathematical models with a clean separation between actual mathematical formulations and conditioning rules for numerical instabilities, in order to support generation of code that is more robust with respect to issues that can be introduced by specific applications. As a long-term effort, it will become a complete open-source CAS system for the standard *Ruby* language, but has been specifically designed to become the main interface for PINS.

Other CAS libraries for *Ruby* are available:

***Rucas* [55], *Symbolic* [6]** : milestone gems, yet at an early stage and with discontinued development status. Both offer basic simplification routines, although they lack differentiation.

***Symengine* [20]** : is a wrapper of the *symengine* C++ library. The backend library is very complete, but it is compatible only with the *vanilla C Ruby* interpreter and has several dependencies. At best of Author knowledge, the gem does not work with *Ruby* 2.x interpreter, and cannot be employed in numerical code generation.

[1] Minimalistic *Ruby* Computer Algebra System

## 1.3 INDUSTRY AND AI

In the Industry 4.0 domain, the cyber-physical system perceives the world through sensors that produce a massive amount of raw data in which important semantic information are usually represented in a distributed way—i.e. for each sensory information, multiple semantic feature may be represented, while a single feature may be a representation of multiple possible sensorial input. One of the challenge is to actually extract the high-profile information that represents orthogonal and factorized data. Chapter 5 discusses and underlines what has been identified as a good research path to tackle the *representation problem*, reviewing what is currently well know in the Deep Learning field [40].

Probably everyone has heard at least once the words *Deep Learning* in the last two years. Deep Learning is for sure one of the emerging technology that is reshaping the future. A sort of revolution by itself, this research field is remodeling completely how computers perceive the world. Even if the basis for this technology must be searched years before the Second World War, under the name of *cybernetics* in 1940s and *connectionism* in the 1980s, it is only in the last ten years (2006 is also known as the year of *the third wave*) that the *artificial neural networks* have returned as one of the central topics for the scientific community [47].

This resurgence is strictly related with the advances in numerical mathematics (with the *stochastic gradient descent*), the application of graphic accelerator (GPUs), and the reduction in digital storage prices. The synergy between those elements has greatly amplified the computational power of common workstations, allowing researcher around the globe to finally train and test networks in a span of time that is extremely convenient. What in 2005 required months, in 2017 can be trained in few minutes. Also, existing libraries allow to distribute the computational requirements over different machines [43].

Nevertheless, the storage technology has increased the possible dimensions for training dataset.

Some of the architecture proposed in this field, like others in the Machine Learning Science, are able to partially disclose hidden structures in the data [7].

Looking at the recent literature, this problem is rarely been directly addressed in the industrial world, even if such property is extensively recognized and discussed in the application of machine learning techniques to industrial problem. The literature mainly focuses at the application of the machine learning science to a very limited task.

For example, Ref. [34] applies Sparse Linear + Discriminant Analysis to synchronous motor signals to categorize the status between normal, abnormal, and faulty behavior, showing how the three categories can be made separable after some feature engineering. In Ref. [97] the tool wear is forecast through a Random Decision Forest, while comparing with a single layered fully connected neural network. Even if the decision forest presented a better performance, also in this case the feature engineering is a prominent task that is manually handled. Ref. [82] applies clustering techniques to separate the overcome of the laser cutting process subject to some parameters. The clusters are then used to infer the optimal parameters to perform a specific cut. In Ref. [85], the quality control variables for the production of car body are analyzed in order to detect outliers. In this particular case the feature space is not reduced, but used directly. Ref. [25] uses Scale Invariant Transformations over images to identify a reduced representation. Subsequently, the reduced representation is used for classification of different powder types for additive manufacturing.

This pattern is repeated across the literature, but the lack of automatic knowledge retrieval is underlined strongly in Ref. [65], where machine learning is applied to an agent with wider tasks to perform. But it is in Ref. [37] that a deep network is employed to learn not only

the classification algorithm, but also a better representation for input data. In this particular case a Deep Belief Network is trained in a unsupervised way in order to reduce the feature space. Then, the reduced space is fine tuned for classification, applying a classical back propagation method. The network inspects input from accelerometers to monitor the cutting state during machining. The architecture of this last work refelcts some of the aspects discussed in Chapter 5.

## 1.4 THE LONG JOURNEY

Industry 4.0 is a long journey that must be travelled with the help of different *enabling technologies* that synergically cooperate in order to achieve the different objectives proposed in the strategic plan.

The following chapters summarize one little step in the search of this synergy. Methodologies that will appear as orthogonal and uncorrelated will be employed in an orchestrated manner, in particular to facilitate the relation between milling machines in shop floor and operators.

# 2 The ARTool Framework

*The chapter presents an overview of the ARTool platform, underlining the assets that are included in the framework, describing different devices required for the information flow, and discussing implementations. A showcase presents some of the demo applications designed to exhibit the prominent capabilities of the framework.*
*The second part of the chapter gives details about the application of ARTool as an input device for programming touch-probe movements. Procedures are detailed with specific algorithms for part-program generation. Operators may select directly the geometries to identify and obtain on the fly a simulation of the machine movements that can be inspected from different point-of-view. Eventually, the generated probe trajectory is loaded in the machine CNC for actual execution.*

## 2.1 AN OVERVIEW OF THE PLATFORM

The ARTool Framework is conceived to support machine manufacturers, technical offices, and machine operators in bringing augmented reality information on the machine and in the production lines.

The main objective of the framework is the optimization of the machining processes by tackling two major shortfalls:

- reducing the unproductive time between production batches, allowing the operators to quickly test the newer part-program

and to correct possible misalignment of blank material with respect to to reference systems saved in numerical control;

- supporting the maintenance procedures through the introduction of augmented manuals that facilitate remote assistance and technical support. Failure diagnosis can be improved highlighting failing components directly on machine chassis.

### 2.1.1 *From Authors to Consumers: the Flow of Data*

The main source of information are the technical offices, that provide tasks to shop-floor. Tasks data include:

- part-programs;
- fixtures list and fixture sequences;
- tooling information.

The technical office stores the authored data in SCADA (Supervisory Control and Data Acquisition) servers: this permits the centralization and distribution to data consumers.

The second authoring agent of the network is the machine manufacturer that through a Content Delivery Network (CDN) distributes assets for augmented manuals that the different SCADA servers of the different industries that acquired the machine download. The SCADA server act as a gateway for delivering updated assets to local machine and shop-floor operators. Optionally, manufacturers can exploit the framework for marketing opportunities. Machine manufacturers shall use the CDN also to deploy up-to-date versions of machine manuals, run a web store for spare parts, and provide a ticketing web interface to allow users to request advanced technical support. In such a way, ARTool opens new communication channels between users and vendors, and guarantees new marketing opportunities for machine producers.

For both technical offices and machine manufacturers, tools for authoring information are developed as plugins for commercially available Computer Aided Engineering (CAE) software [79, 66]. For technical offices, this means to expand the capabilities of common Computer Aided Manufacturing software, while, for manufacturers, the plugins are related to Computer Aided Design (CAD) and Product Life-cycle Management (PLM) software.

The main information consumers are the machine tool and the operator device. Both consumers fetch data from SCADA servers. The computer numerical control (CNC) communicates using a client that can be a software service, for newer machines, or an embedded computer, for older machines. The client requires an implementation of the proprietary communication protocol of the machine, while the communication with the SCADA is performed through standard protocols. The client broadcasts to the SCADA server all relevant information for diagnostic and simulation purposes, such as system states, tools table, etc.

Machine operators carry a personal device that has the hardware necessary to perform the ego-localization task—i.e. camera and inertial sensors—that is the most prominent feature of the ARTool framework. Currently, ARTool has been tested only on tablet devices, which are relatively low-cost and reliable with respect to other solutions.

### 2.1.2 *Operator device*

Operators are equipped with personal devices that have the minimum hardware requirements to perform the ego-localization. The current release of ARTool framework requires an high definition camera for gathering the scene on which assets are overlaid, an inertial measurement unit to filter the ego-localization state and a GPU for rendering the virtual scene.

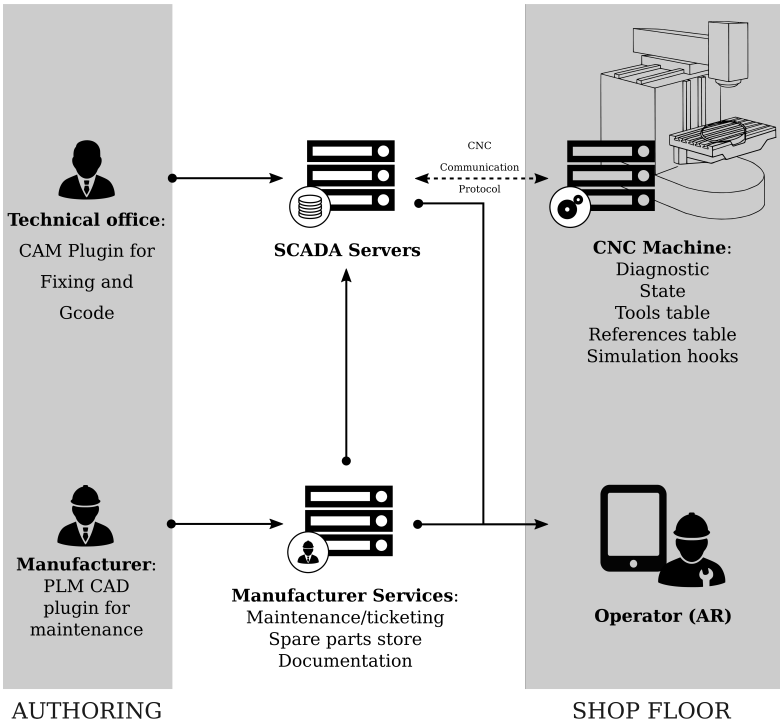



**Figure 2.1:** The ARTool flow of information, from technical offices and manufacturer, to machines and operators

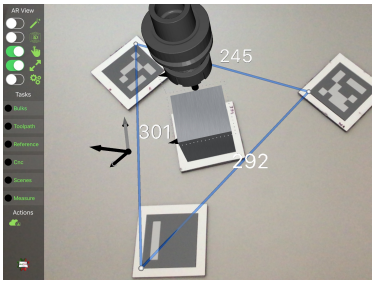



**Figure 2.2:** Screenshot of a very first prototype ARTool iPad app, showing the setup-mode augmented reality view. In this case, marker distances are measured. Camera images are localized in the working area: the application shows a bulk, a trajectory and a tool oriented with machine reference frames (From [MR11] with permissions)

Localization is performed through markers that characterize a scene (see Section 2.3 for a description of *scene* in detail). Once a scene is identified, a query to the SCADA server permits to populate the camera feed with virtual assets.

The framework eases the presentation of different information, that are contextualized with respect to a scene and a *operation mode*, or scope. When the current scope is to setup a new process for a machine, the main assets considered are:

- blank material and possibly the fixing for the bulk;
- tool and optionally machine head;
- mechanical axes simulacra;
- coordinate systems and oriented trajectory;
- marker anchoring elements (see Section 2.4);
- auxiliary descriptive text.

When the intended scope is maintenance, the framework is designed to stage:

- machine contextual information;
- masks over components of the machine;
- contextual manual web pages;
- geometric primitive shapes—e.g. arrows—that can be used to draw operator attention.

The device selected as prototype is an Apple iPad 2 Air Tablet, with iOS 9.3 operating system. The framework is a `C++` library that exposes `Swift` and `Objective-C` bindings. The rendering operation are handled by the Apple Framework `SceneKit` [4].

### 2.1.3 *The SCADA and per-machine server*

The SCADA server is responsible for storage and distribution of augmented assets. It also challenges machine clients for information necessary to present simulation and localized elements:

- the current state of machine, that includes the current position of axes, the active coordinate system, the loaded tool on the spindle and the active part-program;
- part-program simulation hooks, that comes from the numerical control parser/interpolator. If this information is available, ARTool shows the exact tool trajectory as interpolated by the numerical controller. If this information is not actually available, the framework exposes a fallback interpolator, that will generates trajectory with minimal differences;
- coordinate systems table and tool table. The tool table relates the currently loaded tool with a solid model counterpart for rendering. The reference systems table permits to project machine simulacra within the AR view, alongside the correct origins;

- optionally, diagnostic information that guides inexperienced operators in unusual situation and training.

In the experimental system, the server is a Ruby and C++ software on a separated machine, with database composed by a sequence of YAML files—i.e. a format that simplifies inspection and debugging. The server provides a HTML5 web application for authoring, which exploits the C++ component of ARTool framework for the creation of scenes from static images.

## 2.2 APPLICATIONS SHOWCASE

The section presents a series of applications designed to exhibit the prominent capabilities and features of the ARTool framework, leaving aside the authoring tools for machine manufacturer and technical offices.

### 2.2.1 *Origin Debugger*

The application visualizes the origins and the coordinate systems of both markers and numerical controller. The machine client is connected to the Heidenhain iTNC 530 of a Deckel Mori DMU-60T (5-axis milling machine), and takes advantages of an FTP connection for data exchange. From the FTP, the machine client downloads the iTNC file that stores the reference table. The file is queried at constant interval and parsed only if modification time changes.

The application permits to see selected origins of the table projected on the screen, overlaid on the frame captured by the camera. Operators can inspect the scene from different orientations. The application shows also distances between origins for debugging (see Fig. 2.3).

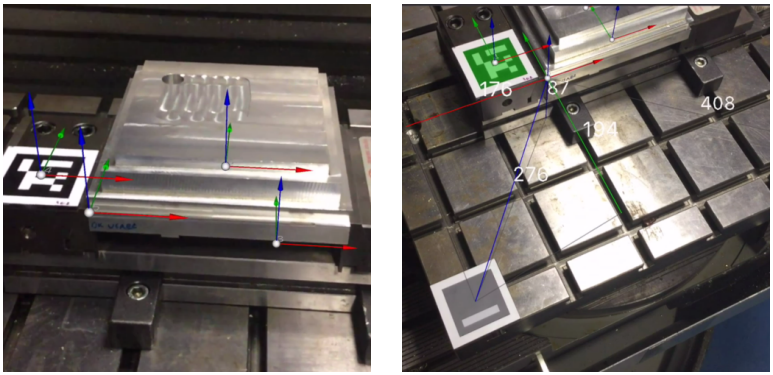



**Figure 2.3:** The Origin Debugger application: on the left, the visualization of the reference frame obtained from the machine client, while on the right a measurement between different origin is performed

### 2.2.2 *Trajectory Inspector*

The application is built upon the capabilities of the previous application. The machine client queries the controller for the currently active part-program and download it through the FTP connection, alongside with origin and tool table.

The tool table is parsed, and the name is used as identifier for the digital model to render, distributed through SCADA server. Since there is no communication channel for the numerical interpolator of this particular machine, it is the fallback ARTool interpolator that parses the part-program source file and generates the tool trajectory for the simulations.

Simulations are projected in a virtual environment that can be navigated, exactly like a common CAE environment. It is also possible to fix the virtual environment through a marker and explore the simulation by moving and reorienting the tablet, as depicted in the screenshot of Fig. 2.4.

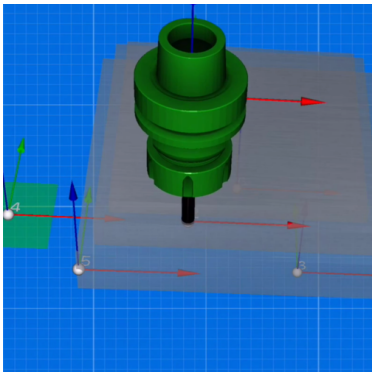


**Figure 2.4:** The Trajectory Inspector: operator can navigate the virtual enviroment or fix it through a marker

### 2.2.3 *Trajectory Simulator*

This application acts exactly like the Trajectory Inspector, and uses machine client and SCADA server to collect data and generate a virtually simulated environment that, in this case, is projected upon the camera feed. Operators can inspect directly the simulation in the working area, against real objects, the result of the interpolated trajectory and intercept collisions, programming errors, and misalignments (see Fig. 2.5).

### 2.2.4 *ARTool Zero*

ARTool Zero is the concept of an Augmented Reality application that allows operator to select directly some geometric features as reference through the touch-probe of a machine tool. Leveraging the input capabilities described in Section 2.4, the approximated feature information input through the augmented interface is transformed on-the-fly in a part-program that allows the touch-probe to precisely identify the geometry.

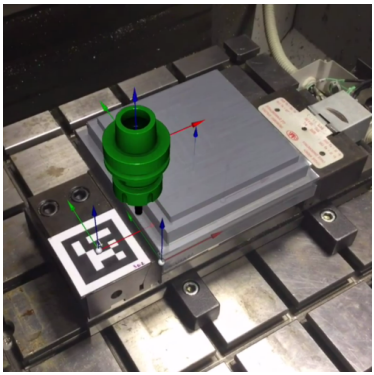


**Figure 2.5:** The Trajectory Simulator: operator can inspect the trajectory that is performed by the virtual tool, super-imposed on the working area

### 2.2.5 *Maintenance Mode*

The maintenance mode is at an early developing stage. The application requires a series of marker installed in the different parts of the machine to allow contextualized information gathering. In this case, the placement of assets on the screen does not require the same accuracy as in simulation, and a single marker covers quite a big area of the machine.

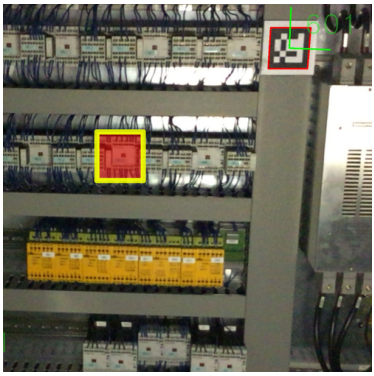


**Figure 2.6:** Maintenance Mode: a failed component highlighted

If a component fails the diagnostic, it is highlighted (see Fig. 2.6) and it is made evident to the operators. At the same time, an operator recall the manual page of a particular component by framing and taping it on the screen (using the input capabilities described in Section 2.4).

## 2.3 THE AR CORE OF ARTOOL

One of the critical requirements for an augmented application is a reliable and precise localization of the device with respect to the scene observed. The library `ARSceneDetector` is the software component that fulfill this task.

During the early development stage, the ARTool framework included the open source libray `ARUCO`, currently distributed with the OpenCV suite [16]. `ARUCO` is a localization library which takes advangtage of the presence of structured markers in scene for reconstruction. `ARUCO` was chosen after a comparison with the ArtoolKit platform [54]: it provided a better responsiveness at the cost of a lower accuracy, on the prototype device.

In a later development stage, in order to tackle the accuracy issues and to get a more stable and reliable localization through sensor fusion, the designed-from-scratch `ARSceneDetector` library has been introduced as core component of the ARTool framework. The library is strongly device-dependent (ARM-processor) and uses specific hardware instructions to speed-up its performances. This allows to squeeze the computational power of the device, attaining a precise and yet responsive placement of virtual assets on the framed scene. The next section describes the internal logical structure of `ARSceneDetector`.

### 2.3.1 `ARSceneDetector` *Library Details*

The `ARSceneDetector` library is logically divided in three different layers, from perception to scene rendering.

- The Sensor Acquisition and information gathering layer is written in `Swift` language. This is required by the platform and uses the current operating system API.
- The Marker Handler layer is written in `C++` and is linked to the OpenCV library. This layer handles the identification of the marker in the scene, the inter-frame tracking and the image stabilization.
- The very last layer is the Scene Detector, a classifier that extracts more information based upon the relative position of the marker in the scene.

The three layers are presented in Fig. 2.7.

As with other computer vision algorithms, `ARSceneDetector` requires a calibration of the camera [42] which results in a camera matrix. Light parameters and thresholds are automatically evaluated through normalization procedures: each frame is enhanced and the edge detection is extracted from the frame in GPU.

Using the internal camera of the prototype device, it is possible to collect frame with 720p and 1080p resolution. The bigger the frame, the lower the update frequency guaranteed for the localization—i.e. 120 Hz and 30 Hz respectively.

Beside the camera frame, accelerations and angular ratios of the device are measured by the on-board IMU sensor. This information permits to stabilize the rendered scene [11]. The combination of the frame and IMU data are passed through the bridge `Swift/C++` and enters the marker handler layer, as Scene container.

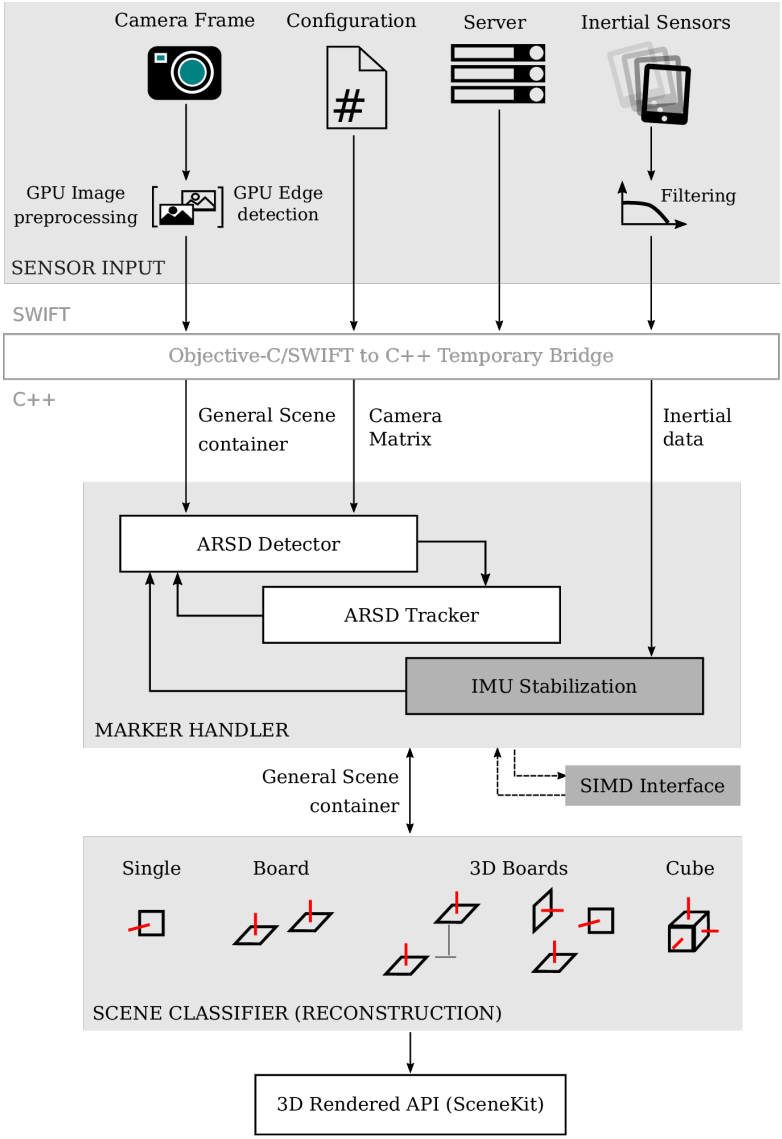



**Figure 2.7:** Library structure. ARSD stands for `ARSceneDetector`. In gray, plugins that are disabled during benchmark

The Marker Detector is the implementation of a classical one-frame-at-the-time algorithm which, for each camera frame, extracts convex quadrilateral shapes as marker candidates. The candidates are then reoriented and checked for squareness. The pose of each square element is reconstructed using different well-known algorithms [56, 71]. The algorithms return a reference system that is oriented through an asymmetrical pattern drawn on the marker itself. The pattern can be a number encoded in a binary form—e.g. the ARUCO encoding—or a image. The reference system is relative to the camera point-of-view and has always the $\hat{z}$ axis perpendicular to the marker surface.

The Tracker is an extension of the Detector algorithm that uses information of the previous frame to reduce the computational efforts of the Detector, limiting the area in which quadrilateral are searched, and lowering the frequency of whole-frame scanning (configurable, but with a default value of 10 frame). It can be disabled. To improve efficiency, Single Instruction Multiple Data (SIMD) instructions are employed.

The IMU Stabilization filters the state of the device, fusing the signal sampled by the IMU sensor.

The result of the Marker Handler is a General Scene Container, a data structure with all the information about identified marker and their position with respect to the device.

The very last layer of the library performs a classification of the General Scene Container. Using a combination of markers it is possible to drastically improve the accuracy of the localization. The possible scenes contain:

- a simple single marker;
- a board of co-planar markers, with parallel $\hat{z}$ axes;
- a board of markers, with parallel $\hat{z}$ axes, and known, non-zero offset in $\hat{z}$ direction;

- a board of three markers with mutually orthogonal $\hat{z}$ axes, with known offset vectors;
- a solid cube of markers.

The SCADA server provides the list of scenes to be classified. The Scene Detector matches the most similar one. Nevertheless, the library may enrich SCADA definitions: this particular feature is used for marker chaining which allows to expand the rendering volume, reaching area in which marker are not currently visible. Once the scene has been classified and reconstructed, the General Scene Container is shared with the render engine, that places the virtual models in a virtual world that is aligned with the perceived one.

## 2.4 ARTool as input

The first version of ARTool ships an augmented output-only interface for shop-floor user. Inputs come in as models localized in machine reference frame, prepared by machine manufacturers and technical personnel (e.g. CAD models, fixture elements, etc). With ARTool Zero, capabilities of the framework are expanded to reach the paradigm of input/output human-machine interface for CNC operators. The input allows to recognize points and geometric features in space, without adding further exotic hardware to the tablet device.

A screen of a mobile device allows to capture a bi-dimensional input. As already discussed, machine workspace is reconstructed via *markers*. Each marker defines a virtual *reference frame*. The $\hat{z}$ direction is the marker normal. Considering Fig. 2.8, which shows the *virtual plane* $\hat{x} \times \hat{y}$ that contains the origin, it is possible to cast through the camera matrix [42] a bi-dimensional screen coordinate to a tridimensional point that lies on the *virtual plane*. In other words, the 2-D screen point is the projection along the line of view on the *virtual plane*. Marker dimension is known, thus it is possible to perceive also

the *world scale*, and ascribe coordinates in metric units.

The procedure is explained in Fig. 2.8. Perspective projection matrices are notation for reference systems. One reference frame with index $j$, defined with respect to another frame with index $i$, namely ${}^{i}\mathbb{T}_{j}$, is composed by: (*a*) a rotational matrix ${}^{i}\overline{\mathbb{R}}_{j}$, which is $3 \times 3$, where each column represents orthogonal directions in space; (*b*) an origin point in homogeneous coordinates, ${}^{i}\mathbb{O}_{j}$, which is $4 \times 1$ where the first three elements are coordinates in space, and the last one represents an homogeneous scaling factor, that is always considered 1; (*c*) a vector of zeros $0_{1\times 3}$: when ${}^{i}\overline{\mathbb{R}}_{j}$ is concatenated with this vector, the notation ${}^{i}\mathbb{T}_{j}$ is used.

$$ {}^{i}\mathbb{T}_{j} = \begin{pmatrix} {}^{i}\overline{\mathbb{R}}_{j} & {}^{i}\mathbb{O}_{j_{1\ldots3}} \\ 0_{1\times3} & 1 \end{pmatrix} = \begin{pmatrix} {}^{i}\mathbb{R}_{j} & {}^{i}\mathbb{O}_{j} \end{pmatrix} \tag{2.1} $$

The inverse transformation is:

$$ \left({}^{i}\mathbb{T}_{j}\right)^{-1} = \begin{pmatrix} \left({}^{i}\overline{\mathbb{R}}_{j}\right)^{\top} & -\left({}^{i}\overline{\mathbb{R}}_{j}\right)^{\top} {}^{i}\mathbb{O}_{j_{1\ldots3}} \\ 0_{1\times3} & 1 \end{pmatrix} \tag{2.2} $$

Each machine has an absolute coordinates system, known as *machine reference* $\mathbb{T}_{0} \in \mathbb{R}^{4\times4}$. Mathematically, this reference is an identity matrix $\mathbb{I}_{4}$. Commercial CNCs save transformation of coordinate from the machine coordinate system in a reference table. A point $p_{r} \in \mathbb{R}^{4\times1}$, described in a reference at the index $r$ of the table $\left({}^{0}\mathbb{T}_{r}\right)$ is reported internally in machine coordinate with the relation:

$$ p_{0} = \mathbb{T}_{0}{}^{0}\mathbb{T}_{r}p_{r} = \mathbb{T}_{r}p_{r} \tag{2.3} $$

The *active reference*, also known as *part reference*, which is the reference currently selected on the CNC, allows to specify a part program with coordinates relative to a position and an orientation of the part, which is the reason why alignment procedures are fundamental.

An ARTool-ready machine has a *fixed machine marker* $A$ that is associated with a known coordinate transformation ${}^{0}\mathbb{T}_{A}$. The marker in

${}^0\mathbb{T}_A$ is used by ARTool library to ego-localize the mobile device. User fixes further *free markers*—e.g. $B$—in working-area: ARTool closes the chain between *machine active reference* and the marker reference, passing through the *fixed marker* reference:

$$ {}^r\mathbb{T}_B = {}^r\mathbb{T}_0 {}^0\mathbb{T}_A {}^A\mathbb{T}_B \tag{2.4} $$

The procedure is also known as *anchoring*: it allows ARTool to save the transformation between free and fixed marker, when both are framed.

When the user taps the mobile screen, using the camera matrix $C_B$ and the previous transformation, the 2D coordinates of the tap on the screen $p_{\text{tap}}$ are transformed in the coordinates of a 3D point projected on the plane of $B$, $p_B$:

$$ p_B = C_B p_{\text{tap}} \tag{2.5} $$

where $p_B \cdot \hat{z} = 0$. There is no need to keep both *machine marker* and *free*

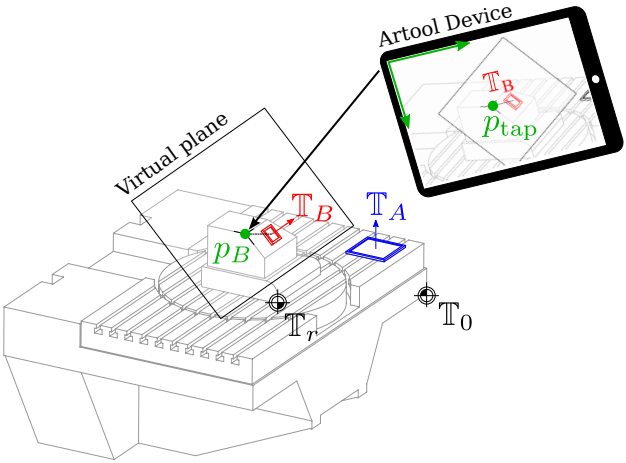



**Figure 2.8:** Using the mobile device as a 3D input system, through a mobile marker

*marker* framed at all times: indeed, once the free marker is positioned

and *anchored*, it can be used as a machine marker. This allows to create *chains of anchored markers*, extending the volume of view in which it is possible to input a position, although the accuracy of the ego-localization decreases exponentially at each chain hop.

The input procedure just depicted is enough to interpret basic geometric features and to perform alignments. The camera feed cannot guarantee the precision required in manufacturing technology—i.e. ARTool showed a repeatability in the order of $\pm$1mm—, but the perceived space is precise enough to maneuver an electronic touch-probe, which collects more precise measurements. This entails that a *client* with part program generation capabilities is connected to the machine.

### 2.4.1 *Client interface*

The client software communicates with the computer CNC, exposing the table reference frames, describing actual end-effector machine coordinates, and commanding the execution of preparatory code. Unfortunately, each machine has a different part program flavor (Siemens, Heidenhain, FANUC, FIDIA, etc.) and different touch-probe preparatory blocks (Heidenhain, Renishaw, etc.), thus the client must abstract an intermediate post processor: this approach is not different from what is currently done by commercial Computer Aided Manufacturing (CAM) softwares.

For the sake of the argument, the following routines are assumed to be abstracted for the communication between client and NC:

**`safe()`** the routine allows to bring the machine end effector in a safe position. For a milling machine, it usually maximize the $\hat{z}$ coordinate in the *machine reference*.

**`goto(`**$\mathbb{T}$**`,`** $p$**`)`** the routine performs an interpolated movement of the end effector. Takes as input a *reference frame* $\mathbb{T}$ and an arrival point $p$

that lays in the same reference; the procedure projects the arrival point in the active reference and than executes a `G01` preparatory block, at maximum feed.

**`getFrame(`**$i$**`)`** the routines reads reference matrix with respect to machine reference, which stored at the index $i$ in the reference table. With no arguments, returns the active reference frame.

**`probe(`**$\mathbb{T}$**`,`** $p$**`,`** $\alpha$**`)`** commands the touch-probe to find a contact point on the line that connects the actual position of the probe and the point $p$ specified in the reference $\mathbb{T}$; the argument $\alpha$ commands the retraction distance, which is the distance that must be traveled back after contact, along the line that connects starting point and contact point. The routines takes into account the uncertainty that surrounds the contact point. This routines calls a preparatory code that usually raises an error on the NC when contact is not achieved.

**`align(`**$\mathbb{T}$**`)`** aligns the $\hat{z}$ of the end effector with $\hat{z}$ of the argument frame, using interpolated movements. Align always performs a `safe()` before movements.

Each probe manufacturer defines a set of proprietary preparatory codes that perform inspection of different geometric features. This forces the implementation of a series of basic procedures that rely on a single interface common to all manufacturers. From these basic functions, complex procedures to perform alignment are derived.

Furthermore, the code generated by ARTool zero may also be optimized, to minimize the dwell movements between consequent touches. The optimization is actually a post-processing procedure and it is described in Chapter 3.

## 2.5 ALIGNMENT PROCEDURE

### 2.5.1 *Features description*

Operators use more than one feature to perform the alignment, and these can differ in shape and size. For this reason, several parametric procedures are developed, which are able to measure primitive entities, and are combined to estimate complex geometric features.

From a geometrical point of view the most interesting features to define a coordinate system in space are:

- simple touch
- line
- plane normal
- inner and outer
- vertex
- sphere

A graphical example of the features is depicted in Fig. 2.9.

The *simple contact* is the very basic feature that represents a contact between the touch-probe and the workpiece, while the probe moves along a direction in space, in a predefined system of coordinates. This procedure is a corner stone for all other routines, while being fundamental for users during identification and alignment of free-form geometries.

The *line* is identified by performing two *simple contacts* on one face of the workpiece, and it is typical for 2.5D machines. From the joining of the two touches it is easy to reckon slope of edges with respect to an arbitrary system of coordinates.

The *plane normal* is geometrically identified through three *simple contacts*. This procedure is actually the combination of two *edges*.

*Inner and outer circles* are in many situations good alignment features, and are associated with a plane normal that determines an origin offset, where the identified circle lays. For *inner circles*, origin is typically placed on the aperture, while for the *outer ones* it is located on top of the extruded material, to avoid collisions.

*Corners* are the most used feature in alignment for a prismatic workpiece: edges can be aligned with machine axis, with three faces representing the zero of each axis. The procedure to identify a *corner* expects at least three *simple contacts* that define an origin of the coordinate system.

The last feature is the *sphere*. In several cases this feature can be useful in the identification of a space, while neglecting its orientation. The sphere center is identified by the solution of a minimization problem.

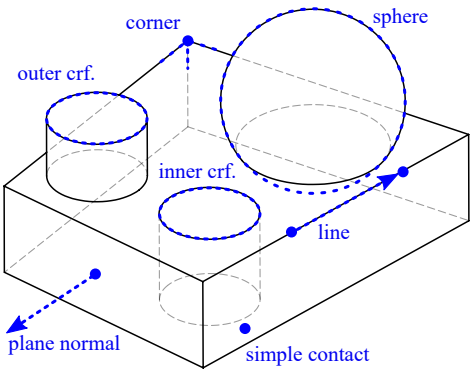



**Figure 2.9:** The different feature list on the model of a sample bulk, with all the features described in this work

### 2.5.2 *Procedure*

When a feature marker appears in field of view of tablet camera, ARTool Zero detects the type of feature, and project a *localization mask* which is specific for that feature, on the marker reference frame. User can translate, rotate and scale the mask to nearly fit the real feature. Each mask is associated with a reference frame and a parameter $\sigma$ that represents the scale of the mask. The AR interface commands the execution of a generated part-program, when user confirm the mask position. Alignments are defined through procedures that enforce robustness with respect to misalignments and and collisions. It must be clear that ARTool Zero is an interface only, and even if procedure definitions try to reduce mistakes and impacts, **user has always to check through the augmented simulation if probe can reach contact points, while avoiding collisions**. These procedure are constructed with respect to the coordinate frame of the marker, where marker normal is assumed to be on the $\hat{z}$ axis.

#### Simple contact

The *simple contact* feature is the very basic one, and it is employed in all other procedures, thus it has to be complete and versatile.

When user frames the marker related to simple contact procedure, a mask shaped like a little circle is shown on display, that is associated with a reference frame ${}^{B}\mathbb{T}_{\mu}$. It is only possible to translate the mask on the plane $\hat{x} \times \hat{y}$ of the marker, thus the origin of the mask frame has third component equal to zero.

Internally, the procedure, transforms the mask reference origin in a contact point to be reached by touch-probe:

$$v = \mathbb{T}_r{}^{r}\mathbb{T}_B{}^{B}\mathbb{O}_{\mu} \tag{2.6}$$

that in Fig. 2.10 is the red dashed vector. The actual trajectory of the probe is the vector $v - w$ and the parameter $\delta$ specify the so called retraction. The final position of the probe is:

$$p = (v - w) - \delta \frac{v - w}{|v - w|} \tag{2.7}$$

The reconstructed reference frame, has the same orientation of the active frame, and it is translated in the contact point $v$

$$^{r}\mathbb{T}_{k} = \begin{pmatrix} ^{r}\mathbb{R} & \mathbb{T}_{r}v \end{pmatrix} \tag{2.8}$$

When $\delta$ is not specified, the probe is considered to retract in its initial position. There is no constraints on the input $\mathbb{T}$, and **internally** it is possible to set a combination of $\mathbb{T}$ and $p$ that has an offset with respect to marker plane. The procedure is described in the following function. The returned frame is always with respect to machine reference.

---

**function** SIMPLECONTACT($\mathbb{T}$, $p$, $\delta$)
  **const** $k$ ▷ Constant number for last touched point
  `probe(`$\mathbb{T}$, $p$, $\delta$`)`
  **return** ← `getFrame(`$k$`)`
**end function**

---

### Line

Line feature is a peculiar case of alignments, mainly introduced for 2.5D machines, where the angle between one straight face of the workpiece and a machine axis has to be compensated. There is a contact of two points, in such a way they can approximate a line that belongs to a face of the workpiece. The mask has the shape of a vec-

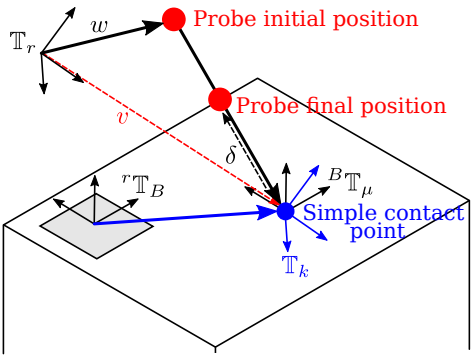



**Figure 2.10:** The simple contact procedure, with probe approaching and retracting trajectory

tor. The origin of the mask reference frame is in the application point of the vector. The contact procedure is shown in Fig. 2.11. The mask should be approximately aligned on one edge-line of the face of the workpiece, that is the actual feature to be reconstructed. Users move, rotate and scale the mask in the $\hat{x} \times \hat{y}$ plane. While the rotation specifies the orientation of the vector, the scale parameter $\sigma$ specify the length, adjustable through pinch-to-zoom. Points to be touched with two simple contacts are:

$$\begin{aligned} v_1 &= \mathbb{T}_r {}^r\mathbb{T}_B {}^B\mathbb{T}_\mu(\delta_1 \hat{x}) \\ v_2 &= \mathbb{T}_r {}^r\mathbb{T}_B {}^B\mathbb{T}_\mu(\delta_1 \hat{x} + \sigma \hat{y}) \end{aligned} \tag{2.9}$$

where $\delta_1$ is a offset imposed for safety reasons that prevents the probe from missing the workpiece. The $\delta_2$ offset sets the clearance between the probe tip and the workpiece surface, that determines the two dwell positions:

$$\begin{aligned} w_1 &= \mathbb{T}_r {}^r\mathbb{T}_B {}^B\mathbb{T}_\mu(\delta_1 \hat{x} + \delta_2 \hat{z}) \\ w_2 &= \mathbb{T}_r {}^r\mathbb{T}_B {}^B\mathbb{T}_\mu(\delta_1 \hat{x} + \sigma \hat{y} + \delta_2 \hat{z}) \end{aligned} \tag{2.10}$$

Both parameters are modifiable through slider elements. The procedure reaches point $w_1$ and performs a simple contact in $v_1$ with a

complete retraction, then moves to $w_2$ and performs a second simple contact in $v_2$. The reconstructed reference frame has $\hat{x}$ axis alongside touched points, $\hat{y}$ axis is the skew symmetric of the normal of the marker reference and the $\hat{x}$. Last axis, $\hat{z}$ is the skew symmetric between $\hat{x}$ and $\hat{y}$[1]. The sequence of operation are summarized in the following function:

---

**function** LINECONTACT($^B\mathbb{T}_\mu$, $\sigma$, $\delta_1$, $\delta_2$)
  **for** $\xi \leftarrow [\delta_1\hat{x}\,,\ \delta_1\hat{x} + \sigma\hat{y}]$ **do**
    `goto(`$^B\mathbb{T}_\mu$, $\xi + \delta_2\hat{z}$`)`
    $^0\mathbb{T}_i \leftarrow$ `simpleContact(`$^B\mathbb{T}_\mu$,$\xi$`)`
  **end for**
  $\hat{x} \leftarrow \dfrac{^0\mathbb{O}_2 - {}^0\mathbb{O}_1}{\|{}^0\mathbb{O}_2 - {}^0\mathbb{O}_1\|}$
  $\hat{y} \leftarrow \left[\left[{}^\mu\mathbb{T}_B\right]_{1\ldots4,3}\right]_\times \hat{x}$ ▷ $[\cdot]_\times$ is the skew operator
  $\hat{z} \leftarrow [\hat{x}]_\times\, \hat{y}$
  **return** $\begin{pmatrix}\hat{x} & \hat{y} & \hat{z} & {}^0\mathbb{O}_1\end{pmatrix}$
**end function**

---

### Plane

To reconstruct a plane, the machine has to probe the workpiece three times. User places a triangular mask, that may be re-oriented and scaled ($\sigma$ factor). The assigned reference frame is centered in the centroid of the mask. The three points are in the corners of the mask, and it is simple to reconstruct their position, in the mask reference

[1]The skew symmetric operator is defined as:

$$[v]_\times = \begin{pmatrix} 0 & -v_3 & v_2 & 0 \\ v_3 & 0 & -v_1 & 0 \\ -v_2 & v_1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

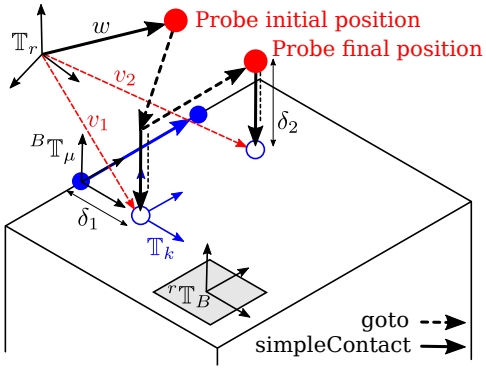



**Figure 2.11:** The edge contact procedure: the mask is in full blue, while actual contact points are in blue stroke and white fill

frame:

$$v_{\mu,j} = \sigma \cos\left(\frac{2\pi j}{3}\right)\hat{x} + \sigma \sin\left(\frac{2\pi j}{3}\right)\hat{y} \quad j = 0,1,2 \tag{2.11}$$

while the cleared positions are:

$$w_{\mu,j} = v_{\mu,j} + \delta\hat{z} \quad j = 0,1,2 \tag{2.12}$$

with $\delta$ parameter for dwell clearance. The vector are not referred in machine active reference: as before, it is possible to use the defined functions to reach them in machine reference. Once the three points have been reached, the reconstructed frame has origin:

$$b = \frac{1}{3}\sum_{k=0}^{2} v_k \tag{2.13}$$

The first orientations that are reconstructed are $\hat{x}$ and $\hat{z}$. The procedure can be configured to reach the secure position and to align the probe to be orthogonal to the mask. While the procedure is summarized in the following function, the sequence of movements is depicted in Fig. 2.12.

**function** PLANECONTACT($^B\mathbb{T}_\mu$, $\sigma$, $\delta$)

`align(`$^B\mathbb{T}_\mu$`)` **if** align? ▷ configured by user

`goto(`$^B\mathbb{T}_\mu$, $\delta\hat{z}$`)`

**for** $\xi \leftarrow [0, 1, 2]$ **do**

$$v_i = \sigma \cos\left(\frac{2\pi k}{3}\right)\hat{x} + \sigma \sin\left(\frac{2\pi k}{3}\right)\hat{y}$$

`goto(`$^B\mathbb{T}_\mu$, $v_i + \delta\hat{z}$`)`

$\mathbb{T}i \leftarrow$ `simpleContact(`$^B\mathbb{T}_\mu$,$v_i$`)`

**end for**

$$b \leftarrow \frac{1}{3}\sum_{i=0}^{2} \mathbb{O}_i$$

$$\hat{x} \leftarrow \frac{\mathbb{O}_0 - b}{\|\mathbb{O}_0 - b\|}$$

$$\hat{z} \leftarrow \left[\frac{\mathbb{O}_1 - \mathbb{O}_0}{\|\mathbb{O}_1 - \mathbb{O}_0\|}\right]_\times \frac{\mathbb{O}_2 - \mathbb{O}_1}{\|\mathbb{O}_2 - \mathbb{O}_1\|}$$

$$\hat{y} \leftarrow [\hat{z}]_\times \hat{x}$$

**return** $\begin{pmatrix}\hat{x} & \hat{y} & \hat{z} & b\end{pmatrix}$

**end function**

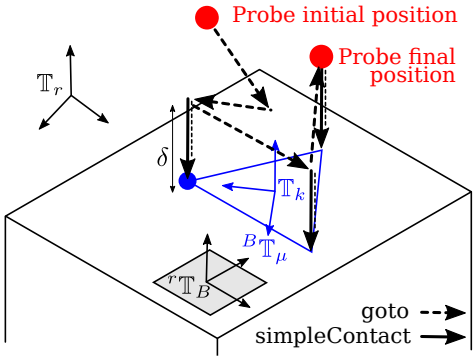



**Figure 2.12:** The plane contact procedure: the mask is in blue, and the first contact point is highlighted with the filled circle

### Inner and Outer circles

Circles are the feature in which AR interface shows its true expressiveness potential. Let's consider the first case of a inner circle identification. There are two masks that are consecutively placed on the workpiece surface: a *plane normal* mask and a *circle mask* that is specific for the procedure. The system engages the user in positioning a *plane normal* mask, since it is necessary to evaluate a precise axis of the circle in order to keep evaluation of circle consistent. *Plane normal* mask has reference ${}^{B}\mathbb{T}_{\mu_1}$

The *inner circle mask* is the slice of a full cylinder that superposes graphically the convex hull of the probe reached space during touches. The mask has orientation defined through two fingers gesture and it is scaled through pinch-to-zoom. Other parameters are input through sliders: the final angle for the sequence ($\pi/12 \leq \delta_1 \leq 2\pi$), the number of touches ($\delta_2 \geq 3$), and the depth at which the actual touch will be performed ($\delta_3 > 0\,\text{mm}$). This depth is the height of the visualized mask. The reference of the mask is ${}^{B}\mathbb{T}_{\mu_2}$ that is centered along the axis of the cylinder. With respect to this reference, the points to be touched are:

$$v_i = \sigma \cos\left(\frac{i\,\delta_1}{\delta_2}\right)\hat{x} + \sigma \sin\left(\frac{i\,\delta_1}{\delta_2}\right)\hat{y} - \delta_3\hat{z} \quad i = 0, \ldots, \delta_2 - 1 \tag{2.14}$$

Before reaching the dwell position, that is in the center of the reference frame identified by the mask, the procedure forces an `align(`${}^{B}\mathbb{T}_{\mu}$`)`.

Points collected by the touch-probe fulfill the equation of a sphere, that has center in $q$ and radius $\rho$

$$(v_i - q)^{\top}(v_i - q) = \rho^2 \tag{2.15}$$

It is possible to manipulate Eq. 2.15 to the following expression:

$$\underbrace{\begin{pmatrix} v_0^\top v_0 \\ \dots \\ v_{\delta_2-1} \end{pmatrix}}_{\Phi} = \underbrace{\begin{pmatrix} 2v_0^\top & 1 \\ \dots & \dots \\ 2v_{\delta_2-1}^\top & 1 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} q \\ \rho^2 - q^\top q \end{pmatrix}}_{Q} \tag{2.16}$$

Unknowns are contained in $Q$. If $A$ has full rank, the solution is obtained through a left pseudo-inverse, that is the solution of the associated minimum squares problem:

$$Q = \left(A^\top A\right)^{-1} A^\top \Phi \tag{2.17}$$

To guarantee the existence of a solution with sample collected on the same plane, the solution has to drop the $\hat{z}$ component of the formulation, that is useless in this particular case. The $\hat{z}$ component is inherited from the *plane procedure*. The modified minimization problem is identified by over-lined symbols ($\overline{\Phi}$, $\overline{Q}$, $\overline{A}$). Even if radius is reconstructed, it is currently dropped, since the CNC interface does expose a unified method for saving variables.

The returned reference frame has the same orientation of *plane contact*, and the frame is centered at the opening of the inner circle. The procedure can be summarized as follows:

---

**function** ICCONTACT(${}^B\mathbb{T}_{\mu_1}$, ${}^B\mathbb{T}_{\mu_2}$, $\sigma_1$, $\sigma_2$, $\delta_0$, $\delta_1$, $\delta_2$, $\delta_3$)
  `align(`${}^B\mathbb{T}_{\mu_2}`)`
  ${}^0\mathbb{T}_z \leftarrow$ `planeContact(`${}^B\mathbb{T}_{\mu_1}$, $\sigma_1$, $\delta_0$`)`
  `goto(`${}^B\mathbb{T}_{\mu_2}$, $\delta_0 \hat{z}$`)`
  `goto(`${}^B\mathbb{T}_{\mu_2}$, $0$`)`
  **for** $\xi \leftarrow [0, \dots, \delta_2 - 1]$ **do**
    $v_i = \sigma \cos\left(\frac{i\,\delta_1}{\delta_2}\right)\hat{x} + \sigma \sin\left(\frac{i\,\delta_1}{\delta_2}\right)\hat{y} - \delta_3 \hat{z}$
    $\mathbb{T}_i \leftarrow \left({}^0\mathbb{T}_z\right)^{-1}$ `simpleContact(`${}^B\mathbb{T}_{\mu}$,$v_i$`)`

**end for**
`goto(`${}^{B}\mathbb{T}_{\mu_2}, \delta_0\,\hat{z}$`)`
$\overline{\Phi} \leftarrow \overline{\Phi}\left(\mathbb{O}_0, \ldots, \mathbb{O}_{(\delta_2-1)}\right)$ ▷ see Eq. 2.17
$\overline{A} \leftarrow \overline{A}\left(\mathbb{O}_0, \ldots, \mathbb{O}_{(\delta_2-1)}\right)$
$\left[q, \rho^2 - q^\top q\right]^\top \leftarrow \left(\overline{A}^\top \overline{A}\right)^{-1} \overline{A}^\top \overline{\Phi}$
**return** $\left({}^{0}\overline{\mathbb{R}}_z \quad \left(q\,\hat{x} + q\,\hat{y} + {}^{0}\mathrm{O}_z\,\hat{z}\right)\right)$
**end function**

---

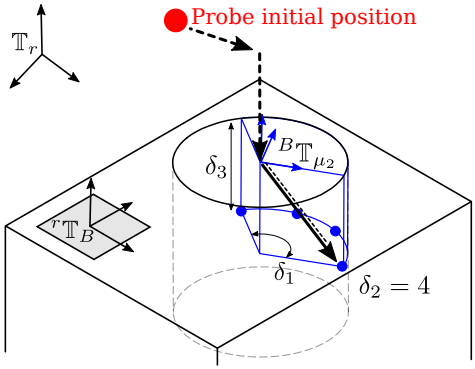



**Figure 2.13:** The inner circle procedure, over $\delta_2 = 4$ contact points. For clarity, only the movements for the contact with the first point are reported

For outer circle, the second mask is a slice of a cave cylinder, that represents the convex hull of the space reached by the touch-probe. The inner radius of this mask should be on the later surface of the geometric feature to be measured. With respect to the previous situation, a new parameter is necessary, that is the thickness of the cylinder on the mask. The procedure details are not reported for the sake of brevity, while the mask parameters are reported in Fig. 2.14.

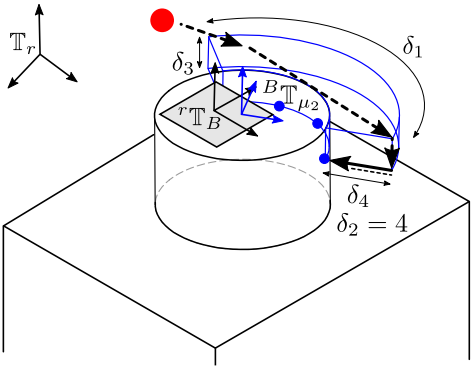



**Figure 2.14:** The outer circle procedure, over $\delta_2 = 4$ contact points. For clarity, only the movements for the contact with the first point are reported. The marker is positioned on the top of the feature

### Corner

Corners are identified through three simple contacts for orthogonal faces. The procedure shows mask that has a cubic shape. The cube has one corner highlighted, that is the target corner. The mask reference is centered in selected corner. The reconstructed reference has the origin in the locus of intersection of the three planes of the active reference, which is the reason why this procedure has to be performed *after* a plane and a line procedure, for use with general orientation. Double tap on the mask changes the highlighted target corner. This changes the approach sequence in the procedure.

The mask dimensions show the extremes at which the touch probe will approach the material to perform contacts. The depth of cube extends below the marker plane.

The procedure has two parameters: $\delta_1 > 0$ that is the dwell position of the probe above the material in each direction, and $\delta_2 \in [0, \ldots, 3]$ that represents the corner to be touched—i.e. the corner of the front of the cubic mask. The effect of $\delta_2$ is to apply a rotation on all points

declared, of $\delta_2 \pi/4$, as pictured in Fig. 2.15. The reference of the mask has origin placed in the center of the corner selected.

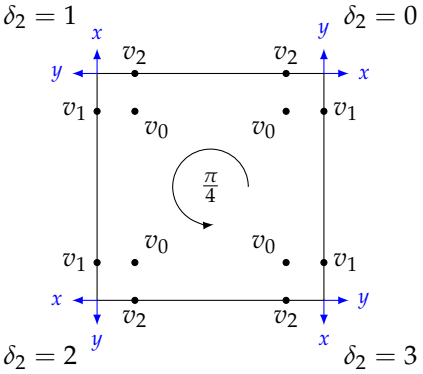



**Figure 2.15:** The touching point schemes derived from parameter $\delta_2$, as a discrete rotation around $\hat{z}$ axis of $\pi/4$ steps, looking at the front face of the cube mask

The procedure function is not reported for the sake of brevity. Fig. 2.16 reports the graphical representation of the procedure.

### Sphere

The mask of the sphere procedure is a slice of an hemisphere. The center of the hemisphere is identified through the minimum square problem expressed in eq. 2.17.

The procedure has several parameters, that determines the grid of contact points of the procedure. The resulting system of coordinate inherits the same orientation of the currently active reference, while the origin is the centre of the hemisphere.

The procedure has several parameter, shown in Fig. 2.18:

- the polar initial angle of the sphere: $0 < \delta_1 < \pi/2$,
- the polar span of the sphere: $0 < \delta_2 < \pi/2 - \delta_1$,

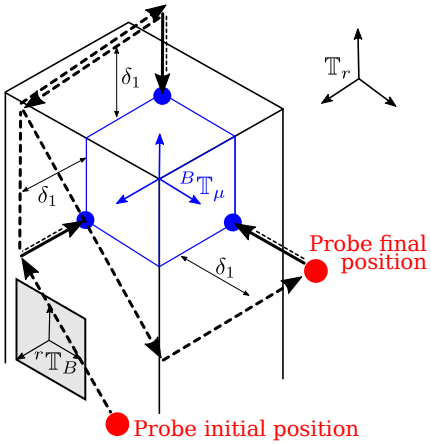



**Figure 2.16:** The complete sequence of movements for the corner contact procedure

- the azimuthal initial angle: $0 < \delta_3 < 2\pi$,
- the azimuthal span angle: $0 < \delta_4 < 2\pi - \delta_3$,
- the subdivisions parameter: $\delta_5 \geq 0$,
- the $\hat{z}$ elevation of the hemisphere: $\delta_6 \geq 0$,
- the dwell position for the probe: $\delta_7 \geq 0$.

Using the maximum possible values for $\delta_2$ and $\delta_4$ the mask is a full hemisphere.

There are two additional procedures used inside the sphere contact. The first one evaluates the subdivisions for the angles to be reached, as in Fig. 2.17. The latter is the moving strategy, that emulates a circular interpolation. The strategy uses simple `goto` calls, in such a way the probe can move over the hemisphere surface without colliding.

---

**function** SPHERECONTACT(${}^B\mathbb{T}_\mu$, $\sigma$, $\delta_1$, $\delta_2$, $\delta_3$, $\delta_4$, $\delta_5$, $\delta_6$, $\delta_7$)

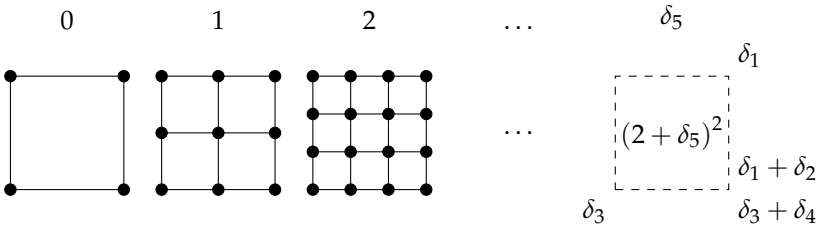



**Figure 2.17:** The subdivision scheme for sphere contact procedure. The subdivision is used to create the moving strategy

`align(`${}^B\mathbb{T}_\mu$`)`
**for** $i, \theta_i, \psi_i \leftarrow$ `subdivision(`$\delta_1, \delta_2, \delta_3, \delta_4, \delta_5$`)` **do**
$v_i \leftarrow \sigma\left(\sin(\theta_i)\cos(\psi_i)\hat{x} + \sin(\theta_i)\sin(\psi_i)\hat{y} + \cos(\theta_i)\hat{z}\right) + \delta_6\hat{z}$
$w_i \leftarrow v_i + \delta_7\left(\sin(\theta_i)\cos(\psi_i)\hat{x} + \sin(\theta_i)\sin(\psi_i)\hat{y} + \cos(\theta_i)\hat{z}\right)$
`moveStrategy(`$w_i$`)`
${}^B\mathbb{T}_i \leftarrow$ `simpleContact(`${}^B\mathbb{T}_\mu, v_i$`)`
**end for**
$\Phi \leftarrow \Phi\left(\mathbb{O}_0, \ldots, \mathbb{O}_{(2+\delta_5)^2}\right)$ ▷ see Eq. 2.17
$A \leftarrow A\left(\mathbb{O}_0, \ldots, \mathbb{O}_{(2+\delta_5)^2}\right)$
$\left[q, \rho^2 - q^\top q\right]^\top \leftarrow \left(A^\top A\right)^{-1} A^\top \Phi$
${}^0\mathbb{T}_r \leftarrow$ `getFrame()`
**return** $\begin{pmatrix} {}^0\mathbb{R}_r & (q,\,1)^\top \end{pmatrix}$
**end function**

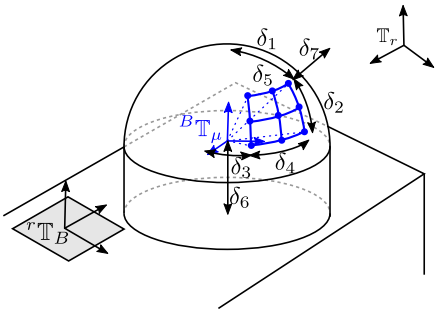



**Figure 2.18:** The parameters for the sphere contact procedures

# 3 Pre-processing Optimization

*The chapter is divided in two sections. The first part presents the formulation of the minimum time optimal control problem for the interpolation of axes motion for a numerically controlled machine. The proposed solution exploits a lateral tolerance to produce a trajectory that crosses an intersection point while pushing the machine to its dynamical limits, using the constrained jerk as a control variable. The second part introduces Mr.CAS, a computer algebra library aimed at rapid prototyping and code generation, written in purely Ruby language.*

## 3.1 Optimal control formulation

The Section recalls the theory behind the optimal control and shows how the optimized trajectory—i.e. the tool path together with the speed profile—can be used for preprocessing an original part program and generating a modified one, made by a sequence of short straight segments, whose track is a discretization of the optimized tool path, and whose feed rate are set to copy the optimized speed profile. Even if previous Chapters presented the method as an optimization for the dwell transfer between consequent touches in ARTool, the approach is general enough to be used in different contexts. In this theoretic introduction, the original notation is kept.

### 3.1.1 *Path Clusterization*

Before diving into the optimal control formulation, the part program must be parsed in order to be transformed in a nominal trajectory that can be included in the optimization routines. The geometry of the path is split in clusters of movements. The clustering approach allows to split a long path made of thousands of position commands in shorter sequences for which boundary conditions are known, without exceeding the computational limits of the machine that executes the optimization task.

The clusterization is performed following four heuristic rules.

1. A new cluster is initialized when a full-stop for all axes is required (e.g. switching from `G00` to `G01`). This rule sets to zero some of the boundary conditions of the problem.

2. A new cluster is initialized when a block defines a linear movement that is long enough to reach the desired feed rate in nominal conditions—i.e. the length of the segment is greater than a critical length defined as $l^\star = f_n^2/a^\star$, where $a^\star < a_{\max}$ is a target acceleration / deceleration. A *short* movement is not long enough to reach $l^\star$.

3. Collinear segments are squashed in the same cluster, unless one of the previous rules apply.

4. Circular arcs and short movements join the subsequent path.

Since the tolerance is expressed as a distance along the orthogonal direction with respect to nominal path, the final optimized path pass through discontinuity points in order to simplify the evaluation of tolerances (i.e. avoid ambiguity and construction of particular tolerance functions around discontinuity points, which may make the problem infinite-dimensional).

### 3.1.2 *Model of System Dynamics*

Taking as a reference the sketch in Fig. 3.1, when $s$ is the arc length, or curvilinear abscissa, the nominal trajectory can be described as a continuos function of $s$:

$$\boldsymbol{P}_n(s) = \begin{pmatrix} x_n(s) \\ y_n(s) \end{pmatrix} \qquad \text{for } 0 \leq s \leq L \tag{3.1}$$

with piecewise continuos derivative. The derivative of the nominal trajectory $\boldsymbol{P}_n(s)$ is discontinuous on a finite number of points, corresponding to arc lengths $0 < s_1 < s_2 < \cdots < s_{m-1} < L$, where $L$ is the total cluster length and $m$ is the number of blocks in the cluster. Being $\|\boldsymbol{P}'_n(s)\| = 1$, it is possible to define $\theta_n(s)$, the angle of the nominal trajectory, as the angle that satisfies:

$$\boldsymbol{P}'_n(s) \quad = \begin{pmatrix} x'_n(s) \\ y'_n(s) \end{pmatrix} = \begin{pmatrix} \cos\theta_n(s) \\ \sin\theta_n(s) \end{pmatrix}, \tag{3.2}$$

whenever $s \neq s_k$. Note that hereafter the prime notation is used for space derivative, while the dot notation is used for the time derivative. The directions tangent and normal to the nominal trajectory are, respectively:

$$\boldsymbol{T}(\theta_n) \quad = \begin{pmatrix} \cos\theta_n \\ \sin\theta_n \end{pmatrix}, \qquad \boldsymbol{N}(\theta_n) \quad = \begin{pmatrix} -\sin\theta_n \\ \cos\theta_n \end{pmatrix}. \tag{3.3}$$

On the basis of the vector $\boldsymbol{N}(\theta_n)$, and of the nominal trajectory $\boldsymbol{P}_n(s)$, one can define a curvilinear coordinate system, where a point $\boldsymbol{P}$ has the curvilinear coordinates $(s, n)$ when $\boldsymbol{P} \equiv \boldsymbol{P}_n(s) + n\boldsymbol{N}(\theta_n)$.

In this curvilinear coordinate system the tool center position can be

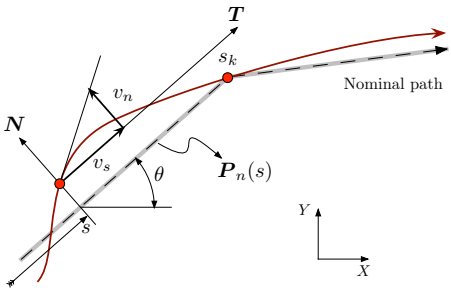



**Figure 3.1:** Local coordinate frame used for formulating the optimal control problem
(From [MR3] with permissions)

described as:

$$\boldsymbol{P}(t) = \boldsymbol{P}_n(s) + n\boldsymbol{N}(\theta_n), \qquad \begin{aligned} &\text{with} \quad s \equiv s(t) \\ &\text{and} \quad n \equiv n(t) \\ &\text{and} \quad \theta_n \equiv \theta_n(s(t)). \end{aligned} \tag{3.4}$$

Notice that, in order of having an univocal definition for $\boldsymbol{P}(t_k)$ and due to the possible discontinuity in the derivative of nominal trajectory at nodal points, the curvilinear coordinate $n(t)$ must be 0 for $t = t_k$, where $s(t_k) = s_k$. This implies that the tool center position is forced to cross the nominal trajectory at nodal points $\boldsymbol{P}(t_k)$. Along this path, the tool velocity $\boldsymbol{V}(t) = \dot{\boldsymbol{P}}(t)$ can thus be expressed as:

$$\boldsymbol{V}(t) = \boldsymbol{T}(\theta_n)(1 - n\kappa)\dot{s} + \boldsymbol{N}(\theta_n)\dot{n}, \quad \kappa(s) = \theta_n'(s) \tag{3.5}$$

where $\kappa(s)$ is the curvature of the nominal trajectory.

The projections of the velocity vector in the local curvilinear reference frame, according to Fig. 3.1, can be thus expressed as:

$$\begin{aligned} v_s(t) &= \boldsymbol{V}(t) \cdot \boldsymbol{T}(s) = (1 - n\kappa)\dot{s}, \\ v_n(t) &= \boldsymbol{V}(t) \cdot \boldsymbol{N}(s) = \dot{n}, \end{aligned} \tag{3.6}$$

and the velocity vector itself can be rewritten as:

$$\boldsymbol{V}(t) = \boldsymbol{T}(\theta_n)v_s + \boldsymbol{N}(\theta_n)v_n. \tag{3.7}$$

Analogously, the acceleration $\boldsymbol{A}(t) = \dot{\boldsymbol{V}}(t)$ can be expressed as:

$$\boldsymbol{A}(t) = \boldsymbol{T}(\theta_n)(\dot{v}_s - \kappa v_n \dot{s}) + \boldsymbol{N}(\theta_n)(\dot{v}_n + \kappa v_s \dot{s}), \tag{3.8}$$

and using again the local projections in the curvilinear reference frame:

$$\begin{aligned} a_s(t) &= \boldsymbol{A}(t) \cdot \boldsymbol{T}(s) = \dot{v}_s - \kappa v_n \dot{s}, \\ a_n(t) &= \boldsymbol{A}(t) \cdot \boldsymbol{N}(s) = \dot{v}_n + \kappa v_s \dot{s}, \\ \boldsymbol{A}(t) &= \boldsymbol{T}(\theta_n)a_s + \boldsymbol{N}(\theta_n)a_n. \end{aligned} \tag{3.9}$$

Finally, the jerk $\boldsymbol{J}(t) = \dot{\boldsymbol{A}}(t)$ can be expressed as:

$$\boldsymbol{J}(t) = \boldsymbol{T}(\theta_n)(\dot{a}_s - \kappa a_n \dot{s}) + \boldsymbol{N}(\theta_n)(\dot{a}_n + \kappa a_s \dot{s}), \tag{3.10}$$

and, by using again the local projections in the curvilinear reference frame:

$$\begin{aligned} j_s(t) &= \boldsymbol{J}(t) \cdot \boldsymbol{T}(s) = \dot{a}_s - \kappa v_s \dot{s}, \\ j_n(t) &= \boldsymbol{J}(t) \cdot \boldsymbol{N}(s) = \dot{a}_n + \kappa v_s \dot{s}, \\ \boldsymbol{J}(t) &= \boldsymbol{T}(\theta_n)j_s + \boldsymbol{N}(\theta_n)j_n \end{aligned} \tag{3.11}$$

By combining Eqs. 3.6, 3.9, and 3.11, one obtains the following system of ordinary differential equations (ODE):

$$\begin{aligned} \dot{s} &= v_s/(1 - \kappa n), & \dot{n} &= v_n, \\ \dot{v}_s &= a_s + \kappa v_n \dot{s}, & \dot{v}_n &= a_n - \kappa v_s \dot{s}, \\ \dot{a}_s &= j_s + \kappa a_n \dot{s}, & \dot{a}_n &= j_n - \kappa a_s \dot{s} \end{aligned} \tag{3.12}$$

whose solution represents the tool trajectory determined by the jerk history.

This ODE is valid wherever $s(t) \neq s_k$, i.e. except for the discontinuity points of the nominal trajectory. In correspondence with these discontinuity points the actual trajectory $\boldsymbol{P}(t)$ must be continuous:

$$\boldsymbol{P}(t_k^-) = \boldsymbol{P}(t_k^+), \quad \boldsymbol{V}(t_k^-) = \boldsymbol{V}(t_k^+), \quad \boldsymbol{A}(t_k^-) = \boldsymbol{A}(t_k^+) \tag{3.13}$$

where the superscripts $+$ and $-$ represent the quantities on the right (i.e. after) and on the left (i.e. before) side of a node, respectively.

As a consequence of the continuity equations and of Eq. 3.4, and by defining $\theta_k^{\pm} = \theta_n(s(t_k^{\pm}))$, it follows that:

$$\boldsymbol{P}_n(s_k) + n(t_k^-)\boldsymbol{N}(\theta_k^-) = \boldsymbol{P}_n(s_k) + n(t_k^+)\boldsymbol{N}(\theta_k^+), \tag{3.14}$$

which, unless $\boldsymbol{N}(\theta_k^-) = \boldsymbol{N}(\theta_k^+)$, implies that:

$$n(t_k^-) = n(t_k^+) = 0 \tag{3.15}$$

i.e. that the tool center passes exactly through the node, as shown in Fig. 3.1 (point $s_k$).

From Eq. 3.7, the continuity condition implies that

$$\begin{aligned} v_s(t_k^+) =& \boldsymbol{T}(\theta_k^+) \cdot \boldsymbol{T}(\theta_k^-) v_s(t_k^-) + \boldsymbol{T}(\theta_k^+) \cdot \boldsymbol{N}(\theta_k^-) v_n(t_k^-) \\ v_n(t_k^+) =& \boldsymbol{N}(\theta_k^+) \cdot \boldsymbol{T}(\theta_k^-) v_s(t_k^-) + \boldsymbol{N}(\theta_k^+) \cdot \boldsymbol{N}(\theta_k^-) v_n(t_k^-) \end{aligned} \tag{3.16}$$

The last pair of equations, by using the formulas

$$\begin{aligned} \cos(a-b) &= \cos(a)\cos(b) + \sin(a)\sin(b) \\ \sin(a-b) &= -\cos(a)\sin(b) + \sin(a)\cos(b) \end{aligned} \tag{3.17}$$

and by defining $\Delta\theta_k = \theta_k^+ - \theta_k^-$, can be finally expressed as:

$$\begin{aligned} v_s(t_k^+) &= v_s(t_k^-)\cos\Delta\theta_k + v_n(t_k^-)\sin\Delta\theta_k\,, \\ v_n(t_k^+) &= v_n(t_k^-)\cos\Delta\theta_k - v_s(t_k^-)\sin\Delta\theta_k\,. \end{aligned} \tag{3.18}$$

After analogous operations, from Eq. 3.9 one can obtain the corresponding continuity equations for the two acceleration components:

$$\begin{aligned} a_s(t_k^+) &= a_s(t_k^-)\cos\Delta\theta_k + a_n(t_k^-)\sin\Delta\theta_k\,, \\ a_n(t_k^+) &= a_n(t_k^-)\cos\Delta\theta_k - a_s(t_k^-)\sin\Delta\theta_k\,. \end{aligned} \tag{3.19}$$

### 3.1.3 *Coordinate change*

The formulation of the tool center position above detailed cannot be used for optimization purposes, for the time at which the tool reaches the discontinuity points $t_k$—i.e. the times at which $s(t_k) = s_k$—are not known *a priory*. To overcome this issue, a coordinate change is here introduced.

Lets consider a set of segments in the part program, for which the initial and final conditions are known (typically known position at zero speed). Let $L_k$ be the length of the $k$-th segment of the nominal tool path, and $T_k$ the time spent for traveling from the beginning to the end of this segment. Set also $t_0 = 0$ and $s(t_m) = L$, being $t_m = t_f$ the time at the very end of the set of segments, and $m$ is the number segments (being $m-1$ the number of discontinuity points). With these definitions, the coordinate $\zeta$ can be defined as:

$$\zeta \quad = \zeta(t) = s_{k-1} + (t - t_{k-1})\tfrac{L_k}{T_k}, \qquad t_{k-1} \le t < t_k, \tag{3.20}$$

satisfying $\zeta(t_{k-1}) = s_{k-1}$, and $\zeta(t_k) = s_{k-1} + L_k = s_k$. By using $\zeta$ as independent coordinate, the set of ODE in Eq. 3.12 becomes (for

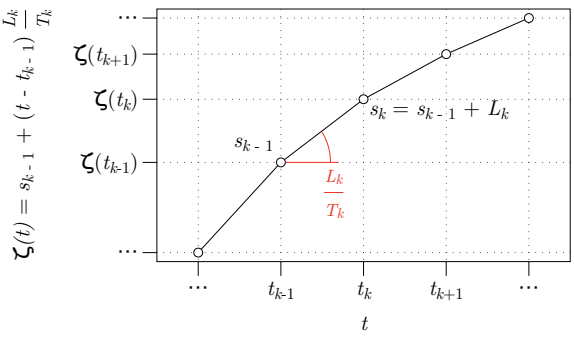



**Figure 3.2:** Coordinate change
(From [MR3] with permissions)

$\zeta \in (s_{k-1}, s_k)$):

$$\begin{aligned}
s'(\zeta) &= \left(\frac{T_k}{L_k}\right) \frac{v_s(\zeta)}{1 - n(\zeta)\kappa(s(\zeta))}, \\
n'(\zeta) &= (T_k/L_k)\, v_n(\zeta), \\
v_s'(\zeta) &= (T_k/L_k)\, a_s(\zeta) + \kappa(s(\zeta))\, v_n(\zeta)\, s'(\zeta), \\
v_n'(\zeta) &= (T_k/L_k)\, a_n(\zeta) - \kappa(s(\zeta))\, v_s(\zeta)\, s'(\zeta), \\
a_s'(\zeta) &= (T_k/L_k)\, j_s(\zeta) + \kappa(s(\zeta))\, a_n(\zeta)\, s'(\zeta), \\
a_n'(\zeta) &= (T_k/L_k)\, j_n(\zeta) - \kappa(s(\zeta))\, a_s(\zeta)\, s'(\zeta)
\end{aligned} \tag{3.21}$$

where the prime operator indicates the first derivative with respect to $\zeta$. The last set of equations is completed with the initial (i.e. $\zeta = 0$) boundary conditions:

$$\begin{aligned}
s(0) &= 0, \quad & v_s(0) &= f^-, \quad & a_s(0) &= 0, \\
n(0) &= 0, \quad & v_n(0) &= 0, \quad & a_n(0) &= 0,
\end{aligned} \tag{3.22}$$

and with the final (i.e. $\zeta = L$) boundary conditions:

$$\begin{aligned}
s(L) &= L, \quad & v_s(L) &= f^+, \quad & a_s(L) &= 0, \\
n(L) &= 0, \quad & v_n(L) &= 0, \quad & a_n(L) &= 0,
\end{aligned} \tag{3.23}$$

where $f^-$ and $f^+$ are the feed rate at the *beginning* and at the *end* of the set of segments—typically 0.

Finally, the interface conditions of Eqs. 3.15, 3.18, and 3.19—after the change of coordinates defined in Eq. 3.20—become:

$$\begin{aligned} s(s_k^+) &= s(s_k^-) = s_k \\ n_s(s_k^+) &= n_s(s_k^-) = 0 \\ v_s(s_k^+) &= v_s(s_k^-)\cos\Delta\theta_k + v_n(s_k^-)\sin\Delta\theta_k\,, \\ v_n(s_k^+) &= v_n(s_k^-)\cos\Delta\theta_k - v_s(s_k^-)\sin\Delta\theta_k\,, \\ a_s(s_k^+) &= a_s(s_k^-)\cos\Delta\theta_k + a_n(s_k^-)\sin\Delta\theta_k\,, \\ a_n(s_k^+) &= a_n(s_k^-)\cos\Delta\theta_k - a_s(s_k^-)\sin\Delta\theta_k\,. \end{aligned} \tag{3.24}$$

### 3.1.4 *Formulation of the Optimal Control Problem*

Informally, the Optimal Control Problem (OCP) can be stated as follows: one wants to calculate the continuous trajectory:

$$\boldsymbol{P}(t) = \boldsymbol{P}(s(t), n(t)),$$

where

$$\boldsymbol{P}(s, n) = \boldsymbol{P}_n(s) + n\boldsymbol{N}(\theta_n(s)),$$

which approximates the nominal path $\boldsymbol{P}_n(s)$ given a prescribed tracking tolerance and by moving as close as possible to the nominal feed rate $f(s)$, which in turn is a piecewise constant function representing the nominal feed rate for each positioning block in the part program.

Since the *path tracking error*, which is the distance between $\boldsymbol{P}_n(s)$ and $\boldsymbol{P}(s, n)$, is $|n|$ by definition, the trajectory $\boldsymbol{P}(t)$ must satisfy $n_{\min} \leq n(t) \leq n_{\max}$, where $n_{\max} \geq 0$ and $n_{\min} \leq 0$ are the maximum allowed path tracking error on the left and right side of the tool path, respectively, and where $n_{\max} - n_{\min} > 0$. It is also worth noting that, as

suggested by the same figure, the width of the error band can assume different values for each path segment, thus allowing fine-tuning of local accuracy and overall time-efficiency.

The most natural definition for a target function to be used in the trajectory planning problem is *time minimization*, i.e.:

$$\text{Minimize:} \qquad \int_0^{t_f} 1 \, \mathrm{d}t = t_f, \tag{3.25}$$

subject to the constraint on the velocity norm

$$f(s(t)) = \sqrt{v_s(t)^2 + v_n(t)^2},$$

which must not be larger than the nominal feed rate: $f^*(s(t)) \geq f(s(t))$. However, the numerical solution of the optimization problem defined by this target function proved computationally expensive.

A different formulation that approximates the minimum time problem is the minimization of the distance between the actual feed rate, $f(s(t)) = \sqrt{v_s(t)^2 + v_n(t)^2}$, and the nominal feed rate, $f^*(s(t))$. A scaled version of this distance is the following *performance index*:

$$\text{Minimize:} \qquad \int_0^{t_f} \left( \frac{f(s(t))}{f^*(s(t))} - 1 \right)^2 \mathrm{d}t. \tag{3.26}$$

Finally, to avoid excessive oscillations above the nominal feed rate, the actual feed rate must satisfy $f(s(t)) \leq f_{\max}$, where $f_{\max} \geq f^*(s)$ is the maximum feed rate allowed.

The Optimal Control Problem, by using $\zeta$ coordinate, takes the form:

1. find positive parameters $T_1, T_2, \ldots, T_m$ and control history $j_s(\zeta)$, $j_n(\zeta)$ that minimize the performance index:

$$\sum_{k=1}^{m} \left( \frac{T_k}{L_k} \right) \int_{s_{k-1}}^{s_k} \left( \frac{f(s(\zeta))}{f^*(s(\zeta))} - 1 \right)^2 \mathrm{d}\zeta, \tag{3.27}$$

2. where $v_s(\zeta)$ and $v_n(\zeta)$ are the solutions of the ODE in Eq. 3.21 with boundary conditions of Eq. 3.22–3.23 and internal or interface conditions of Eq. 3.24;

3. additional constraints on lateral position, velocity, acceleration, and jerk are also included:

$$\begin{aligned} n_{\min} \leq n(\zeta) \leq n_{\max}, \quad & v_s(\zeta)^2 + v_n(\zeta)^2 \leq f_{\max}^2, \\ |a_s(\zeta)| \leq a_{s,\max}, \quad & |a_n(\zeta)| \leq a_{n,\max}, \\ |j_s(\zeta)| \leq j_{s,\max}, \quad & |j_n(\zeta)| \leq j_{n,\max}, \end{aligned} \tag{3.28}$$

Note that Eq. 3.28 limits acceleration and jerk within a rectangle in $(s, n)$ coordinate. It is also possible to limit their values within a circle (i.e. so that they are limited in modulus):

$$a_s(\zeta)^2 + a_n(\zeta)^2 \leq a_{\max}^2, \quad j_s(\zeta)^2 + j_n(\zeta)^2 \leq j_{\max}^2, \tag{3.29}$$

or within a rectangle in $(x, y)$:

$$\begin{aligned} |a_s(\zeta)\cos(\theta) - a_n(\zeta)\sin\theta| &\leq a_{x,\max}, \\ |a_s(\zeta)\sin(\theta) + a_n(\zeta)\cos\theta| &\leq a_{y,\max}, \\ |j_s(\zeta)\cos(\theta) - j_n(\zeta)\sin\theta| &\leq j_{x,\max}, \\ |j_s(\zeta)\sin(\theta) + j_n(\zeta)\cos\theta| &\leq j_{y,\max}. \end{aligned} \tag{3.30}$$

Other kind of constraints can be set depending on the characteristic of the machine tool dynamics or on the purpose of the optimization.

It should be noted that the above formulation does not take into account the convexity of the problem. In fact — altought for simpler problems a proof of convexity can be provided with linear boundaries conditions — the much higher complexity of this formulation makes the same proof a formidable and still open issue. Nevertheless, the implementation of this formulation solves the problem of reducing the execution time of an existing part program, and it is still possibile to check, after the execution of the optimization algorithm, the

first and the second variation [17] with respect to solution in order to verify whether the solution corresponds a minimum or not. The analitycal form of such conditions are not provided here for the sake of brevity.

Solution is obtained through the high-performance in-house implementation of an indirect solver for optimal control problems [10] called PINS. Currently PINS employs as interface for problem definition the proprietary Computer Algebra System (CAS) Maple, distributed by MapleSoft, while for parameters specification an instance of *mRuby* is included in PINS binaries. Section 3.2 introduces the implementation of an ad-hoc library in pure *Ruby* language, developed specifically for code generation.

## 3.2 MR.CAS DESCRIPTION

### 3.2.1 *Software Architecture*

*Mr.CAS* is an object oriented ASD gem—i.e. a dinamically loadable *Ruby* library—that supports computer algebra routines such as simplifications and substitutions. When gem is loaded, it overloads methods of `Fixnum` (integers) and `Float` classes, making them compatible with fundamental symbolic classes.

Each symbolic expression (or operation) is the instance of an object, that inherits from a common virtual ancestor class: `CAS::Op`. An operation encapsulates sub-operations recursively, building a tree, that is the mathematical equivalent of function composition:

$$(f \circ g) \tag{3.31}$$

When a new operation is created, it is appended to the tree. The number of branches are determined by the parent container class of

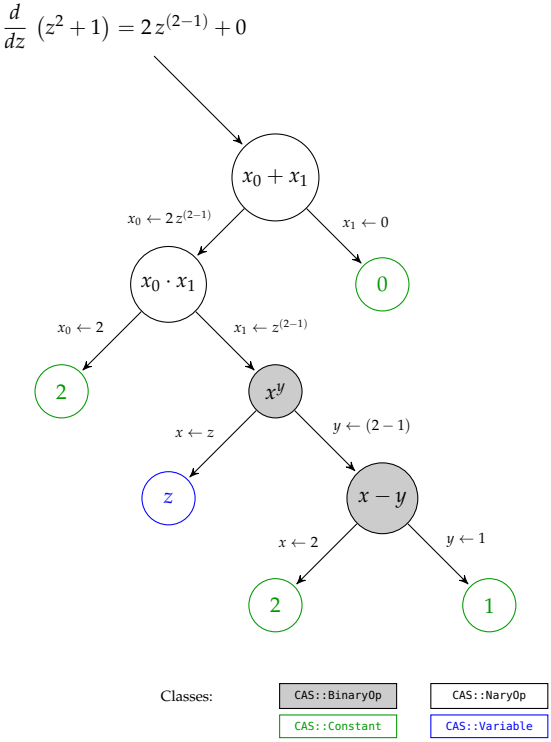



**Figure 3.3:** Tree of the expression derived in Listing 3.1
(From [MR7] with permissions)

the current symbolic function. There are three possible containers:

**`CAS::Op`** unary sub-tree operation—e.g. $\sin(\cdot)$.

**`CAS::BinaryOp`** binary sub-tree operation—e.g. exponent $x^y$—that inherits from `CAS::Op`.

**`CAS::NaryOp`** operation with arbitrary number of sub-tree—e.g. sum $x_1 + \cdots + x_N$—that inherits from `CAS::Op`.

Fig. 3.3 contains a graphical representation of an expression tree. The different kind of containers allows to introduce some properties—i.e. *associativity* and *commutativity* for sums and multiplications [22]. Each container exposes the sub-tree as instance attributes. Basic containers interfaces and inheritances are shown in Fig. 3.4. For a complete overview of all classes and inheritance, see the software documentation.

The leaves of the graph are the classes `CAS::Constant`, `CAS::Variable` and `CAS::Function`. The former models a simple numerical value, the second represents an independent variable, that can be used to perform derivatives and evaluations, and the latter is a prototype for implicit functions. Those classes support only real scalar expressions, with definition of complex, vector, and matrix extensions as milestones for the next major release.

The symbolic differentiation (`CAS::Op#diff`) explores the graph until it reaches leaf nodes. A terminal node is the starting point for derivatives accumulation, the mathematical equivalent of the chain rule:

$$(f \circ g)' = (f' \circ g)\, g' \tag{3.32}$$

The recursion is also used for simplifications (`CAS::Op#simplify`), substitutions (`CAS::Op#subs`), evaluations (`CAS::Op#call`), and code generation.

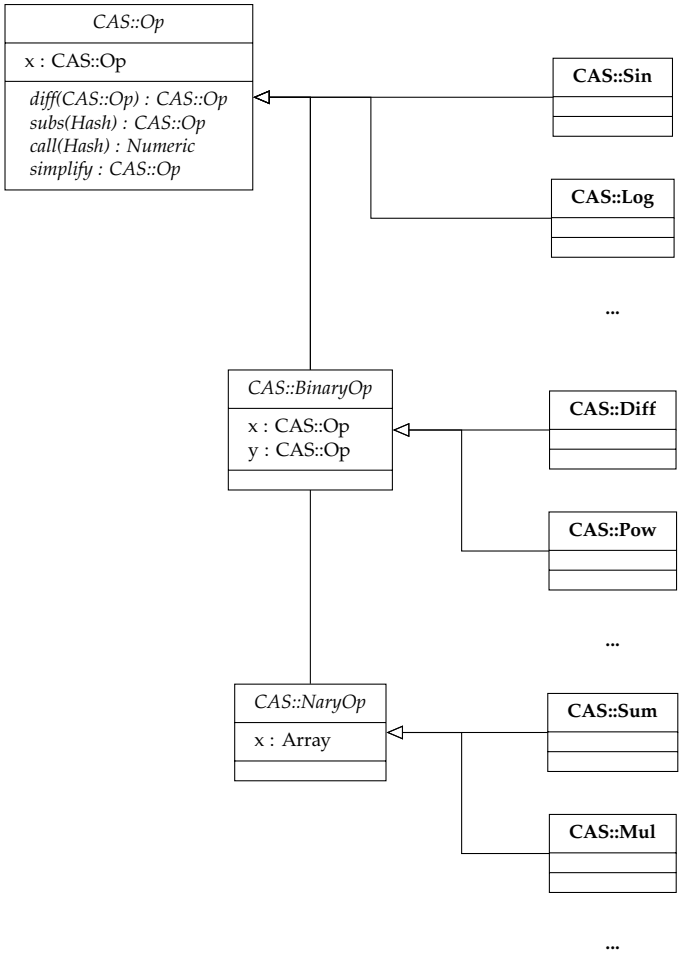



**Figure 3.4:** Reduced version of classes interface and inheritance. The figure depicts the basic abstract class `CAS::Op`, from which the *single argument* operations inherit. `CAS::Op` is also the ancestor for other kind of containers, namely the `CAS::BinaryOp` and `CAS::NaryOp`, the models of container with *two* and *more arguments*

(From [MR7] with permissions)

### 3.2.2 *Main Functionalities*

#### Basic Functionalities

*No additional dependencies are required* and the gem can be installed through the *rubygems.org* provider[1]. Gem functionalities are required using the Kernel method: `require 'Mr.CAS'`, which exposes the module `CAS`. The module contains all the methods and classes described in this chapter.

Symbolic Differentiation (SD) is performed with respect to independent variables (`CAS::Variable`) through forward accumulation, even for implicit functions. The differentiation is done by the method `CAS::Op#diff`, having a `CAS::Variable` as argument, as shown in Listing 3.1.

**Listing 3.1:** Differentiation example

```
z = CAS.vars 'z'           # creates a variable
f = z ** 2 + 1             # define a symbolic expression
f.diff(z)                  # derivative w.r.t. z
# => (((z)^((2 - 1)) * 2 * 1) + 0)
g = CAS.declare :g, f      # creates implicit expression
g.diff(z)                  # derivative w.r.t. z
# => ((((z)^((2 - 1)) * 2 * 1) + 0) * Dg[0](((z)^(2) + 1)))
```

Automatic differentiation (AD) is included as a plugin and exploits the properties of dual numbers to efficiently perform differentiation, see [5] for further details. The AD strategy is useful in case of complex expressions, where explicit derivative's tree may exceed the call stack depth.

Simplifications are not executed automatically after differentiation. Each node of the tree knows rules for simplify itself, and rules are called recursively, exactly like ASD. Simplifications that require a *heuristic expansion* of the sub-graph—i.e. some trigonometric identi-

[1] `gem install Mr.CAS`

ties—are not defined for now, but can be easily achieved through substitutions, as shown in Listing 3.2.

**Listing 3.2:** Simplification example

```
x, y = CAS::vars 'x', 'y'         # creates two variables
f = CAS.log( CAS.sin( y ) )       # symbolic expression
f.subs y => CAS.asin(CAS.exp(x)) # performs substitution
f.simplify                        # simplifies expression
# => x
```

The tree is numerically evaluated when the independent variables values are provided in a feed dictionary. The graph is reduced recursively to a single numeric value, as shown in Listing 3.3.

**Listing 3.3:** Tree evaluation example

```
x = CAS.vars 'x'   # creates a variable
f = x ** 2 + 1     # defines a symbolic expression
f.call x => 2      # evaluates for x = 2
# => 5.0
```

Symbolic expressions can be used to create comparative expressions, that are stored in special container classes, modeled by the ancestor `CAS::Condition`—for example, $f(\cdot) \geq g(\cdot)$. This allow the definition of piecewise functions, in `CAS::Piecewise`. Internally, $\max(\cdot)$ and $\min(\cdot)$ functions are declared as operations that inherits from `CAS::Piecewise`. Usage is shown in Listing 3.4.

Listing 3.4: Expressions and Piecewise functions

```
x, y = CAS.vars 'x', 'y'
f = CAS.declare :f, x
g = CAS.declare :g, x, y
h = CAS.declare :h, y

f.greater_equal g
# => (f(x) >= g(x, y))
pw = CAS::Piecewise.new(f,
       CAS::Piecewise.new(g, h, y.equal(0)),
       x.greater(0))
# => ((x > 0) ? f(x) : ((y  0) ? g(x, y) : h(y)))
CAS::max f, g
# => ((f(x) >= g(x, y)) ? f(x) : g(x, y))
```

### Meta-programming and Code-Generation

*Mr.CAS* is developed explicitly for metaprogramming and code generation. Expressions can be exported as source code or used as prototypes for callable *closures* (the `Proc` object in Listing 3.5):

Listing 3.5: Graph evaluation example

```
x = CAS::vars 'x'              # creates a variable
f = CAS::log(CAS::sin(x))      # define a symbolic function

proc = f.as_proc               # exports callable lambda
proc.call 'x' => Math::PI/2
# => 0.0
```

Compiling a closure of a tree is like making its snapshot, thus any further manipulation of the expression does not update the callable object. This drawback is balanced by the faster execution time of a `Proc`: when a graph needs *only to be evaluated*, transforming it in a *closure* reduces the execution time—for example, in a iterative algorithm, where a closure is called at each iteration.

Code generation should be flexible enough to export expression trees in a user's target language. Generation methods for common

languages are included in specific *plugins*. Users can furthermore expand exporting capabilities by writing specific exportation rules, overriding method for existing plugin, or designing their own exporter, like the one shown in Listing 3.6:

**Listing 3.6:** Example of a Fortran code generation plugin

```
# Rules definition for Fortran Language
module CAS
  {
    # . . .
    CAS::Variable => Proc.new { "#{name}" }
    CAS::Sin      => Proc.new { "sin(#{x.to_fortran})" },
    # . . .
  }.each do |cls, prc|
    cls.send(:define_method, :to_fortran, &prc)
  end
end

# Usage
x    = CAS.vars 'x'
code = (CAS.sin(x)).to_fortran
# => sin(x)
```

# 4 Tests and Validations

*The following chapter is completely dedicated to the validation of the work presented in the previous chapters.*
*The ARTool core library is benchmarked with respect to the well known* `ARUCO` *library. An example of a part-program generated through ARTool Zero is shown.*
*The usability of ARTool is assessed from three major perspective: timing, error identification performances, and cognitive workload.*
*A trajectory is optimized through the application of the OCP. A sample part-program for a milling machine is optimized. The timed outcomes with respect to different tolerances are evaluated, proving a limited acceleration and jerk with respect to the nominal path.*
*Finally, some advanced application of Mr.CAS are shown.*

## 4.1 `ARSceneDetector` BENCHMARKING

This section is dedicated to the comparison between the `ARSceneDetector` and the `ARUCO` library, which is the first solution adopted by ARTool, in the very early developing stages.

The test focuses on:

- computational time;
- reliability in marker identification;

- accuracy in ego-localization.

### 4.1.1 *Methodology*

For the localization, the ground-truth is provided by a professional level Motion Capture System (MoCap *OptiTrack*, equipped with 8 *Prime13* cameras running at 120 fps). For the localization test, a MoCap 3D reference is attached on the iPad that records a video of a board of 4 `ARUCO` markers. At least one marker is always framed during the video (see Fig. 4.1). The MoCap reference frame is placed on the coordinate system of one of the corner of one of the marker—i.e. the origin have a known offset.

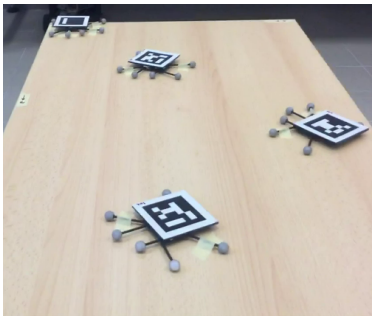


**Figure 4.1:** A frame of the video used for bench-marking

The recorded video is then used to run a testing application with both libraries in profiling mode. Setup parameters are fine tuned to crunch the maximum performances without compromising too much reliability, but some of the very advanced features of the `ARSceneDetector`—i.e. the GPU usage and the SIMD operations—are disabled for a fairer comparison. This effects the real performances of `ARSceneDetector`, but allows to limit the comparison only on the algorithmic level, rather than on differences in filtering and input data processing.

Since the signal length are different for MoCap and iPad, localiza-

**Table 4.1:** Comparison of speed (in frame per seconds) and reliability (percentage of frame identified with respect to total—21 599)

| | ARTool | ARUCO |
|---|---|---|
| Speed | 114.5 fps | 94.3 fps |
| Reliability | 98.9 % (21 380) | 86.8 % (18 739) |

tion data are synchronized minimizing the variance of positions with respect to time. Given the signals:

- $x_0(t)$ the $x$ coords returned by the MoCap at frame $t$
- $x_A(t)$ the $x$ coords returned by the ARTool library at frame $t$
- $x_B(t)$ the $x$ coords returned by the ARUCO library at frame $t$

the distance $\epsilon_x(t, \delta)$ is evaluated as:

$$\epsilon_x(t, \delta) = 2x_0(t) - ((x_A(t + \delta) + x_B(t + \delta)) \tag{4.1}$$

while the variance $\sigma_x(\delta)$ with respect to the shift $\delta$ on the $x$ signal is obtained as:

$$\sigma_x(\delta) = E\left[\epsilon_x(t, \delta) - E\left[\epsilon_x(t, \delta)\right]\right] \tag{4.2}$$

consequently, the time-shift to be used for aligning the signals is the result of:

$$\delta^* = \arg\min_{\delta} \sum_{i=\{x,y,z\}} \sigma_i(\delta) \tag{4.3}$$

Position signals are used because more reliable with respect to the others.

### 4.1.2 *Result Analysis*

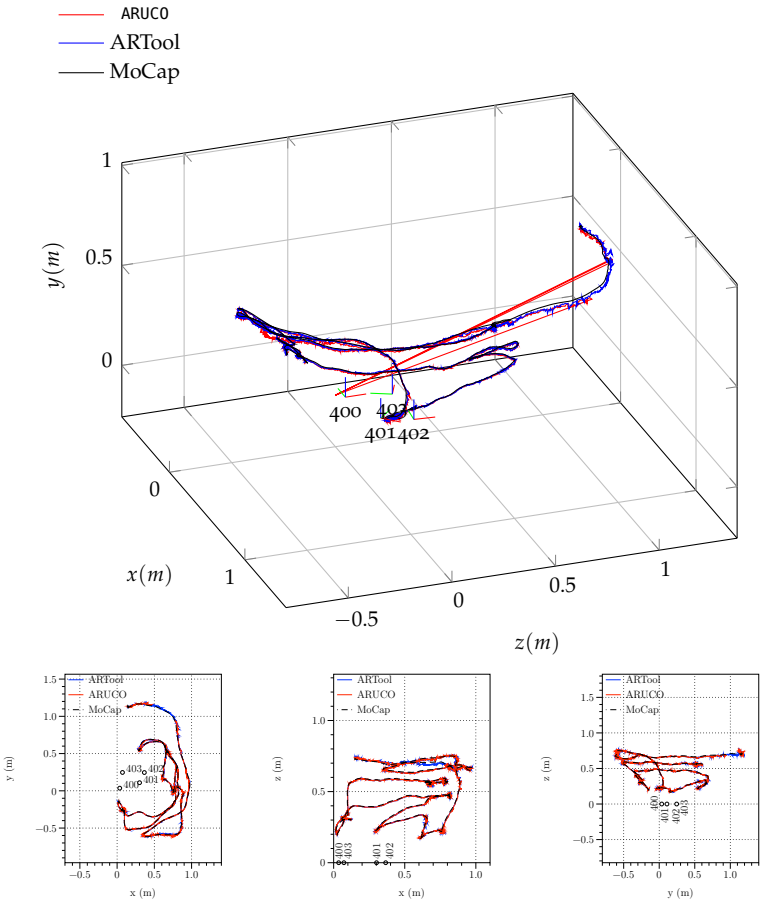



**Figure 4.2:** On top of the image the 3D representation of the trajectories in the video. The reference frames of each marker are also reported. The spikes in the ARUCO trajectory are due to missed identification of markers
(From [MR11] with permissions)

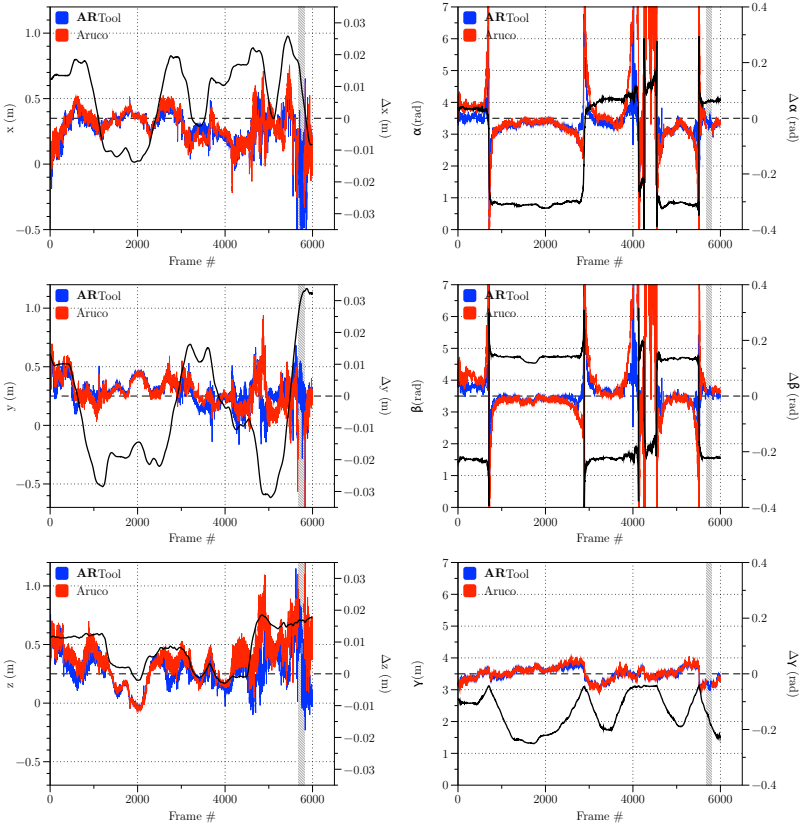



**Figure 4.3:** Ego-localization errors. On the left, there are position plot and errors between marker libraries and MoCap. On the right Euler's angles and their errors are plotted. ARUCO fails the identification between frames 5672 and 5807 (vertical hatched band)
(From [MR11] with permissions)

The benchmark trajectory in space and its projection along the principal direction is depicted in Fig. 4.2. The reference frames of the markers are also presented.

ARUCO localization presents instability, and in different occasions it is not able to reconstruct the pose of the markers. In particular,

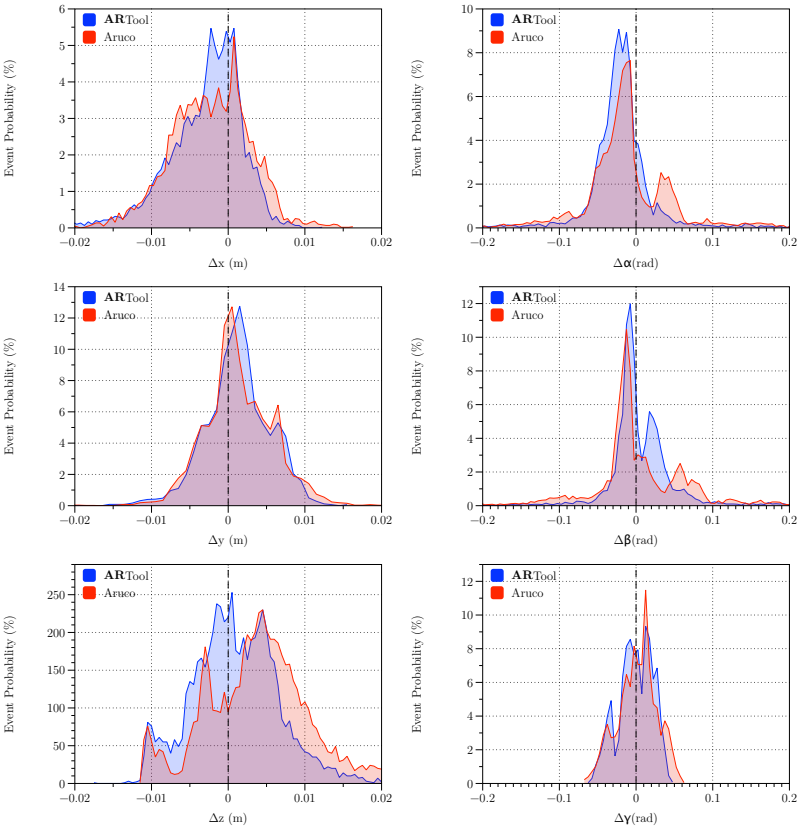



**Figure 4.4:** Ego-localization errors distribution. The left column contains positions, while the right column contains Euler's angle
(From [MR11] with permissions)

between the frame 5672 and 5807 it completely loses the tracking—i.e. the spikes in figure. For further analysis, the `ARUCO` missing trajectory is approximated linearly between the last known and the first new localization. However this segment is the main cause of the differences reported in Tab. 4.1, where it is noticeable the reliability of `ARSceneDetector`, that almost never drops track of the marker, scoring a quite high reliability index (98.9 %).

**Table 4.2:** Statistical indicators for errors distribution (mean $\mu$, standard deviation $\sigma$ and kurtosis $k$)

| | | ARTool | | |
|---|---|---|---|---|
| | | $\mu$ | $\sigma$ | $k$ |
| $x$ | (mm) | $-3.10$ | 5.38 | 7.92 |
| $y$ | (mm) | 1.22 | 4.33 | 8.75 |
| $z$ | (mm) | $9.37 \cdot 10^{-1}$ | 5.66 | 3.59 |
| $\alpha$ | (rad) | $1.92 \cdot 10^{-2}$ | $2.99 \cdot 10^{-1}$ | $1.33 \cdot 10^{-2}$ |
| $\beta$ | (rad) | $-9.07 \cdot 10^{-4}$ | $2.23 \cdot 10^{-2}$ | 2.38 |
| $\gamma$ | (rad) | $2.13 \cdot 10^{-2}$ | $3.84 \cdot 10^{-1}$ | $1.52 \cdot 10^{-2}$ |
| | | ARUCO | | |
| | | $\mu$ | $\sigma$ | $k$ |
| $x$ | (mm) | $-6.04$ | $2.93 \cdot 10^{1}$ | $7.58 \cdot 10^{1}$ |
| $y$ | (mm) | 4.05 | $2.05 \cdot 10^{1}$ | $5.85 \cdot 10^{1}$ |
| $z$ | (mm) | 4.72 | 8.20 | 5.67 |
| $\alpha$ | (rad) | $4.07 \cdot 10^{-2}$ | $3.09 \cdot 10^{-1}$ | $1.09 \cdot 10^{2}$ |
| $\beta$ | (rad) | $-4.20 \cdot 10^{-3}$ | $5.17 \cdot 10^{-2}$ | $4.75 \cdot 10^{1}$ |
| $\gamma$ | (rad) | $1.67 \cdot 10^{-2}$ | $3.39 \cdot 10^{-1}$ | $1.53 \cdot 10^{2}$ |

Fig. 4.3 shows a comparison of the three trajectory and the error of the markers detected trajectories with respect to the MoCap one. In Fig. 4.4, histograms report the probability distribution for errors. For what concern positions, the error distribution of ARUCO tends to be larger with a mode that diverges slightly from zero. Numerical analysis is reported in Tab. 4.2. Regarding attitude estimation, the performance can be considered comparable.

## 4.2 ARTool zero code example

The example presented in this section regards the identification of a corner as an intersection of three orthogonal plane in space. This is one of the very first features that an operator learns to identify.

The example is presented as a series of images that show the user interface, the simulated sequence and the executed sequence. The procedure contains a sequence of three simple contacts, as described in Section 2.5.2

In Fig. 4.5, user overlaps the mask, in gray, over the geometrical feature to be recognized, scaling its dimensions accordingly, to let the part-program generator to derive the sequence of operations. The simulated sequence is depicted in the left column of Fig. 4.6. Once the simulation is completed, a dialog window engages the user, asking to check the simulation further, to send the program to the machine controller, or to completely abort the procedure. The result, in terms of axis actual movements for the corner identification, is shown in the right column of the same figure.

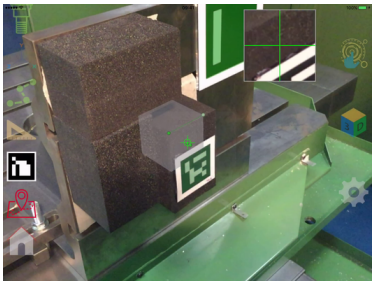
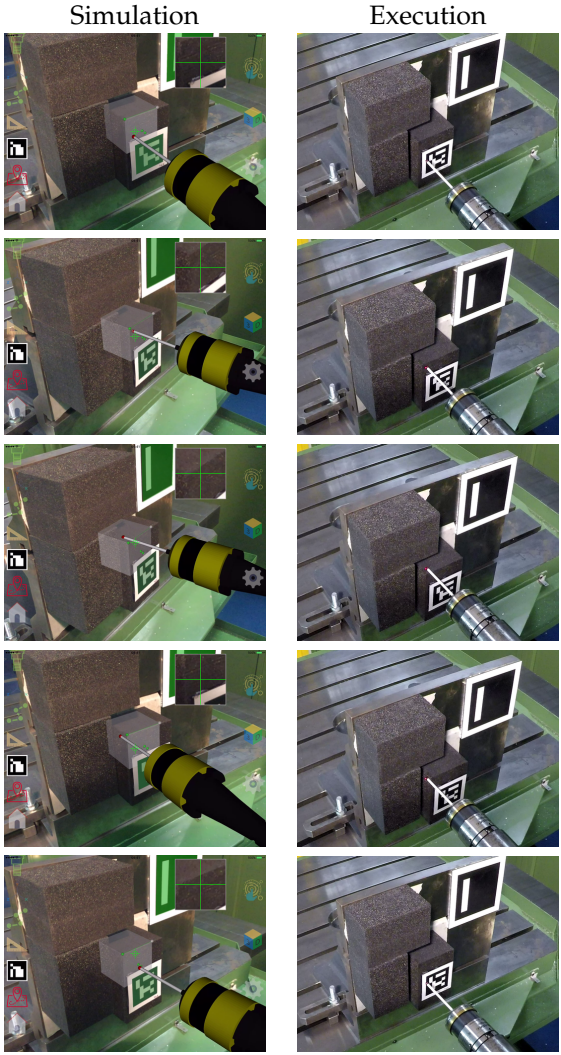


**Figure 4.5:** The mask for the corner is actually a cube, with a dot on the target corner. The image is from an alpha version of the application, in which pinch-to-zoom callback is not yet implemented, and scaling parameter is defined manually in a configuration menu

## 4.3 ARTool usability assessment

To assess the advantages provided by such a new instrument to shop floor operators, efficiency and impact on users workload has to be

**Figure 4.6:** The sequence on the left shows the simulated part program on the display of the tablet: users can frame the scene from different directions to check for collisions; on the right, the actually sequence of operations are depicted. In each of the scene it is possible to see two markers: the *feature marker* that generates the *virtual plane*, and the *machine marker*, whose position is known with respect to machine reference frame. The feature marker triggers the visualization of the *corner mask*, that is evident in simulation



investigated. This section deals with the examination of the cognitive potentials of the proposed AR tool for part-program evaluation.

The main hypotheses here taken into account were:

**Hypothesis 1** compared to the common inspection method through in-air test of a new part-program (i.e. safe execution of part-program at an offset above the workpiece along the tool axis), the AR application reduces the number of errors in detecting collisions and misalignments;

**Hypothesis 2** compared to the common inspection method through in-air test of a new part-program, the AR application reduces the time needed for detecting collisions and misalignments;

**Hypothesis 3** by using the AR application for detecting collisions, the operative workload on user with respect to normal in-air test inspection is reduced.

Hypothesis inference is made through experimental design, in which 24 participants aging from 18 to 25, mostly male [23], with instruction level between B.Sc. and M.Sc. and with no deficits in spatial capabilities, have to identify the position of any possible collision between tool and other objects in the milling machine work area, while the procedure time is recorded.

### 4.3.1 *Tasks definition*

The experimental task is to identify the position of collisions in a 5-axis milling machine (Deckel Maho DMU 60-T), between a 10 mm cylindrical milling cutter and any other object, including the workpiece. The identification is divided in two distinct procedures, proposed to participants according to a random sequence.

1. Identify the presence of **evident collisions**—e.g. at least 5 mm,

half tool diameter—through observation of in-air execution of the part-program. Vertical offset is set to 50 mm. participant can slow down, stop, or accelerate tool motion through the feed-rate override command on machine tool controller. Inversion of motion is not possible, but repetitions can be performed on request. The procedure is timed, and the participant is instructed to try and perform it as quickly as possible. The time needed for auxiliary machine operations—e.g. tool change and rapid movements—is not recorded.

2. Identify presence of **evident collisions** by using an ARTool - equipped iPad. Users can control position of the simulated spindle head and tool using an on-screen slider. The application also renders the finished part simulacrum and the complete trajectory as a dotted trace. The user can freely move and orient the device around the workpiece. Spatially, simulated bulk and real bulk are overlapped. The procedure is timed, and has to be performed as quickly as possible.

On completion of all the tasks, participants are asked to take a RAW NASA TLX survey. The RAW test is a modified version in which user evaluation is not scaled, and its use is suggested in the identification of cognitive workload with respect to the standard scaled test, when low physical workload is required [41].

### 4.3.2 *Part program description*

Three different part-program trajectories were proposed to participants during testing, and are described in Fig. 4.7:

1. is a correct execution with no errors;

2. there is a collision in corner D, along $\hat{x}$ axis, simulating a wrong definition for a variable in the part-program;

3. there is a collision through the whole trajectory, along $\hat{z}$ axis, simulating a wrong zero definition.

A collision may be located in one of the eight cardinal points of the $\hat{x} \times \hat{y}$ plane, in any arbitrary direction. Part-programs are identical on milling machine and on ARTool device, but those trajectories are provided randomly to each participant, thus, in-air and AR trajectory can be different.

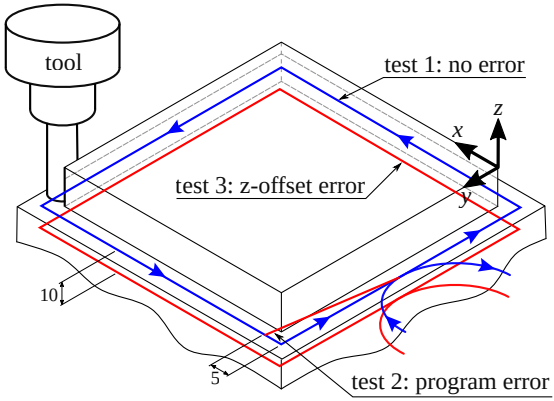



**Figure 4.7:** Visualization of the three test part-programs. *Test 1* is without collisions, while *test 2* contains a collision in $\hat{x}$ direction, and *test 3* collide through the whole trajectory due to a $\hat{z}$ shift
(From [MR11] with permissions)

### 4.3.3 *Results*

For what concerns the Hypothesis 1, test results are quite evident. None of the participants made a complete correct collision forecast with the simple observation of the in-air part-program. Conversely, by using the ARTool device 22 participants correctly identified collisions. This is an noteworthy result for users that are unexperienced both in the use of AR and of machine tools.

For what concerns the Hypothesis 2, the times required for identification of collisions in both methods are comparable. In this case it is not conceivable to reject the null hypothesis—*t-test* providing a $p$-value of 0.9839. It must be noted that timing for in-air tests only considers interpolated axes movement (`G01`/`G02`/`G03`), and not for rapid ones (`G00`), nor tool-changing axes maneuvers.

RAW NASA TLX also gives some important results regarding target self-evaluation. In fact, users were requested to answer the following 5 questions, with a score from 1 to 10:

- perceived mental demand,
- perceived temporal demand,
- required effort,
- perceived performance,
- frustration in performing the task.

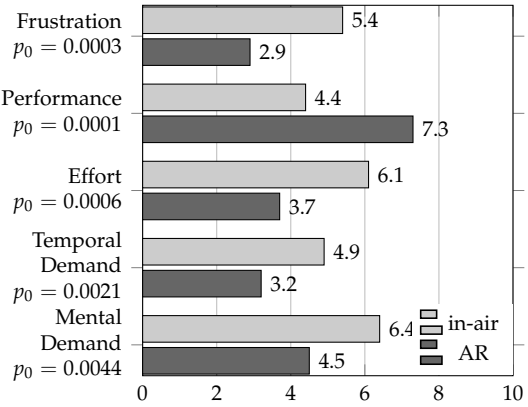



**Figure 4.8:** Mean scores for RAW NASA TLX test
(From [MR11] with permissions)

Each participant was required to answer the RAW NASA TLX test both for the AR task and for the in-air execution task. Participants

perceived AR tasks simpler to perform from a cognitive workload point-of-view. Also, frustration in performing is reduced, and perceived time request is reduced. From a cognitive workload point-of-view, this result shows an evident benefit from the introduction of ARTool in manufacturing—at least for the case of unexperienced users—increasing the self-confidence in the operation of the machine tool. Users' mean scores are reported for reference in Fig. 4.8.

## 4.4 EXPERIMENTAL VALIDATION OF OCP

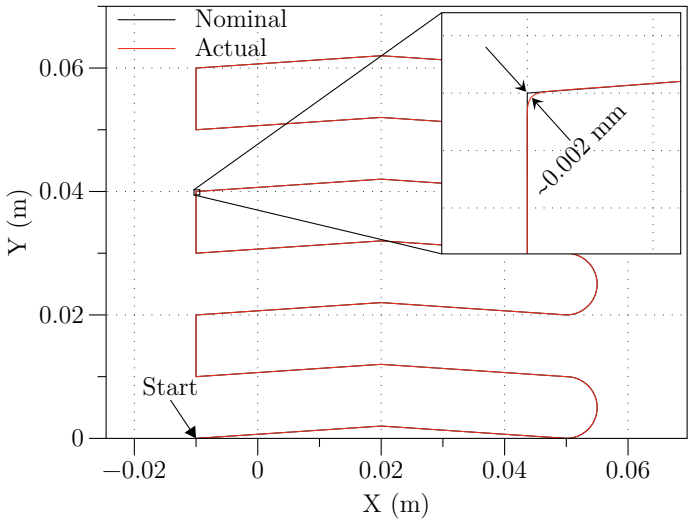



**Figure 4.9:** Test tool path. It consists in 17 straight segments and 3 circular arcs. Detail shows the difference between nominal and actual toolpath, as measured by the CNC oscilloscope
(From [MR7] with permissions)

Demonstrating the effectiveness of the approach described in the previous section would require to bypass the CNC of a real machine tool. Given that machine tools are rather complex and closed systems,

Authors decided to follow a different approach, yet opening some interesting possibilities from a practical point of view. In short, the idea is to start from a standard part program (as the one producing the tool path reported in Fig. 4.9), calculate an optimized trajectory by applying the OCP approach above described, and finally generate a new part program, where the original segments have been replaced with (typically much-) shorter segments that match the optimized tool path, and have a nominal feed rate set to the optimal one. The following subsections describe how this new part program can be generated, and how its efficiency has been tested comparing both execution time and motion smoothness when a modern CNC machine tool runs the original and the optimized part program.

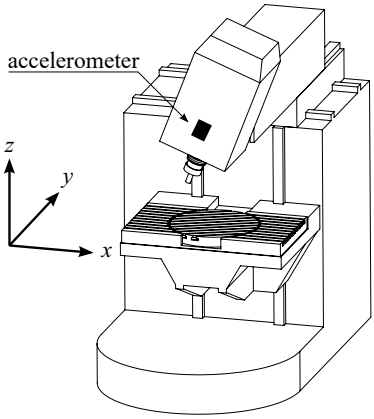



**Figure 4.10:** Experimental setup: the machine tool has X and Y axes on the head, Z on the table; a triaxial capacitive accelerometer is mounted on the moving head
(From [MR7] with permissions)

### 4.4.1 *Nodes resampling*

The solution of the OCP is stored as a sequence of points. The distance between consequent points is often smaller than the average

precision of an industrial machine tool (i.e. 5 $\mu$m). Before generating the output part program, the set of points $\mathbb{P}$ has to be reduced. Reduction is performed by a post-processing script that takes as input the OCP solution:

$$\mathbb{P} = \{\boldsymbol{P}(t) : \text{OCP solution for } \boldsymbol{P}_n(s)\} \tag{4.4}$$

and performs three steps of reduction:

1. minimum distance elimination;
2. completely aligned point elimination;
3. chordal distance elimination.

While implementation of the first step is trivial, the second and the third steps rely on the idea of a circle that passes through three points. Given a set of three non-coincident points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, $\boldsymbol{P}_3$, the center $\boldsymbol{P}_c$ and the radius $R$ of the circle are the solution of the problem:

$$\left(P_{i,x} - P_{c,x}\right)^2 + \left(P_{i,y} - P_{c,y}\right)^2 = R^2, \quad \begin{cases} i = 1 \dots 3 \\ R \geq 0 \end{cases} \tag{4.5}$$

Solution exists and it is unique. If three points are aligned, $R \to +\infty$. This is the second elimination criterion.

Chordal distance is defined as distance between point $\boldsymbol{P}_2$ and intersection point $\boldsymbol{P}_\zeta$. The position of the intersection point is given by solution of:

$$\begin{aligned} \left(P_{3,x} - P_{1,x}\right)\left(P_{\zeta,y} - P_{1,y}\right) &= \left(P_{3,y} - P_{1,y}\right)\left(P_{\zeta,x} - P_{1,x}\right) \\ \left(P_{2,x} - P_{c,x}\right)\left(P_{\zeta,y} - P_{c,y}\right) &= \left(P_{2,y} - P_{c,y}\right)\left(P_{\zeta,x} - P_{c,x}\right) \end{aligned} \tag{4.6}$$

which can be formulated in matrix form as $\mathbf{A}\boldsymbol{P}_\zeta = \mathbf{b}$, easy to solve analytically. The existence of $\mathbf{A}^{-1}$ is ensured by problem hypothesis (points not aligned). It should be clear now that we are using chordal distance as an approximation of the curvature. If this curvature is

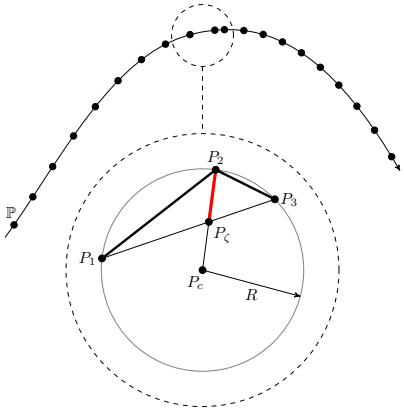



**Figure 4.11:** Circle through three points
(From [MR7] with permissions)

below a certain threshold then point $\boldsymbol{P}_2$ is eliminated, and $\boldsymbol{P}_1$ and $\boldsymbol{P}_3$ are considered aligned, as a third elimination criterion.

Description of other, more complex re-interpolation algorithms are omitted for the sake of brevity.

### 4.4.2 *Results*

The machine tool used for testing is a Deckel-Maho DMU60-T with a Heidenhain iTNC530 controller, schematically shown in Fig. 4.10. The machine has a maximum feed rate of 20 m/min on the X and Y axes and 10 m/min on the Z axis, though the latter was not moving during the tests. A triaxial capacitive accelerometer was mounted on the machine moveable head and used to record head accelerations with a sample rate of 5 kHz. Finally, the on-board software oscilloscope of the iTNC530 was used for recording time (with a sampling period of 3.6 ms), instant position and acceleration of X and Y axes, actual feed rate, and block number. part program were executed in-air (no workpiece) and with non-rotating spindle, in order to only measure

accelerations produced by the two feed axes (X and Y).

The toolpath represented in Fig. 4.9 has beed encoded into an ISO G-code part program, which has been pre-processed according to the OCP problem and the nodes resampling procedure above described. Constraints on the OCP have been set to the limits of the DMU60-T machine, according to the identification reported in Ref. [13]. Several combinations of optimization parameters and node reresampling parameters have been tested. Due to space restrictions, only the more interesting cases are hereafter reported, and namely three combinations of tool path tracking tolerance: 50 $\mu$m, 250 $\mu$m, and 500 $\mu$m). Eventually, the original part program and the pre-processed ones have been executed on a CNC machine tool.

It is worth noting that the system here proposed is of special interest for high speed movements, i.e. when the nominal feed rate is close or equal to the machine limit, since under these conditions most of the cycle time is spent in accelerating or decelerating the axes, and a constant value of the actual feed rate is seldom maintained for the larger part of each positioning block. For this reason, the tests were performed at relatively high feed rates of 5, 15, and 20 m/min, the latter corresponding to the machine limit.

Figure 4.12 compares the nominal tool path with the actual tool paths recorded by the onboard oscilloscope for the three selected tracking tolerances. Figure 4.13 shows how the actual tool paths comply with selected tracking tolerance limits, with the notable exception of some short overshoots in the case of the smaller tolerance, which is related to the fact that the OCP solver implements the constraints on tracking error as penalty functions.

It is evident that the optimized part program follows a smoother trajectory that results in smaller accelerations, as can be noted by observing the acceleration components $a_x$ and $a_y$ reported in Fig. 4.15. At the same time—and what really matters—the execution time is significantly reduced as can be observed from data in Tab. 4.3, with gains

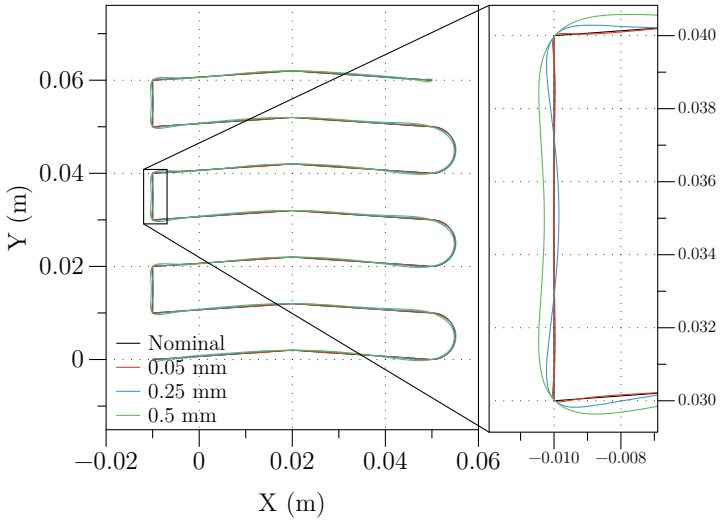



**Figure 4.12:** Nominal tool path (black) compared with the real tool paths as reported by the on-board CNC oscilloscope with three different tracking tolerances

(From [MR7] with permissions)

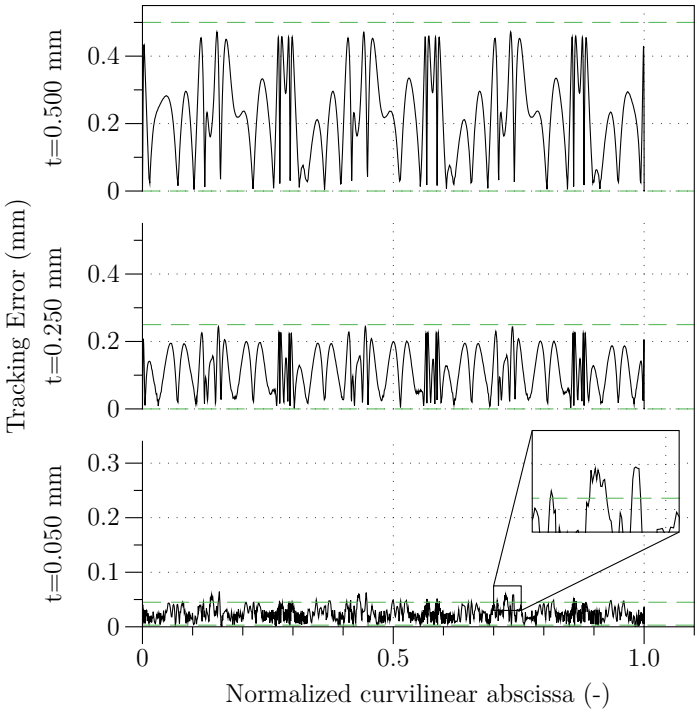



**Figure 4.13:** Path tracking errors compared with OCP tracking tolerances
(From [MR7] with permissions)

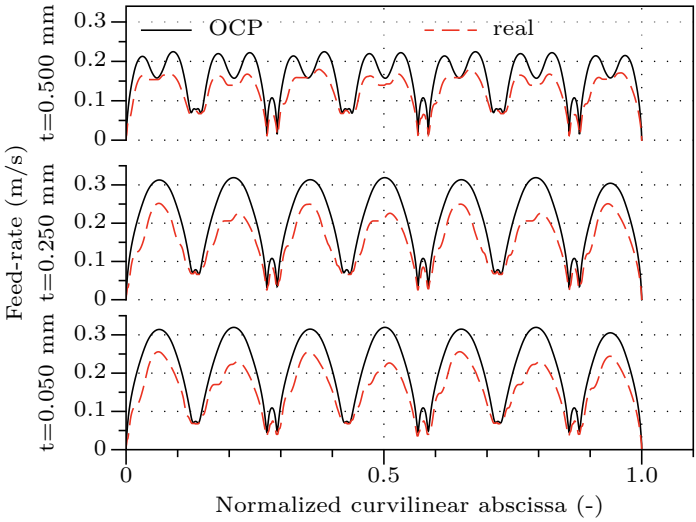



**Figure 4.14:** Actual feed rate as measured by the oscilloscope compared with OCP result, plotted against normalized nominal curvilinear abscissa for three different tracking tolerances
(From [MR7] with permissions)

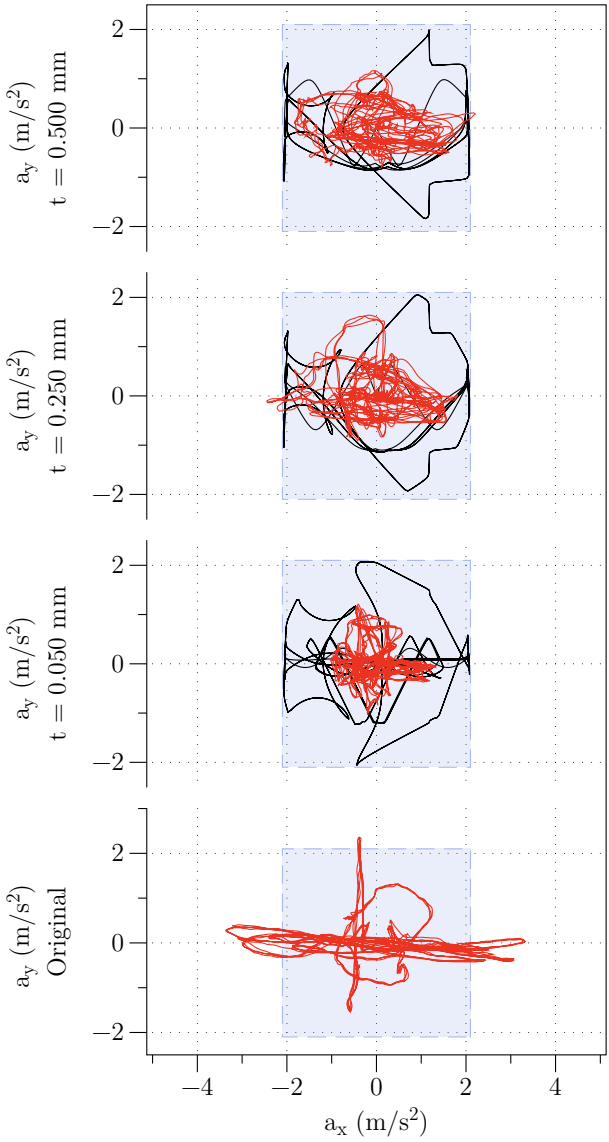



**Figure 4.15:** Accelerations along X and Y axes, as measured by the accelerometer (red) and as calculated by the OCP. Square boxes represent the constraints set to the OCP
(From [MR7] with permissions)

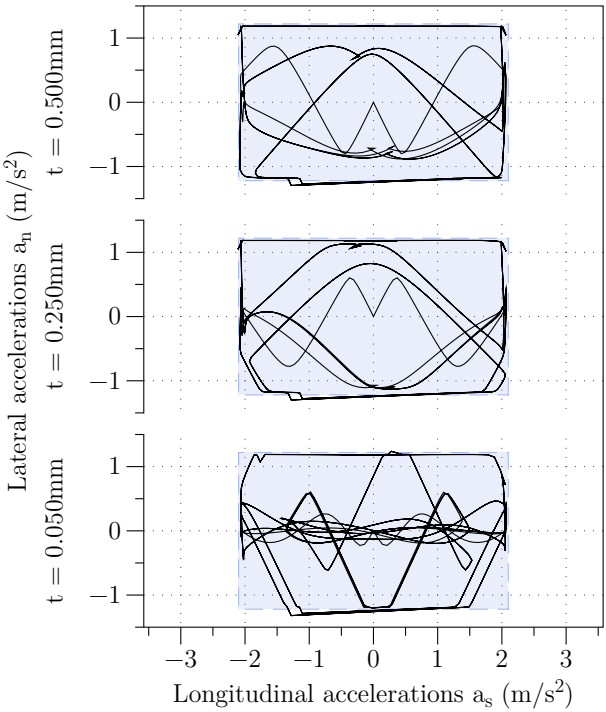



**Figure 4.16:** Longitudinal and lateral accelerations,calculated by OCP. Square boxes represent the constraint set in longitudinal and lateral directions
(From [MR7] with permissions)

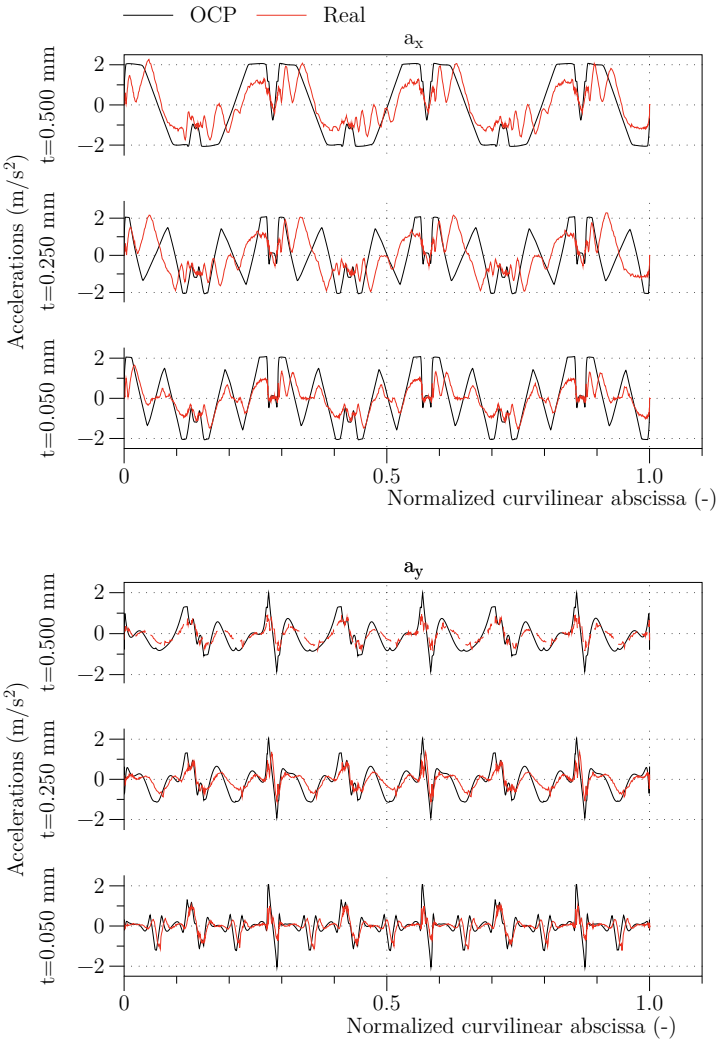



**Figure 4.17:** Comparison between OCP solution (black) and real machine acceleration (red, measures from on-board oscilloscope)
(From [MR7] with permissions)

ranging from 5 to 20% in the most relevant cases. Moreover, it should be noted that the execution times obtained by the OCP solution are

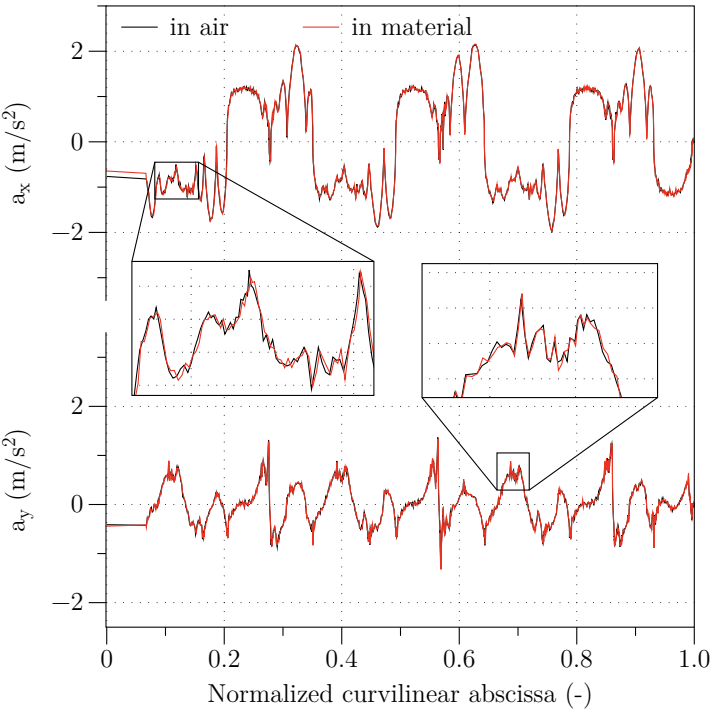



**Figure 4.18:** Comparison between execution in air and in material, with feed 18 m/min and tolerance 500 $\mu$m
(From [MR7] with permissions)

Table 4.3: part program execution times

| feed rate (m/min) | tolerance (mm) | exec. time (s) | change (%) | OCP exec. time (s) |
|---|---|---|---|---|
| 20.00 | nominal p.p. | 4.67 | – | – |
| 20.00 | 0.50 | 3.98 | −14.8 | 3.02 |
| 20.00 | 0.25 | 4.42 | −5.2 | 3.13 |
| 20.00 | 0.05 | 4.97 | 6.4 | 3.69 |
| 15.00 | nominal p.p. | 5.22 | – | – |
| 15.00 | 0.50 | 4.21 | −19.4 | 3.18 |
| 15.00 | 0.25 | 4.50 | −13.9 | 3.29 |
| 15.00 | 0.05 | 4.99 | −4.4 | 3.71 |
| 5.00 | nominal p.p. | 6.93 | – | – |
| 5.00 | 0.50 | 6.67 | −3.6 | 6.27 |
| 5.00 | 0.25 | 7.85 | 13.4 | 6.36 |
| 5.00 | 0.05 | 8.21 | 18.5 | 6.40 |

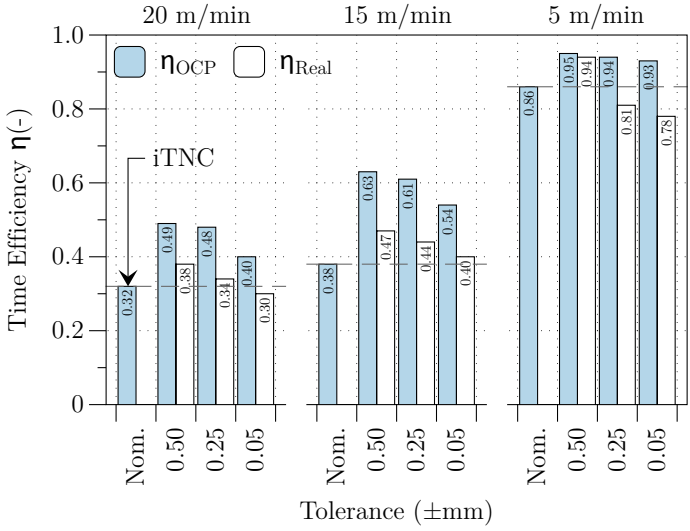



Figure 4.19: Comparison for time efficiency metric $\eta$

always faster than the nominal case, meaning that the complete substitution of current CNC profiling/interpolation scheme with one

based on the OCP described in Ref. [13] would ensure significant advantages in every condition. The different performance of optimization and real execution has been assessed using time efficiency metrics [13], defined as ratio between minimum theoretical execution time and effective execution time:

$$\eta = \frac{L}{f\,T} \tag{4.7}$$

Time efficiency for both OCP and real execution time are compared in Fig. 4.19, while timing are reported in Table 4.3.

On the negative side, Fig. 4.14 remarks how the CNC own feed rate profiling is significantly reducing the actual feed rate with respect to that resulting from the OCP solution, thus somehow limiting the effectiveness of the proposed solution in terms of minimum attainable execution time. Those effects could also be seen in terms of optimized and actual accelerations. Even if the optimized solution lays inside experimentally identified boundaries (Fig. 4.16), sometimes during the tests it appears to be further limited by the CNC, Fig. 4.17.

In Fig. 4.18 a comparison of accelerations between two executions of the same optimized part program — in-air and in-material — is presented. The operation is the same toolpath as in Fig. 4.9, with a nominal feed rate 18 m/min and a tolerance of 500 $\mu$m. It is clear that presence of material does not change the foundamental dynamic of execution, but it also gives only an higher frequency contribution.

## 4.5 ADVANCED USAGE FOR MR.CAS

### 4.5.1 *Code Generation as C Library*

This example shows how a *user of* Mr.CAS can export a mathematical model as a C library. The `c-opt` plugin implements advanced features

such as code optimization and generation of libraries.

The library `example` implements the model:

$$f(x,y) = x^y + g(x)\ \log(\sin(x^y)) \tag{4.8}$$

where the expression $g(x)$ belongs to a external object, declared as `g_impl`, whose interface is described in `g_impl.h` header. What should be noted is the corpus of the exported code: the intermediate operation $x^y$ is evaluated once, even if appears twice in eq. 4.8. The C function that implements $f(x,y)$ is declared with the token `f_impl`. The exporter uses as default type `double` for variables and function returned values. Library created by `CLib` contains the code shown in Listing 4.3.

**Listing 4.1:** Calling optimized-C exporter for library generation

```
# Model
x, y = CAS.vars :x, :y
g = CAS.declare :g, x

f = x ** y + g * CAS.log(CAS.sin(x ** y))

# Code Generation
g.c_name = 'g_impl'             # g token

CAS::CLib.create "example" do
  include_local "g_impl"        # g header
  implements_as "f_impl", f     # token for f
end
```

**Listing 4.2:** C Header

```
// Header file for library:
    example.c

#ifndef example_H
#define example_H

// Standard Libraries
#include <math.h>

// Local Libraries
#include "g_impl"

// Definitions

// Functions
double f_impl(
  double x, double y);

#endif // example_H
```

**Listing 4.3:** C Source

```
// Source file for library:
    example.c

#include "example.h"


double f_impl(double x, double y)
    {
  double __t_0 = pow(x, y);
  double __t_1 = g_impl(x);
  double __t_2 = sin(__t_0);
  double __t_3 = log(__t_2);
  double __t_4 = (__t_1 + __t_3);
  double __t_5 = (__t_0 + __t_4);

  return __t_5;
}

// end of example.c
```

The function $g(x)$ models the following operation:

$$g(x) = (\sqrt{x+a} - \sqrt{x}) + \sqrt{\pi + x} \tag{4.9}$$

and may suffer from *catastrophic numerical cancellation* [45] when the $x$ value is considerably greater than $a$. The user may decide to specialize code generation rules for this particular expression, stabilizing it through rationalization. Without modifying the actual model, $g(x)$ the rationalization for differences of square roots[1] is inserted into the exportation rules, as in Listing 4.4. The rules are valid only for the current user script. For more insight about `__to_c` and `__to_c_impl`, refer to the software manual.

[1] i.e.: $\sqrt{\phi(\cdot)} - \sqrt{\psi(\cdot)} = \dfrac{\phi(\cdot) - \psi(\cdot)}{\sqrt{\phi(\cdot)} + \sqrt{\psi(\cdot)}}$

**Listing 4.4:** Conditioning in exporting function

```
# Model
a = CAS.declare "PARAM_A"

g = (CAS.sqrt(x + a) - CAS.sqrt(x)) + CAS.sqrt(CAS::Pi + x)

# Particular Code Generation for difference between square roots.
module CAS
  class Diff
    alias :__to_c_impl_old :__to_c_impl

    def __to_c_impl(v)
      if @x.is_a? CAS::Sqrt and @y.is_a? CAS::Sqrt
        "(#{@x.x.__to_c(v)} + #{@y.x.__to_c(v)}) / " +
        "( #{@x.__to_c(v)} + #{@y.__to_c(v)} )"
      else
        self.__to_c_impl_old(v)
      end
    end
  end
end

CAS::CLib.create "g_impl" do
  define "PARAM_A()", 1.0   # Arbitrary value for PARAM_A
  define "M_PI", Math::Pi
  implements_as "g_impl", g
end

puts g
# => ((sqrt((x + PARAM_A())) - sqrt(x)) + sqrt(( + x)))
```

It should be noted the separation between the *model*, which does not contain stabilization, and the *code generation rule*. For this particular case, the code generation rule in Listing 4.4 overloads the predefined one, in order to obtain the conditioned code. Obviously, the user can decide to apply directly the conditioning on the model itself, but this may change the calculus behavior in further manipulation.

**Listing 4.5:** `g_impl` Header

```
// Header file for library: g_impl
    .c

#ifndef g_impl_H
#define g_impl_H

// Standard Libraries
#include <math.h>

// Local Libraries


// Definitions
#define PARAM_A() 1.0
#define M_PI 3.141592653589793

// Functions
double g_impl(double x);

#endif // g_impl_H
```

**Listing 4.6:** `g_impl` Source

```
// Source file for library: g_impl
    .c

#include "g_impl.h"

double g_impl(double x) {
  double __t_0 = PARAM_A();
  double __t_1 = (x + __t_0);
  double __t_2 = sqrt(__t_1);
  double __t_3 = sqrt(x);
  double __t_4 = (__t_1 + x) / \
    ( __t_2 + __t_3 );
  double __t_5 = (M_PI + x);
  double __t_6 = sqrt(__t_5);
  double __t_7 = (__t_4 + __t_6);

  return __t_7;
}

// end of g_impl.c
```

### 4.5.2 *Using the module as interface*

As example, an implementation of an algorithm that estimates the *order of convergence* for trapezoidal integration scheme [94] is provided, using the symbolic differentiation as interface.

Given a function $f(x)$, the trapezoidal rule for primitive estimation for the interval $[a, b]$ is:

$$I_n(a,b) = h\left(\frac{f(a)+f(b)}{2} + \sum_{k=1}^{n-1} f\left(a + k\,h\right)\right) \tag{4.10}$$

with $h = (b-a)/n$, where $n$ mediates the step size of the integration. When exact primitive $F(x)$ is known, approximation error is:

$$E[n] = F(b) - F(a) - I_n(a,b) \tag{4.11}$$

The error has an asymptotic expansion of the form:

$$E[n] \propto C\, n^{-p} \tag{4.12}$$

where $p$ is the convergence order. Using a different value for $n$, for example $2\,n$, the ratio 4.13 takes the approximate vale:

$$\frac{E[n]}{E[2\,n]} \approx 2^p \quad \rightarrow \quad p \approx \log_2 \left( \frac{E[n]}{E[2\,n]} \right) \tag{4.13}$$

Listings 4.7 and 4.8 contain the implementation of the described procedure using the proposed gem and the well known *Python* [84] library for symbolic math *SymPy* [75]. Mean time comparison for the scripts is reported in Tab. 4.4.

**Listing 4.7:** Ruby version

```ruby
require 'Mr.CAS'


def integrate(f, a, b, n)
  h = (b - a) / n

  func = f.as_proc

  sum = ((func.call 'x' => a) +
        (func.call 'x' => b)) /
            2.0

  for i in (1...n)
    sum += (func.call 'x' =>
      (a + i*h))
  end
  return sum * h
end

def order(f, a, b, n)
  x = CAS.vars 'x'

  f_ab = (f.call x => b) -
         (f.call x => a)
  df   = f.diff(x).simplify
  f_1n = integrate(df, a, b, n)
  f_2n = integrate(df, a, b, 2 * n
       )

  return Math.log(
    (f_ab - f_1n) /
    (f_ab - f_2n),
  2)
end

x = CAS.vars 'x'
f = CAS.arctan x

puts(order f, -1.0, 1.0, 100)
# => 1.9999999974244451
```

**Listing 4.8:** Python version

```python
import sympy
import math

def integrate(f, a, b, n):
  h = (b - a)/n
  x = sympy.symbols('x')
  func = sympy.lambdify((x), f)

  sums = (func(a) +
          func(b)) / 2.0

  for i in range(1, n):
    sums += func(a + i*h)

  return sums * h


def order(f, a, b, n):
  x = sympy.symbols('x')

  f_ab = sympy.Subs(f, (x), (b)).n
       () - \
         sympy.Subs(f, (x), (a)).n
              ()
  df   = f.diff(x)
  f_1n = integrate(df, a, b, n)
  f_2n = integrate(df, a, b, 2 * n
       )

  return math.log(
    (f_ab - f_1n) /
    (f_ab - f_2n),
  2)


x = sympy.symbols('x')
f = sympy.atan(x)

print(order(f, -1.0, 1.0, 100))
# => 1.9999999974244451
```

**Table 4.4:** Mean time for examples 4.7 and 4.8 on Intel i7 M640 with Arch Linux amd64. Output from `time` keyword in `bash` 4.4.12

| | Ruby (2.4.3) Mr.CAS (0.2.7) | Python (3.6.4) Sympy (1.1.1) |
|---|---|---|
| Real | 0.104 s | 0.551 s |
| User | 0.096 s | 0.514 s |
| System | 0.006 s | 0.042 s |

### 4.5.3 *ODE Solver with Taylor's series*

In this example, a solving step for a specific ordinary differential equation (ODE) using Taylor's series method [18] is derived. Given an ODE in the form:

$$y'(x) = f\left(x, y(x)\right) \tag{4.14}$$

the integration step with order $n$ has the form:

$$y(x+h) = y(x) + h\,y'(x) + \cdots + \frac{h^n}{n!}\,y^{(n)}(x) + E_n(x) \tag{4.15}$$

where it is possible to substitute equation 4.14:

$$y^{(i)}(x) = \frac{\partial y^{(i-1)}(x)}{\partial x} + \frac{\partial y^{(i-1)}(x)}{\partial y} y'(x) \tag{4.16}$$

For this algorithm, three methods are defined. The first evaluates the factorial, the second evaluates the list of required derivatives, and the third returns the integration step in a symbolic form. The result of the third method is transformed in a C function. In this particular case, the ODE is $y' = xy$. For the resulting C code of Listing 4.9, refer to the online version of the examples.

**Listing 4.9:** Generator for ODE integration step

```
$x, $y, $h = CAS::vars :x, :y, :h
# Evaluates n!
def fact(n); (n < 2 ? 1 : n * fact(n - 1)); end
# Evaluates all derivatives required by the order
def coeff(f, n)
  df = [f]
  for _ in 2..n
    df << df[-1].diff($x).simplify + (df[-1].diff($y).simplify * df
        [0])
  end
  return df
end
# Generates the symbolic form for a Taylor step
def taylor(f, n)
  df = coeff(f, n)
  y = $y
  for i in 0...df.size
    y = y + (($h ** (i + 1))/(fact(i + 1)) * df[i])
  end
  return y.simplify
end

# Example function for the integrator
f = $x * $y
# Exporting a C function
clib = CAS::CLib.create "taylor" do
  implements_as "taylor_step", taylor(f, 4)
end
```

Other examples are available online[2]: (*a*) adding a user defined `CAS::Op` that implements the sign$(\cdot)$ function with the appropriate optimized C generation rule; (*b*) exporting the operation as a continuous function through overloading or substitutions; (*c*) performing a symbolic Taylor's series; (*d*) writing an exporter for the LaTeX language; (*e*) a Newton-Raphson algorithm using automatic differentiation plugin.

[2] `http://bit.ly/Mr_CAS_examples`

# 5 Deep Understanding

*The chapter engages the application of artificial neural networks to industrial problems. The chapter introduces the reader to the neural nets terminology and current practices, while showing the application of such techniques to identification and regression of phase in X-Ray Crystallography. The example allows to introduce the representation problem, and the unsupervised learning for such representation through autoencoders, for which a library has been implemented.*

## 5.1 A BRIEF INTRODUCTION TO ARTIFICIAL NEURAL NETS

### 5.1.1 *Algorithms as Functions*

Automated sytems should gain knowledge from experience, without the need of human intervention in specifying all the rules needed to perform a task, in terms of formal languages. This approach is at the very core of the *Machine Learning* (ML) field, in which, instead of *manually* describing a procedure to solve a task, an algorithm to perform it is *automatically* derived.

The performance of such algorithms depends heavily on the interpretation of the input. From those data it is important to outline

the essential information—i.e. *features*. This problem may be handled manually with some *feature engineering*, or leaved to the learning algorithm. In the latter case, the machine not only learns how to solve the task, but also how to elaborate a better representation that points out the prominent attributes to perform the task. This approach is know as *representational learning* [46]. One class of algorithms that is able to tackle the representation of data autonomously are the *Artificial Neural Networks* (ANNs) [40].

ANNs are systems of interconnected nodes that operates a simple transformation from an input $\mathbf{x}$ to an output $\mathbf{y}$:

$$\text{ANN} : \mathbb{R}^{\dim(\mathbf{x})} \mapsto \mathbb{R}^{\dim(\mathbf{y})} \tag{5.1}$$

### 5.1.2 *Topology of an ANN*

Topologically, an ANN is composed by *layers*. There are many different *layers* formulations in the field of ANNs, but the most common one is the *fully connected* structure, in which the layers comprises simpler operations, called neurons, that share the same input. The neuron may have different mathematical formulations, but the most common one is the *perceptron* [68]:

$$p : \mathbb{R}^{\dim(\mathbf{z})} \mapsto \mathbb{R}, \quad p(\mathbf{z} \mid \mathbf{w}, b) = f\left(\mathbf{z} \cdot \mathbf{w} + b\right) = h \tag{5.2}$$

where $\mathbf{w}$ (weights) and $b$ (bias) are *trainable* parameters—i.e. their values must be learned through a training procedure—and $f$ is an *activation function* $f : \mathbb{R}^{\dim(\mathbf{w})} \mapsto \mathbb{R}$. Fig. 5.1 represents a simple perceptron. When more than one perceptron is combined, the resulting structure is a layer:

$$l(\mathbf{z}) : \mathbb{R}^{\dim(\mathbf{z})} \mapsto \mathbb{R}^{\dim(\mathbf{h})}, \tag{5.3}$$

$$l(\mathbf{z}) = \left[p_1(z \mid \mathbf{w}_1, \mathbf{b}_1), \ldots, p_{\dim(\mathbf{h})}(z \mid \mathbf{w}_{\dim(\mathbf{h})}, \mathbf{b}_{\dim(\mathbf{h})})\right] = \mathbf{h} \tag{5.4}$$

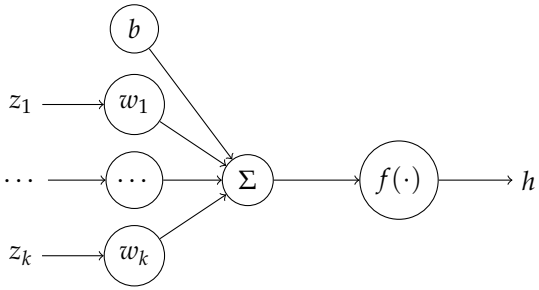



**Figure 5.1:** The representation of a perceptron

An ANN may have more than one layer, and the inner ones are commonly entitled *hidden layers*. The most common ANN, the multi-layer perceptron, is similar to the one presented in Fig. 5.3.

### 5.1.3 *Different Approaches to Learning*

There are different approaches to learning. The first one that is taken into account is the *supervised learning*. Algorithms trained with this approach are mainly employed for:

**classification** to discriminate input data in different categories;

**regression** to estimate the relationship among input to predict a continuous or discrete series of output.

The training procedure is used to determine the value of the weights ($\mathbf{w}$) and biases ($b$) in order to obtain the desired behavior from the network. The term "supervised" underlines the fact that the machine observes a series of complete examples for which the correct answer is know—i.e. the *training* dataset has examples that are a tuple of pattern and output: $[\mathbf{x}, \hat{y}]$.

The evolution of the network is guided through a *loss* function that underlines how the networks is performing with respect to the super-

vised data. The loss depends on the network itself:

$$\mathcal{L} : \mathbb{R}^{\dim(\mathbf{y})} \times \mathbb{R}^{\dim(\mathbf{y})} \mapsto \mathbb{R} \tag{5.5}$$

and the training is an optimization problem:

$$\begin{aligned} \text{minimize} \quad & \mathcal{L}(\text{ANN}(\mathbf{x} \mid \mathbf{w}, \mathbf{b}), \bar{\mathbf{y}}) \\ & [\mathbf{x}, \bar{\mathbf{y}}] \in \text{Database of examples} \end{aligned} \tag{5.6}$$

where $\bar{\mathbf{y}}$ represents the supervised output, the one that should be returned by the perfect ANN. The optimization is carried out through different flavor of *gradient descent* algorithm. One of the advantages of the ANNs approach is related to the fact that the training can be carried out by showing a *partial sequence* of examples (batches) from the training dataset to the optimization routine and statistically update the gradient that guides the optimization step—i.e. *stochastic gradient descent* (SDG). Several algorithms based upon this approach have been derived to train networks—e.g. SDG [14], ADAM, ADAGRAG [30], RMSPROP, etc. This approach allows to train with respect to enormous database of examples, and it is in contrast with respect to many ML algorithms where the training requires the whole dataset in a single batch, limiting the maximum number of examples, due to technological constraints.

A different approach of particular interest for the challenges proposed by Industry 4.0 is the *unsupervised learning*, where learning refers to the capacity of the automatic algorithm to discover, if present, the inner structure of data that are not labeled.

Multiple target for representation may be achieved, where the most employed ones are:

**lower dimensional representation** a representation that describes the input points with no information loss, but using a dimensionally smaller space (lower number of features);

**sparser representation** a representation in which input are sparse has almost all dimensions equal to zero: the information tend to accumulate along some directions;

**independent representation** different dimensions are considered as orthogonal in the representation of the data.

Those representations are not mutually exclusive and expose some interesting property to exploit.

Other typical training approaches are: the *semi-supervised learning* (a combination of the previous two techniques), and the *reinforcement learning* (the learner, which acts over an environment, is trained on the basis of a specific reward).

## 5.2 ANNS APPLIED TO AN INDUSTRIAL PROBLEM

### 5.2.1 *ANNs and X-Ray Crystallography*

When an X-ray beam is shot toward a sample of matter, the sample reflects the incident beam with a specific intensity that depends upon the angle between the incident ray and the surface normal. The reflections generate patterns that contain information upon atomic and molecular structures of the sample.

This physical phenomenon has been deeply employed in different scientific fields, in particular in the non-destructive identification of crystalline matter, where the reflections are function of the Bragg's Law [93]. It is possible to identify the substances in a crystalline powder by comparing the peaks of the experiments in the X-ray difractometer with respect to the peaks associated to substances in big databases of theoretical and experimental patterns. Nevertheless, due to noise and concentration uncertanties, it is necessary to manually handle the experimental pattern. Moreover, when the sample is

a compound of different substances—phases—the experience of the researcher becomes fundamental.

At the core of the work there is an architecture where a 1D convolutional classifier operates in tandem with a fully connected regressor. The two networks are developed to be general, but trained specifically to recognize the presence and regress the concentration of a particular phase $P_i$.

The two networks fulfill the following pseudo-code:

---

1: **function** $P_i(\mathbf{x})$
2: **if** $\text{ANN}_{\text{cl}}^{P_i}(\mathbf{x}) > 0.5$ **then**
3: **return** $\text{ANN}_{\text{rg}}^{P_i}(\mathbf{x})$
4: **else**
5: **return** $0.0$
6: **end if**
7: **end function**

---

where $\text{ANN}_{\text{cl}}^{P_i}$ is the classifier trained on the phase $P_i$ and $\text{ANN}_{\text{rg}}^{P_i}$ is the regressor, trained on the same phase. Both networks operate on an input spectrum $\mathbf{x}$ (normalized with respect to the maximum value). The full *application* is composed by different ANN of different phases. Notice that the regressor is actually called in the application flow only if the classifier identifies the specific phase.

Conceptually, the pseudo-code for the complete application is:

---

1: **function** APPLICATION($\mathbf{x}$)
2: **return** $\{ P_1 : P_1(\mathbf{x}), \ldots, P_n : P_n(\mathbf{x}) \}$
3: **end function**

---

that allows to arbitrarily change the number of the phase, but does not guarantee that the sum of concentrations is 1.0.

The formal definition of the classifier function is:

$$\mathrm{ANN}_{\mathrm{cl}}^{P} \; : \; [0,1]^{\dim(x)} \mapsto [0,1] \tag{5.7}$$

and the output domain is the probability of presence of the phase $P$. The greater the probability, the greater the confidence of the network. The fact that there may be millions of phases, and thus millions of classifiers that run for a single spectrum sets some limitations upon the classifier structure: the classifier must be computationally as lightweight as possible, while keeping an overall high accuracy. This is the reason why the classifier contains some convolutional layers (see Section 5.2.2) to maintain the number of parameters to train as limited as possible. This parameters reduction allows to load several networks in calculator memory simultaneously, a straightforward way to parallelize the task. The structure of the network is summarized in Fig. 5.2. The convolutional layers of the network try to project the information in a compressed space. The compression is obtained through the shared kernels.

The formal definition of the regressor is:

$$\mathrm{ANN}_{\mathrm{rg}}^{P} \; : \; [0,1]^{\dim(x)} \mapsto [0,100] \tag{5.8}$$

and the output is the concentration of the phase in percentage. Since the number of regressors that must be loaded in memory is limited by the classification, there is more freedom in the selection of the architecture of this network. The structure is summarized in Fig. 5.3. The fully connected network is the one that proved the best performances.

### 5.2.2 *The Convolution Layer*

The adoption of functional layers in a ANN allows to specialize the network with respect to the target input. The specialization is actually the injection of an *a priori* knowledge about the representation of the

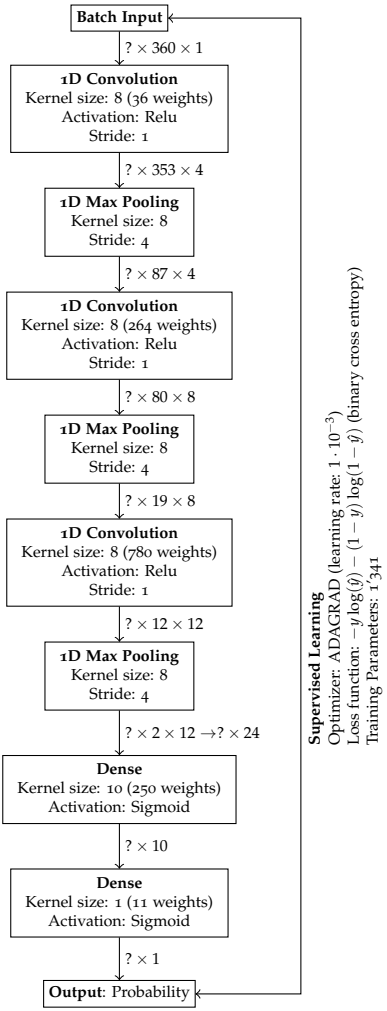



**Figure 5.2:** The classifier structure and the training parameters

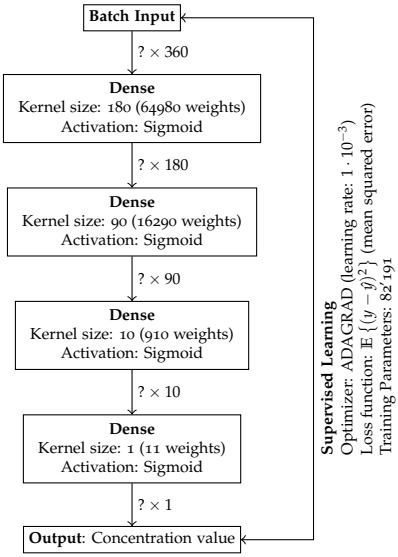



**Figure 5.3:** The classifier structure and the training parameters

input data. In case of convolution, a strong emphasis is given to the spatial distribution of values in the input vector.

The convolution operation[1] uses *kernels* to generate an output that is referred to as *feature map*. The elements of the feature map are evaluated as follows:

$$\mathbf{m}_k = (\mathbf{x} * \mathbf{w})_k = \sum_{i=1}^{\dim(\mathbf{w})} \mathbf{x}_{k+i}\, \mathbf{w}_k \tag{5.9}$$

A sample of the operation is presented in Fig. 5.4. A single convolutional layer may employ multiple kernels. The usage of more kernels allows to search different features on the same input space. If the input of a layer is in $\mathbb{R}^{m\times 1}$ and the convolution operation is performed with a kernel that is $\mathbb{R}^{a\times b}$, the output vector is of dimensions $\mathbb{R}^{\tilde{m}\times b}$,

[1]The *convolution* operation is actually a *cross-correlation* operation in many implementations. They are both called convolution by convention.

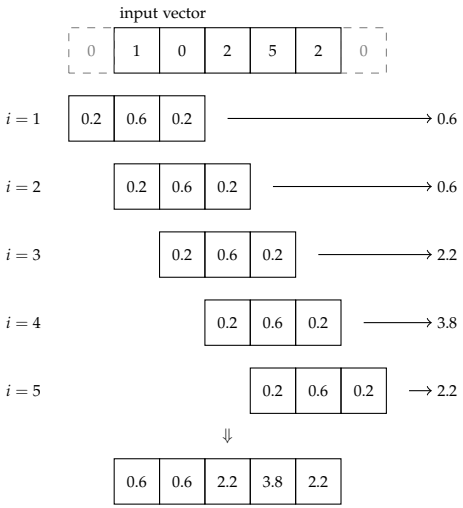



**Figure 5.4:** Convolution operation example, with *same* padding

where $b$ is the new feature depth[2].

But the importance of the convolution lies in the properties that emerge from this kind of layer [40]:

**sparse interaction** in a common fully connected layer all input are forced to interact with all the output units. But in a convolutional layers the kernel are smaller than the input, in order to learn to disclose little feature in big inputs. Fewer parameters are stored to accomplish the task and this reduces the memory requirement for a model, improving its efficiency.

**parameters sharing** in a fully connected network, each weight applies exactly once for each input, while in the convolution the weight applied to an input is bound to the same weight applied in another position of the input, due to the correlation.

**equivariance to translation** is the robustness with respect to small

[2]The first dimension $m$ may change ($m \neq \tilde{m}$) due to *padding*.

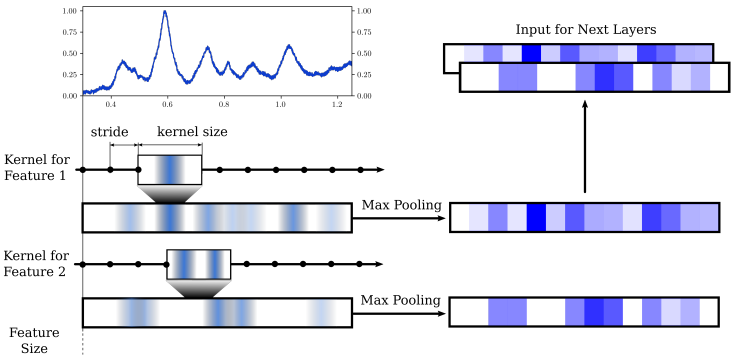



**Figure 5.5:** A complete example of 1D convolution and max pooling

translation of particular feature in the input data, and directly derives from the parameter sharing. This result is also reinforced by the application of a *max pooling layer*.

Fig. 5.5 represents the output of a very first layer of the classifier. The convolution is typically applied to images, where the grid like topology is related to the pixel composition that characterize a raster image. The formulation of the convolution operation is slightly different, and the accumulation is performed along both dimension of the image.

The convolution is not the only peculiar layer that insert a infinitely strong priori interpretation for input data in the network. Many other layers may be applied to a network to impose some preference on the interpretation of inputs.

### 5.2.3 *Generation of Examples*

There is a great advantage in training the networks if a very big database of labeled examples is available. In the context of this application, it is possible to programmatically generate a new labeled

example starting from the theoretical peaks of phases. The original peaks are retrieved from ICCD PDF$^{TM}$ cards, in the form of angles ($2\theta$) and intensities ($I$).

The generation procedure introduces different physical-based aberrations in order to present to the network an example as realistic as possible, allowing the network to understand which is the good representation to identify and regress the presence of a specific phase. For the generation of the input of an example (pattern):

1. a random shift in the $2\theta$ domain is applied, both for single ($\mathbf{b}$) and concurrent peaks ($a\mathbf{1}^{\dim(2\theta)}$):

$$(2\theta)' = (2\theta) + \mathbf{b} + a\mathbf{1}^{\dim(2\theta)} \tag{5.10}$$

   where[3]:

$$\begin{aligned} a &\sim \mathcal{U}(\Delta\theta_{\min}, \Delta\theta_{\max}) \\ \mathbf{b} &\sim \{\mathcal{U}(\Delta\theta_{\min}, \Delta\theta_{\max})\}^{\dim(2\theta)} \end{aligned}$$

2. after a projection in $d^\star = 2\sin(\theta')/\lambda$, a secondary shift is applied:

$$(d^\star)' = (2\theta) + \mathbf{b} + a\mathbf{1}^{\dim(d^\star)} \tag{5.11}$$

   with $a$ and $\mathbf{b}$ defined as above. $\lambda$ is the wavelength of the source;

3. the pattern for a single phase is generated as a linear combination of Cauchy and Normal probability density functions, where parameters define peaks heights and shape:

$$\phi(d) = \sum_{d_i \in (d^\star)'} I(d_i)\,(\,\nu\mathcal{N}(d, d_i, \sigma_1) + (1-\nu)\mathcal{C}(d, d_i, \sigma_2)\,) \tag{5.12}$$

4. a noise proportional to the square root of the local intensity is

[3] $\mathcal{U}$ is the uniform distribution

added to the pattern:

$$\phi'(d) = \phi(d) + \sqrt{\phi(d)}\, n(d) \quad n(d) \sim \mathcal{U}(0,1) \tag{5.13}$$

Everything can be collected in a single operation:

$$\phi'(d) = \Phi(d, 2\theta, I, \lambda) \tag{5.14}$$

5. mixture of phases are combined as a weighted summation, were the weight is proportional to concentration. Also a white noise and a background is added to obtain the final signal:

$$\phi''(d) = \beta(d) + \sum_{P_i \in \mathbf{P}} \gamma(P_i)\, \Phi(d, 2\theta(P_i), I(P_i), \lambda(P_i)) \tag{5.15}$$

where $\gamma(P_i)$ is the concentration of each phase, such that

$$\sum_{P_i \in \mathbf{P}} \gamma(P_i) = 1,$$

and $\beta(d)$ is the background and the white noise contribution. Different formulation for the background noise may be used: polynomial, exponential, etc. For example:

$$\beta(d) = \frac{a_1}{1 - e^{-d+a_2}} + n(d) \qquad n(d) \sim \mathcal{U}(n_{\min}, n_{\max}) \tag{5.16}$$

The presence of the target phase—i.e. the one for which the ANNs are trained—depends on a sample from the distribution $\mathcal{U}(0,1)$. If the sample is greater than 0.5, the target phase is included in the pattern. When generating pattern for the regressor the target phase is *always* present.

6. the final example is normalized with respect to the maximum value:

$$\phi'''(d) = \frac{\phi''(d)}{\max(\phi''(d))} \tag{5.17}$$

this is necessary for keeping the learning problem numerically stable.

The generation of the output of an example, necessary for the supervised training is quite simple. For the classifier:

$$\bar{y}_{\text{cl}} = \begin{cases} 1 & \text{if } P_{\text{target}} \in \mathbf{P} \\ 0 & \text{otherwise} \end{cases} \tag{5.18}$$

while for the regressor it is the concentration:

$$\bar{y}_{\text{rg}} = \gamma(P_{\text{target}}) \tag{5.19}$$

The combination of pattern and output are used to train the networks. Some patterns are used to validate the network.

### 5.2.4 *Inference Examples*

In the following section some inference examples, Figg. 5.7–5.10, are reported for different concentrations of positively labeled validation input. The images contain the actual input for the network (the pattern as seen by a diffractometer) and the distribution of theorical peaks for $BaSO_4$. The performance of both classifier and regressor are proportionally dependent to the concentration, with a drop for concentration below 20% for classification, as reported in Fig 5.6. The drop is due to the fact that other phases and background noise mask the presence of the target phase. It is easy to find some situations where the classifier is wrong for low concentrations, as reported in Fig. 5.11

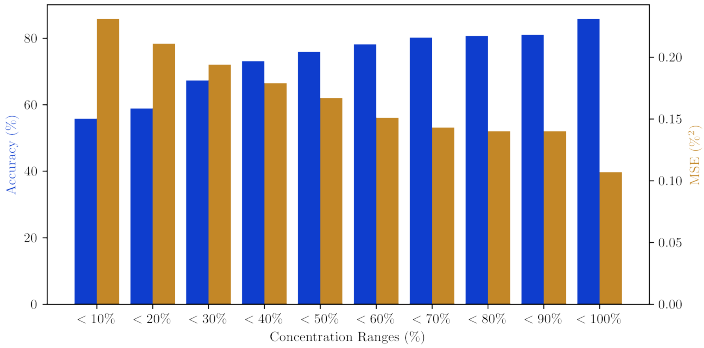



**Figure 5.6:** The graph contains a summary of the performance for the two networks with respect to concentration

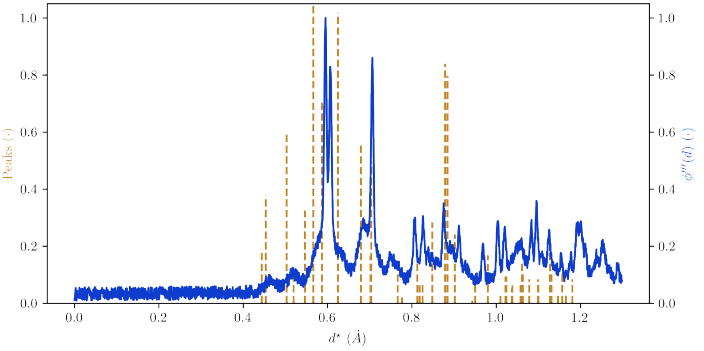



**Figure 5.7:** Pattern for $BaSO_4$ with concentration at 16%. The classifier perceives the phase with a confidence of 71%, and the regressor confirms a concentration of 19%

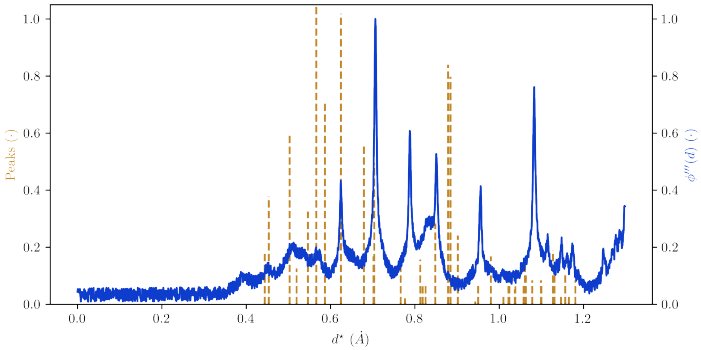



**Figure 5.8:** Pattern for $BaSO_4$ with concentration at 25%. The classifier perceives the phase with a confidence of 80%, and the regressor confirms a concentration of 26%

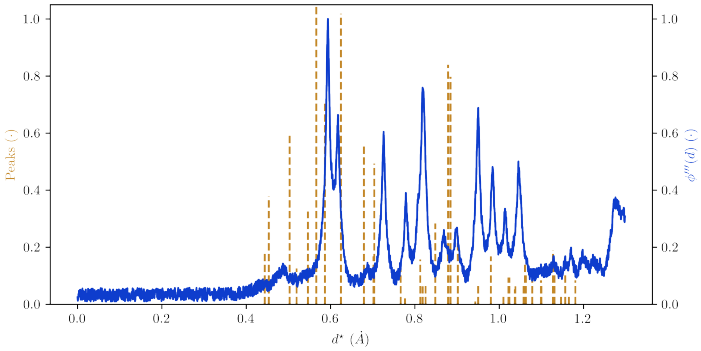



**Figure 5.9:** Pattern for $BaSO_4$ with concentration at 40%. The classifier perceives the phase with a confidence of 79%, and the regressor confirms a concentration of 39%

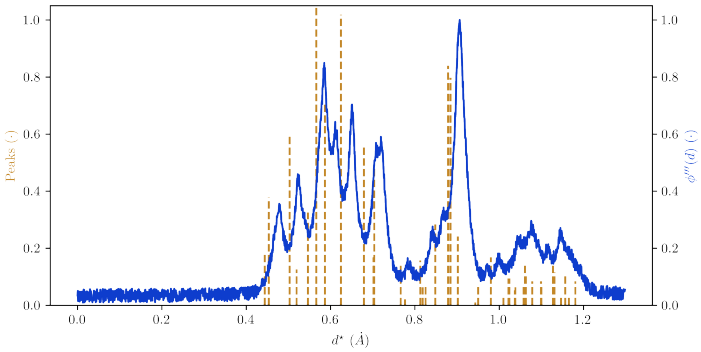



**Figure 5.10:** Pattern for $BaSO_4$ as only phase. The classifier perceives the phase with a confidence of 89%, and the regressor confirms a concentration of 98%

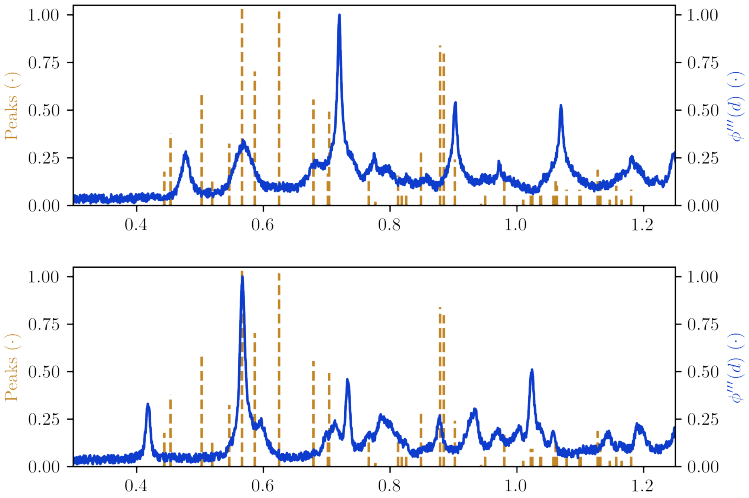



**Figure 5.11:** Pattern for $BaSO_4$ with concentration at 11% and 12%. In the first case the classifier fails in perceiving the phase (confidence of 7%), thus the regressor is not called. In the second case the classifier identifies the phase (with a low confidency score of 67%) but the regressor fails, reporting a too high concentration (21%)

## 5.3 A MATTER OF REPRESENTATION

### 5.3.1 *Representation and Transfer Learning*

The lesson taken from the previous example it is quite straightforward. The algorithm learns a representation for the data through the observation of the couple $[\mathbf{x}, \bar{y}]$. In principle, it is possible to identify in the very first layers of the network a transformation from one feature space to another, but the algorithm is forced to learn a particular kind of transformation due to the strong effect of the label $\bar{y}$. This is a further reason why the two networks, the classifier and the regressor in the previous example, do not share the same basic representation. The feature transformation identified by the classifier probably loses the information necessary to the concentration reconstruction.

In general it is profitable that the algorithm learns a good representation, i.e. the representation that allows it to perform the required task with the best performance possible. This approach is so important that, in a common feed forward network for classification, the initial layers transport data in a feature space where the very last layers can linearly separate the data [69]. On this basis it is possible to notice an interesting property of the network: the *transfer learning*. Lets say a single learner is put in front of two different tasks, in which it is known that some initial layer may share the same representation transformation, but only for the very first task the dataset is sufficiently large for performing a supervised training. The machine first layers, after a training, can be shared to a new network, where last layer are trained on the smaller dataset while leveraging on an already learned representation. It is the case of machines that works on images: it is quite common that the very first layers in the network contains weights that detect edges and primitive geometries [64], and those layers are shared across different tasks. This technique transfers the ability to perceive and extract a semantic knowledge of an

input space from a network to another. This concept is at the very core of the *transfer learning*. This is also one of the reason why neural networks usually operates at perception level (near the sensory raw input). And in fact several side effects of this techniques were used in Section 5.2: the networks learn:

- to de-noise the data
- to disentangle the different property of each phase
- to clusterize the input domain

and all this process starts from the very first layer.

### 5.3.2 *Good Representations*

What characterize a good representation and how to achieve it? This is actually an open challenge but Ref. [40] and Ref. [7] identify several clues that may help a learner in the creation of the transformation:

**Regularization** the representation may be constrained to be locally smooth, and the learner may be forced to be robust with respect to minor disturbances in the input space.

**Manifold Learning** the idea is that data with high dimensionality and a shared representation accumulates near loci, namely manifolds, that can be described with a lower dimensional feature space. Learning the structure of a manifold is an important task, since such transformation can be seen as a hierarchical organization of the data.

**Sparsity** a representation wants to be sparse when the majority of the directions of the feature space are not relevant for the description of the input. This is common in the case of networks for objects detection. The direction should be different from

zero only if the feature to detect is actually present in the input, while all the others should be nil.

**Orthogonality** the representation should disentangle the causal factors in the data in such a way they are independent. Usually an orthogonal representation is also the one that factorizes the data and describes them in the most compact space. For example, a very orthogonal representation identifies a marginal dependence: $P(\mathbf{h}) = \prod_i P_i(h_i)$.

**Supervised Learning** a learning strategy with labels is a strong a-priori that creates a peculiar representation that is completely guided by the provided supervised output, and tries to mimic it.

### 5.3.3 *Autoencoders and Representation*

An *autoencoder* [15] is a particular network composed by two structures: an *encoder* that applies the function:

$$E : \mathbb{R}^{\dim(\mathbf{x})} \mapsto \mathbb{R}^{\dim(\mathbf{h})} \quad E(\mathbf{x}) = \mathbf{h} \tag{5.20}$$

and a decoder that applies the function:

$$D : \mathbb{R}^{\dim(\mathbf{h})} \mapsto \mathbb{R}^{\dim(\mathbf{x})} \quad E(\mathbf{h}) = \mathbf{r} \tag{5.21}$$

The complete autoencoder applies the function:

$$AE : \mathbb{R}^{\dim(\mathbf{x})} \mapsto \mathbb{R}^{\dim(\mathbf{x})} \quad AE(\mathbf{x}) = D(E(\mathbf{x})) = \mathbf{r} \tag{5.22}$$

where $\mathbf{x}$ is the input, $\mathbf{h}$ the *code*, and $\mathbf{r}$ is the reconstruction. Sometimes the input is perturbed in order to make the reconstruction robust with respect to noise (*denoising autoencoder*). This architecture is usually trained to mimic the identity function, in such a way the reconstructed output is only an approximation of the input. One of the

most used constraint is a dimensionality reduction, in such a way:

$$\dim(\mathbf{x}) \gg \dim(\mathbf{h}) \tag{5.23}$$

and the training is guided by a loss function that minimizes a norm from the input and the reconstruction:

$$\mathcal{L}(\mathbf{x}|AE) = ||\mathbf{x} - \mathbf{r}||_L = ||\mathbf{x} - AE(\mathbf{x})||_L \tag{5.24}$$

The training does not use labeled examples, and thus is referred as a unsupervised training. This kind of autoencoders are said to be *undercomplete*. There is a balance between dimensions for the code and its representation capacity, since it may fall in a representation that can reconstruct all input examples using only a simple index (that is the example index): if the autoencoder has a capacity which is too big may learn nothing useful about the representation of the data. This is the reason why the autoencoders are usually regularized—i.e. some additional constraints are applied to the basic loss function.

There are different objectives to add to the loss function that changes the kind of representation that is learned:

- the autoencoder may be forced to learn a sparse coding when a penalty function that enforces a sparsity for the code representation (*sparse autoencoder*) is applied:

$$\Omega(\mathbf{h}) = \lambda \sum_i |h_i| \tag{5.25}$$

- alongside the de-noising strategy, it is possible to regularize the code representation by limiting during training the variation of the code with respect to the input (*contractive autoencoder*):

$$\Omega(\mathbf{h}, \mathbf{x}) = \lambda \sum_i ||\nabla_{\mathbf{x}} h_i||^2 \tag{5.26}$$

In any case autoencoders exploit the accumulation of data along

lower dimensional manifolds, in order to actually learn the structure of the manifold. This structure characterizes how an input may change, and such form of variation is understood mathematically as the tangent planes of the manifold in the point $\mathbf{x}$. From this principle it is possible to get a rule to dimension the capacity of the network: the autoencoder should be capable only to learn variations that reconstruct an input example. The set of variations are the real representation of data, and are also known as the *embeddings*.

It must be clear that all the discussion relies on the hypothesis that the manifold is smooth enough [8]. But the properties of the autoencoders are already been employed to information retrieval task, after the application of dimensionality reduction to input data, in an application called *semantic hashing* [69].

### 5.3.4 *Autoencoder Library*

In the pursuit of a better understanding of the properties of autoencoders, the `cae` library for experiments has been written in *Tensorflow* [43]. The library is actually an implementation of a *convolutional autoencoder* that operates mainly on images. It allows to:

- specify completely the structure of each layer;
- insert specific training loss, alongside with additional regularization;
- perform *greedy layer-wise* training [47];
- perturbate the input;
- use transposed encoder weights as decoder weights;
- perform a differential learning [44];
- inspect each layer output and trained weights.

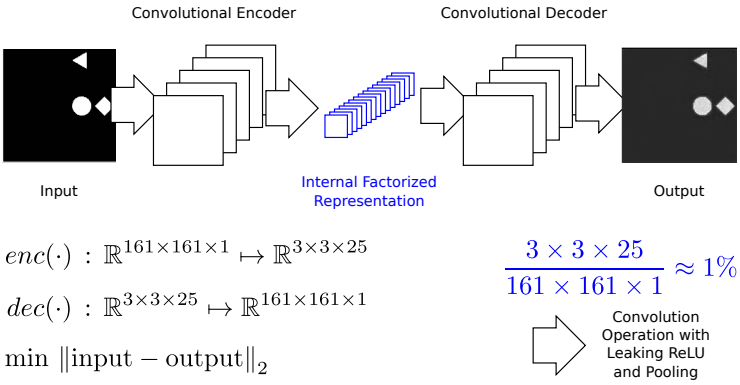



**Figure 5.12:** A test application for the convolutional autoencoder, using the `cae` library. The black figures are a real input and a reconstructed output.

The library has been written as basis for future works in the application of unsupervised learning to industrial dataset, in order to derive favorable representation and applications, and it is freely attainable from GitHub[4].

A very first application, to test the capability of the library in compressing the information is reported as example in Fig 5.12. The application constraints an input $\mathbf{x} \in \mathbb{R}^{161\times161\times1}$ to a code in the domain $\mathbf{h} \in \mathbb{R}^{3\times3\times25}$. The information are compressed by 99%, but the output can still be reconstructed. The factorized representation has dimensions near to the dimension of the actual information present in the images (3 different shapes in possible 25 positions, never overlapped), but since in the example no regularization is applied to the training the inner code is quite dense. But still, the library proved its efficacy.

[4] `cae`, a Convolutional AutoEncoder Library: `https://github.com/MatteoRagni/cae`

# 6 Conclusions

Filling the Industry 4.0 leap is and will be a long journey made by transformations. At the end of the road it is possible to forsee a new generation of industrial automated assets that interact with operators. Those assets overlook tasks that can be regarded as contingential with respect to the overall production process. Those tasks, when carried out by human operators, can be considered unsafe, unpleasent and even exhausting.

On the same line, human operators are deeply connected and extensively informed. The newer human-machine interfaces engage increasingly the sensorial perception of the operators, through augmented interfaces, and the exchange of information is bi-directional.

The design of a product moves its focus away from the propedeutic production process. The identification of the production goals and the coordination of the assets is decentralized on a *distributed intelligence* that operates at different abstraction layers. The highest levels operate on the definition of sequence of goals that characterize a specific production process, while lower levels operates on the optimization of single step of the sequence, or specific task, perceiving and adapting their actions with respect to real operative conditions.

There is no technology that answers directly to all the challenges proposed by the Industry 4.0 strategic plan, but instead it is possible to apply a synergy of *enabling technologies*. Some of such enabling technologies have been identified and their application is introduced in the manufacturing world.

The work presents an industry-ready framework for the application of augmented reality and optimal control to machine tools setup and maintenance that engages specifically the *information transparency* and the *technical assistance*. Ironically, computer numerical control machines, even of recent production, employ as human-machine interface the G-CODE language (ISO 6983), an 1980 standard. To overpass the specific limitations of the language, the different machine manufacturers have to expand it with not standard features [98]. The cost to pay is an extremely fragmented interfacing scenario, where each manufacturer employs its own dialect that requires a specific formation for the operator.

Through the application of an augmented interface, a human operator can use the system to inspect part-programs, to program touch-probes faster, and to check machines for maintenance purposes, while machine manufacturers and technical offices have an easier way to distribute documentations and technological information. The user is not engaged on a semantic level (through a programming language), but instead on a sensory level (through visual stimula): the operators can see the world as perceived by the machine and input directives that are based upon concrete objects in the real world.

To improve the efficiency of existing machines, an optimal control strategy has been implemented to perform minimum time optimization to existing part-program in pre-processing, alongside with specific tools to easily develop similar strategies. The approaches have been validated through experimental activities.

However, this futuristic scenario is contrasted by the intrinsic reality of the European manufacturing industry, characterized by a multitude of medium and small industries with dated machines that do not even support a network connection. That is the reason why all the technologies introduced always considered the existing manufacturing scenario as an implementation constraint.

What actually remains a completely open topic is the *interoperabil-*

*ity* and the *decentralized decisions*, for which an higher autonomous level is required. Indeed, before thinking about the decision making process, a strategy should be envisioned to actually extract semantic information from the big amount of raw data perceived from sensors. Even if the very last two years presented alternatives—e.g. dropout, leaky rectifing linear units, etc.—that outperform shared representational methodologies, those methods still require quite a big volume of labeled data to truly achieve the results, and this data are not available in the industrial practice.

The unsupervised learning of the manifold structures that describe the data is extremely interesting. As initial step in this direction, a library to build autoencoders has been implemented and tested on experimental dataset for feature reduction. However, the results of the Deep Learning field has been applied to one of the many industrial problem, proving its efficacy with success.

It will be a long journey.

# Bibliography


[1] Y. Altintas and N.A. Erol. Open architecture modular tool kit for motion and machining process control. *CIRP Annals - Manufacturing Technology*, 47(1):295–300, 1998.

[2] Y. Altintas and W. K. Munasinghe. A hierarchical open-architecture CNC system for machine tools. *Annals of CIRP*, 43(1):349–354, 1994.

[3] George H Amber and Paul S Amber. *Anatomy of automation*. Prentice-Hall, 1962.

[4] Apple Inc. SceneKit Framework. `https://developer.apple.com/scenekit/`, 2016.

[5] Michael Bartholomew-Biggs, Steven Brown, Bruce Christianson, and Laurence Dixon. Automatic differentiation of algorithms. *Journal of Computational and Applied Mathematics*, 124(1):171–190, 2000.

[6] Ravil Bayramgalin. Symbolic. `https://github.com/brainopia/symbolic`, 2012. Online; commit: `bbd588e8676d5bed0017a3e1900ebc392cfe35c3`.

[7] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[8] Yoshua Bengio and Martin Monperrus. Non-local manifold tangent learning. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 129–136. MIT Press, 2005.

[9] Xavier Beudaert, Sylvain Lavernhe, and Christophe Tournier. Feedrate interpolation with axis jerk constraints on 5-axis NURBS and G1 tool path. *International Journal of Machine Tools and Manufacture*, 57(0):73–82, 2012.

[10] Francesco Biral, Enrico Bertolazzi, and Paolo Bosetti. Notes on numerical methods for solving optimal control problems. *IEEJ Journal of Industry Applications*, 5(2):154–166, 2016.

[11] Gabriele Bleser and Didier Stricker. Advanced tracking through efficient image processing and visual–inertial sensor fusion. *Computers & Graphics*, 33(1):59–72, 2009.

[12] I. Bondrea and R.E. Petruse. Augmented reality - an improvement for computer integrated manufacturing. *Advanced Materials Research*, 628:330–336, 2013.

[13] Paolo Bosetti and Enrico Bertolazzi. Feed-rate and trajectory optimization for CNC machine tools. *Robotics and Computer-Integrated Manufacturing*, 30(6):667–677, 12 2014.

[14] Léon Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998. revised, oct 2012.

[15] H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4):291–294, 9 1988.

[16] Gary Bradski et al. The opencv library. *Doctor Dobbs Journal*, 25(11):120–126, 2000.

[17] A.E. Bryson and Y.C. Ho. *Applied optimal control: optimization, estimation, and control*. Halsted Press book'. Hemisphere Publishing Company, 1975.

[18] J.C. Butcher. *Numerical Methods for Ordinary Differential Equations, Second Edition*. John Wiley & Sons, 2008.

[19] S.a Büttner, O.a Sand, and C.b Röcker. Extending the design space in industrial manufacturing through mobile projection. In *MobileHCI 2015 - Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct*, pages 1130–1133, 2015.

[20] Ondrej Certik, Dale Lukas Peterson, Thilina Bandara Rathnayake, et al. Symengine. `https://github.com/symengine/symengine.rb`, 2016. Online; commit: `8cf9e08c972085788c17da9f4e9f22898e79d93b`.

[21] J.W.S.a Chong, S.K.c Ong, A.Y.C.a c Nee, and K.a b Youcef-Youmi. Robot programming using augmented reality: An interactive method for planning collision-free paths. *Robotics and Computer-Integrated Manufacturing*, 25(3):689–701, 2009.

[22] Joel S Cohen. *Computer algebra and symbolic computation: Mathematical methods*. Universities Press, 2003.

[23] David W Collins and Doreen Kimura. A large sex difference on a two-dimensional mental rotation task. *Behavioral neuroscience*, 111(4):845, 1997.

[24] S.a Ćuković, G.a Devedžić, F.b Pankratz, K.c Baizid, I.d Ghionea, and A.a Kostić. Augmented reality simulation of cam spatial tool paths in prismatic milling sequences. *IFIP Advances in Information and Communication Technology*, 467:516–525, 2015.

[25] B.L. DeCost, H. Jain, A.D. Rollett, and E.A. Holm. Computer vision and machine learning for autonomous characterization of am powder feedstocks. *JOM*, 69(3):456–465, 2017.

[26] J. Dong and J. A. Stori. Bidirectional scan algorithm for constrained feed-rate optimization. *Journal of Dynamic Systems, Measurement, and Control*, 128:379–390, 6 2006.

[27] J. Dong and J.A. Stori. A generalized time-optimal bidirectional scan algorithm for constrained feed-rate optimization. *Journal*

*of Dynamic Systems, Measurement and Control, Transactions of the ASME*, 128(2):379–390, 2006.

[28] Jingyan Dong, P.M. Ferreira, and J.A. Stori. Feed-rate optimization with jerk constraints for generating minimum-time trajectories. *International Journal of Machine Tools and Manufacture*, 47(12–13):1941–1955, 2007.

[29] A.a Doshi, R.T.a Smith, B.H.a Thomas, and C.b Bouras. Use of projector based augmented reality to improve manual spot-welding precision and accuracy for automotive manufacturing. *International Journal of Advanced Manufacturing Technology*, pages 1–15, 2016.

[30] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

[31] V. Elia, M.G. Gnoni, and A. Lanzilotto. Evaluating the application of augmented reality devices in manufacturing from a process point of view: An ahp based model. *Expert Systems with Applications*, 63:187–197, 2016.

[32] H.C. Fang, S.K. Ong, and A.Y.C. Nee. Interactive robot trajectory planning and simulation using augmented reality. *Robotics and Computer-Integrated Manufacturing*, 28-2:227–237, 2012.

[33] H.C. Fang, S.K. Ong, and A.Y.C. Nee. Robot path and end-effector orientation planning using augmented reality. In *Procedia CIRP*, volume 3-1, pages 191–196, 2012.

[34] J.G. Ferreira and A. Warzecha. An application of machine learning approach to fault detection of a synchronous machine. In *2017 International Symposium on Electrical Machines, SME 2017*, 2017.

[35] M. Fiorentino, A.E. Uva, M. Gattullo, S. Debernardis, and G. Monno. Augmented reality on large screen for interactive

maintenance instructions. *Computers in Industry*, 65(2):270–278, 2014.

[36] David Flanagan and Yukihiro Matsumoto. *The ruby programming language*. O'Reilly Media, Inc., 2008.

[37] Y. Fu, Y. Zhang, H. Qiao, D. Li, H. Zhou, and J. Leopold. Analysis of feature extracting ability for cutting state monitoring using deep belief networks. In *Procedia CIRP*, volume 31, pages 29–34, 2015.

[38] A. Gasparetto, A. Lanzutti, R. Vidoni, and V. Zanotto. Experimental validation and comparative analysis of optimal time-jerk algorithms for trajectory planning. *Robotics and Computer-Integrated Manufacturing*, 28(2):164–181, 2012.

[39] WW Gilbert. Economics of machining. *Machining theory and practice*, pages 465–485, 1950.

[40] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[41] Sandra G Hart. Nasa-task load index (nasa-tlx); 20 years later. In *Proceedings of the human factors and ergonomics society annual meeting*, volume 50-9, pages 904–908. Sage Publications, 2006.

[42] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[45] N. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, 2002.

[46] G. E. Hinton and T. J. Sejnowski. Learning and relearning in boltzmann machines. In David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*, pages 282–317. MIT Press, Cambridge, MA, USA, 1986.

[47] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.

[48] Lei Hou, Xiangyu Wang, Leonhard Bernold, and Peter ED Love. Using animated augmented reality to cognitively guide assembly. *Journal of Computing in Civil Engineering*, 27(5):439–451, 2013.

[49] Javad Jahanpour and Behnam Imani. Real-time P-H curve CNC interpolators for high speed cornering. *The International Journal of Advanced Manufacturing Technology*, 39:302–316, 2008.

[50] Ariacutty Jayendran. CNC machines (CNC Maschinen). In *Mechanical Engineering*, pages 177–192. Teubner, 2006.

[51] N.-M. Jozef, J. Miroslav, and N.-M. Ludmila. Augmented reality aided control of industrial robots. *Advanced Materials Research*, 1025-1026:1145–1149, 2014.

[52] Bing-Feng Ju, Xiaolong Bai, Jian Chen, and Yaozheng Ge. Design of optimal fast scanning trajectory for the mechanical scanner of measurement instruments. *Scanning*, 2013.

[53] Serope Kalpakjian, Steven R Schmid, and Chi-Wah Kok. *Manufacturing processes for engineering materials*. Pearson-Prentice Hall, 2008.

[54] Hirokazu Kato. Artoolkit: library for vision-based augmented reality. *IEICE, PRMU*, 6:79–86, 2002.

[55] John Lees-Miller. Rucas. `https://github.com/jdleesmiller/rucas`, 2010. Online; commit: `047a38b541966482d1ad0d40d2549683cf193082`.

[56] David G. Lowe. Fitting parameterized three-dimensional models to images. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 1(5):441–450, 1991.

[57] Héctor Martínez, Seppo Laukkanen, and Jouni Mattila. A new hybrid approach for augmented reality maintenance in scientific facilities. *International Journal of Advanced Robotic Systems, Manuel Ferre, Jouni Mattila, Bruno Siciliano, Pierre Bonnal (Ed.), ISBN*, 1729:8806, 2013.

[58] B.a Meden, S.a Knodel, and S.b Bourgeois. Markerless augmented reality solution for industrial manufacturing. In *ISMAR 2014 - IEEE International Symposium on Mixed and Augmented Reality - Science and Technology 2014, Proceedings*, pages 359–360, 2014.

[59] A.a Monroy Reyes, O.O.a Vergara Villegas, E.b Miranda Bojórquez, V.G.b Cruz Sánchez, and M.a Nandayapa. A mobile augmented reality system to support machinery operations in scholar environments. *Computer Applications in Engineering Education*, 24(6):967–981, 2016.

[60] A.Y.C. Nee and S.K. Ong. Virtual and augmented reality applications in manufacturing. In *IFAC Proceedings Volumes (IFAC-PapersOnline)*, pages 15–26, 2013.

[61] Alex Olwal, Jonny Gustafsson, and Christoffer Lindfors. Spatial augmented reality on industrial cnc-machines. In *Proc. SPIE*, volume 6804, pages 6804–9, 2008.

[62] S.K. Ong, M.L. Yuan, and A.Y.C. Nee. Augmented reality applications in manufacturing: A survey. *International Journal of Production Research*, 46(10):2707–2742, 2008.

[63] S.K.b Ong, Y.a Pang, and A.Y.C.a b Nee. Augmented reality aided assembly design and planning. *CIRP Annals - Manufacturing Technology*, 56(1):49–52, 2007.

[64] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6 2014.

[65] E. Ostermeyer, C. Danjou, A. Durupt, and J. Le Duigou. Retrieval of manufacturing knowledge using machine learning - a review. In *Advances in Transdisciplinary Engineering*, volume 6, pages 515–521, 2017.

[66] PTC Inc. Product Lifecycle Management (PLM) Software. `http://www.ptc.com/product-lifecycle-management`, 2016. accessed on April 17, 2018.

[67] H.a b Ramírez, E.a Mendoza, M.a Mendoza, and E.b González. Application of augmented reality in statistical process control, to increment the productivity in manufacture. In *Procedia Computer Science*, volume 75, pages 213–220, 2015.

[68] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[69] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[70] P Schaumlöffel, M Talha, D Gorecky, and G Meixner. Augmented reality applications for future manufacturing. *Proceedings of the 5th Manufacturing Science and Education-MSE*, 1(5):2–5, 2011.

[71] Gerald Schweighofer and Axel Pinz. Robust pose estimation from a planar target. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(12):2024–2030, 2006.

[72] B. Sencer, Y. Altintas, and E. Croft. Feed optimization for five-axis CNC machine tools with drive constraints. *International Journal of Machine Tools and Manufacture*, 48(7–8):733–745, 2008.

[73] Amedeo Setti, Paolo Bosetti, and Matteo Ragni. Artool - augmented reality platform formachining setup and maintenance. In *Proceedings of SAI Intelligent Systems Conference (Intellisys) 2016*, volume 2, pages 273–281. Springer, 8 2017.

[74] Zvi Shiller. On singular time-optimal control along specified paths. *IEEE Transactions on Robotics and Automation*, 10(4):561–566, 1994.

[75] Christopher Smith, Aaron Meurer, Mateusz Paprocki, et al. Sympy 1.0. https://doi.org/10.5281/zenodo.47274, 2016. Online; accessed: 2016-10-15.

[76] Suk-Hwan Suh, Seong Kyoon Kang, Dae-Hyuk Chung, and Ian Stroud. *Theory and Design of CNC Systems*. Springer Series in Advanced Manufacturing. Springer, 1st edition, 2008.

[77] F.a Suárez-Warden, E.G.a Mendívil, H.b Ramírez, L.E.b Garza Nájera, and G.b Pantoja. Mill setup manual aided by augmented reality. In *Mechanisms and Machine Science*, volume 25, pages 433–441, 2015.

[78] A.a Syberfeldt, O.a Danielsson, M.a Holm, and L.a b Wang. Dynamic operator instructions based on augmented reality and rule-based expert systems. In *Procedia CIRP*, volume 41, pages 346–351, 2016.

[79] Dassault Systems. Solidworks Model Based Definitions. `http://www.solidworks.it/sw/products/technical-communication/packages.htm`, 2016. accessed on April 17, 2018.

[80] Kazuaki Tanaka, Avinash Dev Nagumanthri, and Yukihiro Matsumoto. mruby–rapid software development for embedded sys-

tems. In *15th International Conference on Computational Science and Its Applications (ICCSA)*, pages 27–32. IEEE, 2015.

[81] Technical Committee : ISO/IEC JTC 1/SC 22. ISO/IEC 30170 – Information technology – Programming languages – Ruby. Standard, International Organization for Standardization, Geneva, CH, 4 2000.

[82] H. Tercan, T.A. Khawli, U. Eppelt, C. Büscher, T. Meisen, and S. Jeschke. Improving the laser cutting process design by machine learning techniques. *Production Engineering*, 11(2):195–203, 2017.

[83] John E Tolsma and Paul I Barton. On computational differentiation. *Computers & chemical engineering*, 22(4):475–490, 1998.

[84] Guido Van Rossum and Fred L Drake. *The Python language reference manual*. Network Theory Ltd., 2011.

[85] B. Van Stein, M. Van Leeuwen, H. Wang, S. Purr, S. Kreissl, J. Meinhardt, and T. Back. Towards data driven process control in manufacturing car body parts. In *Proceedings - 2016 International Conference on Computational Science and Computational Intelligence, CSCI 2016*, pages 459–462, 2017.

[86] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl. Time-energy optimal path tracking for robots: a numerically efficient optimization approach. In *Advanced Motion Control, 2008. AMC '08. 10th IEEE International Workshop on*, pages 727–732, 3 2008.

[87] N.a Vignais, M.b Miezal, G.b Bleser, K.c Mura, D.c Gorecky, and F.a Marin. Innovative system for real-time ergonomic feedback in industrial manufacturing. *Applied Ergonomics*, 44(4):566–574, 2013.

[88] Joachim Von Zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge university press, 2013.

[89] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, 2006.

[90] Andreas Wächter and Carl Laird. Ipopt-an interior point optimizer. `https://projects.coin-or.org/Ipopt`, 2009. Online; accessed: 2016-11-28.

[91] Xiangyu Wang, Peter ED Love, Mi Jeong Kim, Chan-Sik Park, Chun-Pong Sing, and Lei Hou. A conceptual framework for integrating building information modeling with augmented reality. *Automation in Construction*, 34:37–44, 2013.

[92] ZB Wang, SK Ong, and AYC Nee. Augmented reality aided interactive manual assembly design. *The International Journal of Advanced Manufacturing Technology*, 69(5-8):1311–1321, 2013.

[93] Bertram Eugene Warren. *X-ray Diffraction*. Courier Corporation, 1969.

[94] J André C Weideman. Numerical integration of periodic functions: A few examples. *The American mathematical monthly*, 109(1):21–36, 2002.

[95] K. Weinert, A. Zabel, E. Ungemach, and S. Odendahl. Improved nc path validation and manipulation with augmented reality methods. *Production Engineering*, 2(4):371–376, 2008.

[96] Tomasz Wójcicki. Supporting the diagnostics and the maintenance of technical devices with augmented reality. *Diagnostyka*, 15(1):43–47, 2014.

[97] D. Wu, C. Jennings, J. Terpenny, R.X. Gao, and S. Kumara. A comparative study on machine learning algorithms for smart manufacturing: Tool wear prediction using random forests. *Journal of Manufacturing Science and Engineering, Transactions of the ASME*, 139(7), 2017.

[98] Xun Xu. Machine tool 4.0 for the new era of manufacturing. *The International Journal of Advanced Manufacturing Technology*, 92(5):1893–1900, Sep 2017.

[99] J. Zhang, S.K. Ong, and A.Y.C. Nee. A multi-regional computation scheme in an ar-assisted in situ cnc simulation environment. *CAD Computer Aided Design*, 42(12):1167–1177, 2010.

[100] Ke Zhang, Chun-Ming Yuan, Xiao-Shan Gao, and Hongbo Li. A greedy algorithm for feedrate planning of CNC machines along curved tool paths with confined jerk. *Robotics and Computer-Integrated Manufacturing*, 28(4):472–483, 2012.

# Personal Bibliography


[MR1] Enrico Bertolazzi, Francesco Biral, and Matteo Ragni. Gtoc 9: Results from university of trento (team elfman). *Acta Futura*, (11):79–90, jan 2018.

[MR2] Paolo Bosetti and Matteo Ragni. Milling part-program preprocessing for jerk-limited, minimum-time toolpaths based on optimal control theory. *SAMCON 2015*, 3 2015.

[MR3] Paolo Bosetti and Matteo Ragni. Milling part program preprocessing for jerk-limited, minimum-time tool paths based on optimal control theory. *IEEJ Journal of Industry Applications*, 5(2):53–60, 2016.

[MR4] Paolo Bosetti, Matteo Ragni, and Matteo Leoni. Modern machine-learning tools for crystallography. 73:C562–C562, 12 2017.

[MR5] Laura Lugli, Matteo Ragni, Laura Piccardi, and Raffaella Nori. Hypermedia navigation: Differences between spatial cognitive styles. *Computers in Human Behavior*, 66:191–200, 2017.

[MR6] Matteo Ragni. Avionic perception-action model for uav aimed at avalanche buried searching. In Genova University Press, editor, *Conference procedings AIMETA 2015 - XXII Conference - The Italian Association of Theoretical and Applied Mechanics*, volume 1, page 311, 9 2015. Abstract.

[MR7] Matteo Ragni. Mr.CAS: A minimalistic (pure) ruby cas for fast prototyping and code generation. *SoftwareX*, 6:128–134, 2017.

[MR8] Matteo Ragni, Matteo Perini, Amedeo Setti, and Paolo Bosetti. Artool zero: Programming trajectory of touch-probes using augmented reality. *Computers and Industrial Engineering (IN REVIEW - not yet published)*, 2017.

[MR9] Emilio Sanfilippo, Ferruccio Mandorli, Claudio Masolo, and Matteo Ragni. The interplay between shape and feature representation. In *Joint Ontology Workshops 2017 - SHAPES 4.0*, 9 2017.

[MR10] Amedeo Setti, Paolo Bosetti, and Matteo Ragni. *ARTool—Augmented Reality Human-Machine Interface for Machining Setup and Maintenance*, pages 131–155. Springer International Publishing, Cham, 2018.

[MR11] Amedeo Setti, Paolo Bosetti, and Matteo Ragni. *ARTool-Augmented Reality Platform for Machining Setup and Maintenance*, pages 457–475. Springer International Publishing, Cham, 2018.

[MR12] Andrea Zignoli, Matteo Ragni, Alessandro Fornasiero, Paul Laursen, Federico Schena, and Francesco Biral. Estimating oxygen uptake in cycling using neural network analysis of easy-to-obtain inputs. In Springer Verlag Italia, editor, *Sport Sciences for Health Sismes IX National Congress*, volume 13, page 50, 9 2015.