# High Level Programmable and Flexible Industrial Robotized Cells

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

**Marcos Ferreira**

Doctoral Program in Electrical and Computer Engineering

Supervisor: António Paulo Gomes Mendes Moreira

Co-Supervisor: Joaquim Norberto Cardoso Pires da Silva

A thesis submitted for the degree of

Doctor of Philosophy (PhD)

January 2014

# Abstract

Industrial manipulators play a key role on major production lines. They are one of the most powerful automation solutions for flexible mass production, delivering efficiency and quality at reduced prices. In an entirely different scene, there are the small and medium enterprises (SMEs), which mostly rely on high skilled labour, delivering less quantity but increased customization. In the face of modernization, and to sustain global competitiveness, SMEs are increasingly adopting robotized solutions. Yet, despite of the numerous researches and contributions to improve human-robot interface, programming an industrial robot is still a very technical and time-consuming challenge.

On this scenario, this thesis sets focus on developing new means of intuitive and high level robot programming. Specifically, a new method for robot programming by demonstration is proposed: the factory operator demonstrates a given task, and the robot is automatically programmed to mimic the human operation. This is a framework for human-robot skill transfer, since the know-how of the operator is translated into robot language. Additionally, the final solution is designed and optimized for maximum industrial applicability: use of low-cost hardware, low impact on the industrial process, high success rate and accuracy, fast (real-time) output.

The proposed framework puts together a stereoscopic vision module and a luminous marker. Unlike equivalent motion tracking solutions available, this one uses visible markers. Using a synchronization technique known as the *sincrovision*, cameras and markers and simultaneously triggered. This results in very accurate and robust measures, even under harsh industrial environment conditions. A collection of algorithms, that take advantage of the marker shape and the colour of the markers, is able to reconstruct the

path in six degrees of freedom (6 DoF) — position and orientation. 3D B-Splines and quaternion splines are used to smooth the tracked path before generating the robot program.

A prototype of the system was implemented at an industrial demonstrator, for a spray painting application. The operator was able to demonstrate several painting trajectories using a minimal software interface to start and stop the procedure. Immediately after demonstration (negligible post-processing time), the robot successfully imitated the human moves, and the results were validated by the painters. The company, Flupol, can now instantaneously transfer to the robot the 25 years experience of the oldest painter.

# Resumo

Os manipuladores industriais têm um papel preponderante nas grandes linhas de produção. São uma das soluções mais avançadas tendo em vista o conceito de produção em massa flexível, produzindo com eficiência e qualidade a um custo reduzido. Num outro cenário completamente diferente encontram-se as pequenas e médias empresas, PMEs, que dependem fundamentalmente da mão de obra qualificada dos seus trabalhadores, produzem em menor quantidade mas com maior grau de customização. Na perspectiva de se modernizarem, e de forma a manterem a competitividade no mercado global, as PMEs têm vindo cada vez mais a apostar na robotização dos seus processos. Contudo, apesar de toda a investigação em torno da criação de novos métodos para melhorar o interface homem-robô, a programação de um manipulador industrial ainda requer um elevado nível de conhecimento técnico para além de ser uma tarefa morosa.

É neste cenário que esta tese pretende contribuir, desenvolvendo métodos intuitivos para a programação de robôs industriais. Mais especificamente, é proposto um novo método para programação por demonstração: o operador humano demonstra uma determinada tarefa e o robô é automaticamente programado de modo a que consiga imitar o movimento humano. No seu todo, esta solução consiste numa *framework* para transferência de *know-how* do operador humano para o robô. Mais ainda, a solução final é projetada e otimizada do ponto de vista da aplicabilidade industrial: é usado equipamento de baixo custo, o impacto no processo é mínimo, apresenta elevada exatidão e taxa de sucesso, o resultado é obtido em tempo real.

A *framework* proposta nesta dissertação engloba um sistema de visão estereoscópica e um marcador luminoso. Ao contrário de algumas soluções disponíveis para seguimento de movimento, esta é baseada em marcadores

na gama do visível. Usando uma técnica de sincronização denominada por *sincrovision*, as câmaras e o marcador são acionados simultaneamente, permitindo assim uma leitura precisa e robusta do marcador, mesmo em condições adversas típicas de um ambiente industrial. Um conjunto de algoritmos otimizados para a forma e cor dos marcadores permite fazer uma reconstrução da trajetória com seis graus de liberdade (6 DoF) — posição e orientação. Antes da geração do código para o manipulador industrial, a trajetória é ainda suavizada usando B-Splines em 3D e splines no espaço dos quaterniões.

Um protótipo do sistema desenvolvido foi implementado num demonstrador industrial, numa aplicação de pintura por pulverização. O operador foi capaz de demonstrar várias trajetórias de pintura usando, para tal, uma interface de software onde tem de indicar apenas o início e fim do processo de demonstração. Imediatamente após a demonstração (o tempo de pós-processamento é praticamente nulo), o robô imitou com sucesso o movimento humano, tendo os próprios pintores validado a qualidade da execução. A empresa onde se realizaram os ensaios, Flupol, tem agora a possibilidade de transferir os 25 anos de experiência do seu mais antigo pintor para um manipulador industrial, conseguindo-o fazer de forma praticamente instantânea.

*When you make the finding yourself - even if you're the last person on Earth to see the light - you'll never forget it.*

CARL SAGAN

To my parents...

# Acknowledgements

First and foremost I must thank my supervisors, Professor A. Paulo Moreira and Professor Norberto Pires, for their guidance. To Professor Norberto, whose extensive and remarkable work on the field of robotics served as an inspiration for my research. To Professor A. Paulo, to whom I owe a great debt of gratitude for all this years of advice, support and friendship. His enthusiasm for research, his liveliness and his faithfulness in my work were a major motivation for me.

I would like to thank everyone in the Robotics and Intelligent Systems Unit (Robis) of INESC-TEC. They are the most amazing colleagues and they make of Robis laboratory the most interesting place to work at. A special word of gratitude for my labmates Heber Sobreira, Miguel Pinto, Filipe Santos, André Figueiredo, Luís Rocha and Germano Veiga for sharing their knowledge and friendship over these past years. My sincere thanks to Luís Rocha for all the help, the availability, and for enduring many hours of frustration setting up the industrial solution. Furthermore, I'm also grateful to Germano Veiga for all the insightful discussions, for sharing his experience, and for taking time to examine my work and questioning it. I would also like to give my sincerest thanks to my friends Paulo Malheiros and Professor Paulo Costa. To Paulo Malheiros, I thank him for all his support and for being the most helpful labmate. And also, because I was able to develop my research upon the basis of his work. To Professor Paulo Costa, I thank him for all the guidance, the most useful suggestions and ideas, and for all those late night discussions which proved to be utterly valuable.

My sincere appreciation and gratitude to José Bandeira, Pedro Bandeira and Flupol. They believed and invested in my research since the very beginning. They were patient, supportive and welcomed me at Flupol when the time came to install the final product. They provided means for testing and validating my research as well as to ultimately spread words of its success. I thank all of Flupol's staff, and particularly Rui Gomes, for their help and care.

I will take this opportunity to express my appreciation to all Professors of DEEC/FEUP who contributed to my pursuit of knowledge. I am very fortunate to have crossed paths and to have being taught by such brilliant minds and distinctive characters, encountering some odd personalities along the way.

Finally, my last acknowledgements are the most heartfelt. I ought to thank my close family and friends. Although I do not usually share much of my academic and professional life with them, I still hope to feel their comfort at the end of the day and to embrace their love and friendship in every step of my way. And you Sara... I just can't put it into words; a thousand would be too few. Thank you for being there.

To all, thank you so very much.

Marcos Ferreira,
Porto, January 2014

# Official Acknowledgements

Marcos Ferreira acknowledges FCT (Portuguese Foundation for Science and Technology) for his PhD grant SFRH/BD/60221/2009, without which this research work would not have been possible.

The work presented in this thesis, being part of the Project PRODUTECH PTI (number 13851) - New Processes and Innovative Technologies for the Production Technologies Industry, has been partly funded by the Incentive System for Technology Research and Development in Companies (SI I&DT), under the Competitive Factors Thematic Operational Programme, of the Portuguese National Strategic Reference Framework, and EU's European Regional Development Fund.

Additionally, this work was also sponsored by the project QREN-SIIARI. SIIARI - Sistema para o Incremento da Inteligência Artificial em Robótica Industrial (QREN- Quadro de Referência Estratégico Nacional) 2010-2012.

The author also thanks the FCT for supporting this work through the project PTDC/EME-CRO/114595/2009 - High-Level programming for industrial robotic cells: capturing human body motion.

Marcos Ferreira,

Porto, January 2014

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 On the use of industrial robotic manipulators

Industrial robotic manipulators are, as of today, a fully grown commercial off-the-shelf product that integrate the largest and most complex manufacturing facilities world-wide. These machines are taken as the ultimate automation tool:

A powerful and robust mechanical solution, fit to most demanding operations — heavyweight cargo lifting and handling, ability to sustain hazardous environments, full 24h work-shift.

Top grade electronics using state of the art sensors and motion controllers — high precision, high production quality and speed; integration with the most recent vision, laser and force sensing technologies to satisfy the most heterogeneous systems.

Dedicated software and user interfaces — the manipulators can be reprogrammed on-site, adapted to different work objects, configured with different grippers and tools; in short, they can be recycled on and on through a number of distinct applications.

To these advantages must also be added the cost. As the price varies with the target application and robot payload, the robotic market offers solutions for welding (arc and spot), painting, handling, assembly, palletizing, packing, etc, with payloads ranging from a few Kg to more than a ton. Choosing the right robot for a given application, and for the right price is simple while avoiding expensive "home-made" or overrated machines.

Robots are the way to go in an ever increasing competitive market. But is it always so? Precision, quality and robustness all come with the pack, i.e they are bought with the industrial machine. Flexibility, however, it is an entirely different matter that may cost huge extras.

In fact, that stands true on one face of today's industry: the large scale production, the large companies, the major markets. As depicted in Fig. 1.1, the electrical/electronics (PCs, TVs, cellphones) industry, rubber/plastic, metal/machinery and particularly the automotive industries are the most relevant purchasers of industrial robots.

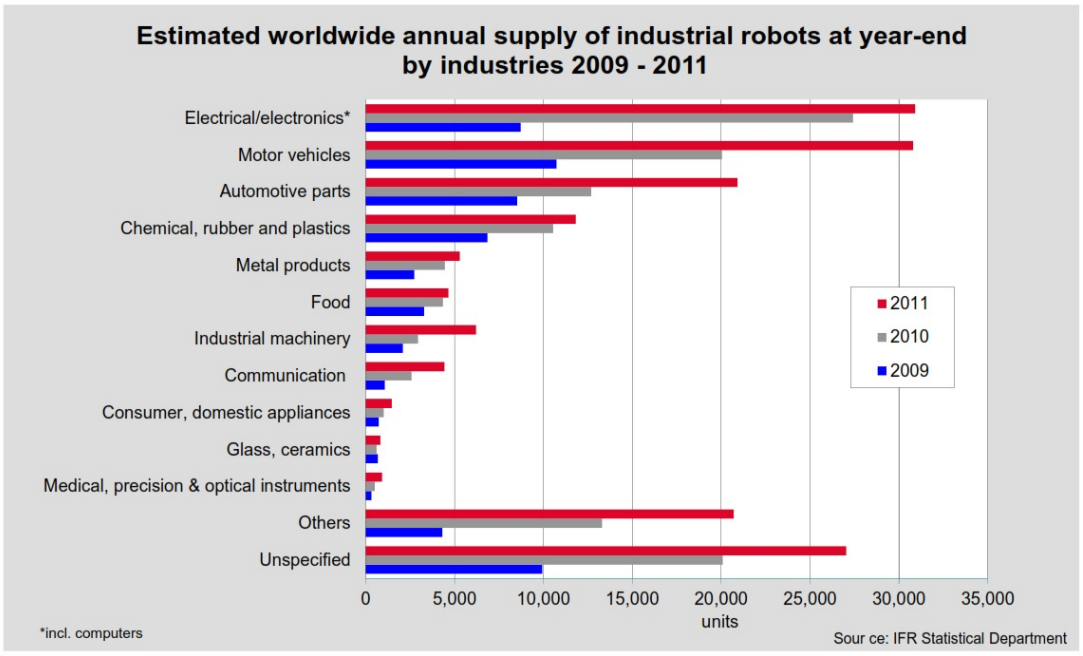


**Figure 1.1: Industries purchasing robots -** Estimated wordwide annual supply (taken from [1])

Manipulators fit perfectly to these environments: production resumes for months, sometimes even years — in the mean time, no reprogramming is needed, only casual maintenance; large factories are build from scratch, supported by fine design (with virtual models), with each step of the production stage carefully studied and identified — it is known *a priori* how many robots, which task they are appointed to, the workspace and its layout; there are experienced programmers that can develop machine code and there are expensive simulation software products to aid them. The bet on robots is certain and undisputed. Flexibility, in this scenario, means that robots can do the tedious and repetitive tasks humans would have to do otherwise; means calling in the programmer, weeks apart, to debug and correct the manipulation procedure or even change it completely to something else, instead of having to hire and teach an human operator for days. The 2012 annual report [1] from the International Federation of Robotics (IFR) shows the increasing use of industrial robots (Fig. 1.2, since the 2009 crisis and up until 2015(predicted)). Apart from the automotive industry, which has always been the major growth stimulator on process robotization, all other sectors are creating conditions to apply manipulators to their processes as they are seen as the prime tool for modernization, competitiveness and improvement of quality of work for human operators.

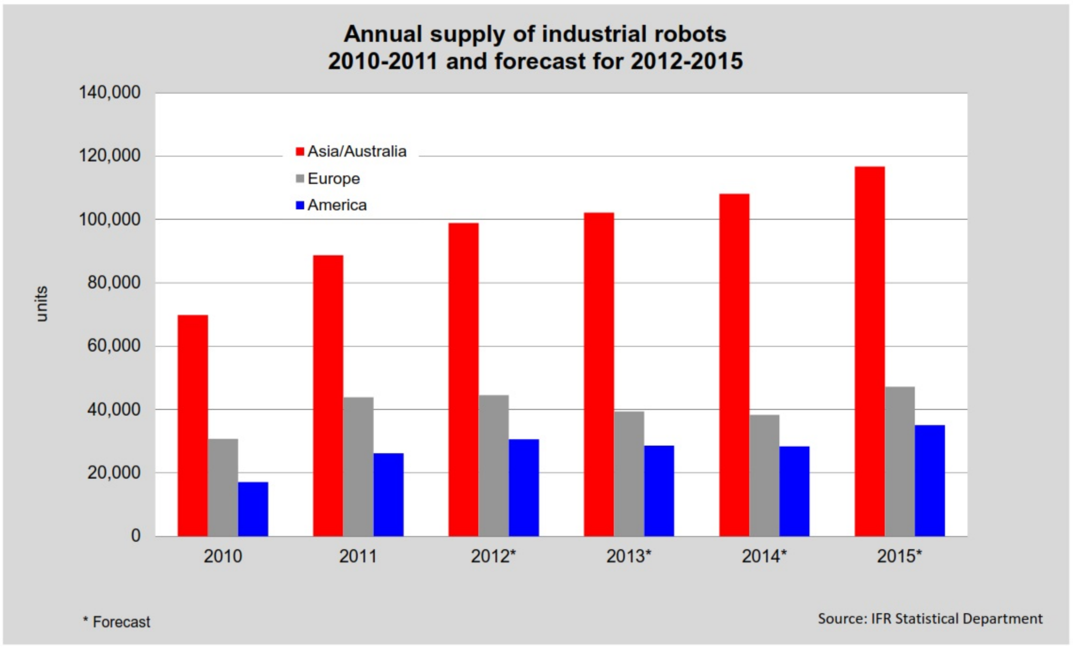


**Figure 1.2: Industrial Robots** - annual supply (taken from [1])

What's then on the other face? The small batch production, the most limited resources, companies with just a few tens of employees, the small and medium enterprises (SMEs). Are robots of as much importance as on large companies? Do they bring the same benefits? It is believed that SMEs are the economy driving force [2]; it is particularly true on rather small countries like Portugal. For years, SMEs relied on skilled labour and scarcely automated processes, but the market has changed; technology prices dropped low, companies started to modernize, the competition is at global level and above all, demand is different. Together, these factors led to the current mass customization trend. Production has to face smaller series with greater customization; consumers search the market for personalized goods, for tailored solutions. And few scenarios can be more dynamic than those based on customers tastes. Companies strive to answer this call and compete on the global market but this dynamic is costly: the production lines must constantly adapt to new processes; skilled operators take years to train. As such, the main work force is very limited. Moreover, mass customization does not work if the product price does not remain low. To achieve the required flexibility and low unit cost SMEs need no modernize; support their processes on automated and computer-aided systems; increase productivity with the same human labour. Robots do this at large companies. For SMEs, on the other hand, robots do not pose such an obvious solution. The long-term flexibility of manipulators do not fit the short-term demands. Reprogramming a robot takes too much time: a simple change in a pre-existing program takes hours and a full reconfiguration may require days. Furthermore, SMEs do not have the financial power to hire a full time skilled robot programmer. All of these represent extra cost for the enterprise: hiring of experts, automated machines, recurring configurations and the associated down-time. Before this scenario robots have a hard time penetrating and earning their place into small business.

### 1.1.1 The quest for human-robot interfaces

Overcoming the implementation limitations of robots in SMEs has been one of the most researched themes in the robotics field. Improving the interfaces with these machines achieves several goals at once. It starts by dispensing a specific programmer; then, under skilled operators with no technology know-how can operate complex machines — no new hirings and no training; faster reconfiguration times indicates increased productivity. So a quest has begun to develop the most advanced robot interfaces. They range from instructing the robot using voice commands or gestures, to new ways of developing machine code using block diagrams and pseudo instructions. The goal is to create intuitive human robot interfaces (HRI); means of controlling the robot using natural language and common daily use devices; or even allowing the robot and the human to operate in the same space with complete safety.

The research on this field has diverged into many streams. In the late years it has expanded over other robotic areas such as medical, domestic services, security... the appearance of humanoids led to a pursuit of ways to make the machine, made at our image, to behave like ourselves. But the research at industrial level as continued because there isn't a solution that can fit the so many problems industry faces. Also, much of the state of the art contributions use technologies that are not prepared for industrial use, others are simply cost prohibitive. Chapter 2 presents an in depth view of the current developments in HRI and easy robot programming. Yet it must be noted that as the current market demands more of the companies, the research on this field must also be able to answer the call and support small and medium business, with new methods that facilitate the integration of robots in highly flexible processes.

## 1.2 Motivation

On the very same line of thought presented above, this dissertation was motivated by the lack of industrial grade solutions that enable intuitive and easy robot programming.

The changes on the global market, the problem of small series production, customization, improving robot flexibility, etc... for years, contributions on this field have been supported by the same reasons, targeting the same goals. So why isn't there a solution already? Processes change everyday, companies attack new markets, develop new products, new solutions. It is impossible to cover every area with a single system, a unique programming or interface method. Moreover, technology is getting cheaper everyday; automation of production lines becomes available at reduced costs. Despite existing numerous contributions there are always new sensors to explore, more recent machines to experiment.

Another motivational factor comes exactly with the industrial applicability of the existing or currently researched techniques. Flexible robotic manipulators can aid the most flexible business, SMEs. Yet this is where financial resources are the lesser. Proposed solutions must be affordable both in matters of money and time. The three major limitations that can be found on the proposed solutions found in literature (discussed on the next chapter) are:

- The actual financial cost. Most of the times, researched technology is still fresh, i.e, the market price is too high. Simply put, the solutions become unaffordable.

Companies do need to improve production and quality but they also need to keep the cost low otherwise competitiveness is lost from the start. Robotization, i.e, acquiring the robot and re-designing production lines to fit it in is already a strong investment. The support technology, HRIs and intuitive programming software, cannot pose an equivalent financial burden.

- The dimension of the hardware apparatus and/or its inadequacy to industrial use. A grand part of the research work (on industrial manipulators) lacks actual industrial validation. Sometimes the hardware apparatus is inappropriate, fragile or it simply cannot deliver in an industrial environment — there are examples such as using retro-projectors to teach paths, which are equipments clearly not ready for dust, intensive work... the aggressiveness of the industrial environment; using IMUs (inertial measurement units) when, most of the times, the production lines or working tools are metallic and the sensor is most likely to fail; using special gloves requiring that the operator wears an object he is not used to work with and affects his performance; or even using vision systems that require major transformations in the workspace to control lighting thus being costly.
- Time. Many solutions base themselves on complex and computational intensive algorithms. In addiction to the required hardware (high-end PCs or PC grids) which are expensive alone, there is also the time it takes to interact with the robot or to generate paths, making it unaffordable. Productivity cannot cope with elevated downtime. Whether it is a gesture/voice recognition setup or a path demonstration system, a quick delivery is mandatory for increased productivity. A 90-plus percent identification rate on a recognition-based interface still means that after a thousand work pieces or tasks almost a hundred failed; a path demonstration system that takes several minutes to process data from a minute-length task do not improve reconfiguration times – if the path has to be generated twice or a third time to perfect output, and it happens so for every different object, production efficiency drops instead of being boosted.

With each of the pointed facts, that alone make enough reasons to invest on the research of new intuitive programming methods, is also a last argument. Only a fraction of current contributions focus on acquiring knowledge from the operator. Simulators, CAD based programming, gesture recognition, software interfaces, all fit to a particular set of problems and they contribute by providing means of interacting with the machine. A more broader solution, one that would satisfy a larger set of applications, is one that can learn from the operator, him who knows best the process and procedure. Rather than giving a mean to communicate with the machine, give it the know-how. Factory operators hold all the expertise required to perfectly perform the task at hands; companies survive due to them, they are the major asset. So, instead of providing the operator with a way to speak to the robot intuitively, it is amazingly powerful to have the ability to transfer skill from man to machine.

### 1.2.1 A case study

This thesis has been particularly fuelled by a real industrial case scenario.

Flupol [3] is a Portuguese SME operating in the area of industrial coatings. Their applications range from house-ware utilities, automotive parts, baking industry, and up to any general industrial application where advanced coating is required (for surface adhesion, dry lubrication or corrosion prevention).

Its main asset lies on the application (spraying) of complex and delicate products which only a few extremely experienced operators are qualified to do. Hence, the major surplus comes from the operators' know how. This is quite valuable since the company estimates that fully training new operators takes as much as 8 years. Right now, only a couple of employees are considered experts.

Flupol also struggles with the reduced dimension of Portugal and the Portuguese market. Over 75% of the production is destined to external markets. As such, Flupol finds itself as a global competitor, dealing against cheaper labour or more technologically advanced adversaries. Even so, Flupol's exclusive expertise on the product application and on the control of several process parameters give them the edge. These variables (know-how and process parameters), however, are non patentable which makes property protection very restricted. To overcome these constraints Flupol relies on an highly dynamic and versatile business model: costumers are treated individually, solutions are customized, processes are designed to fit exclusive orders — it is a one to one service. For that, the company relies on non dedicated production lines/cells, each able to be adapted to the must varied demands. To keep costs low, each application is designed to make best use of the available assets; to improve productivity however, machines and cells must be constantly updated to make use of new automation devices. The next step: robotization.

This scenario is the most perfect match to the one presented in the previous sections: high skilled labour, highly flexible processes, constant reconfigurations... the need to increase competitiveness by boosting productivity while maintaining quality... the seek of increasingly automated solutions which, ultimately, cries for the industrial robot.

And if the urge for robots is true, so are the requirements and limitations of a flexible small company: hiring programmers is too expensive nor it delivers the required fast reconfigurations; typical operators stand as painters, not informatics – programming the robot must be the most intuitive.

Finally, the greatest question on this case: is it possible to transfer the skill of the painter over to the robot and avoid 8 years training of an operator? Can a 25-year-expertise badge be instantaneously transferred to the machine?

## 1.3 Proposed Solution/Aims

Against the panorama analysed above, to aid penetration of robots into the smaller business and with a real test case in hands, this thesis sets focus on developing an industrial grade human-robot skill transfer framework.

Starting on the target machine, the robot, the proposed system interfaces an off-the-shelf industrial manipulator: these are mature products whose cost, maintenance and technical support cannot be matched with home-made solutions. Standard industrial manipulators include years of refinement in motion control and security aspects. They are the most solid choice for the end product.

The human-robot skill transfer ability shall be achieved by means of capturing the human movement and replaying it with the robot. Knowing that the deepest know-how lies on the details of operation and tasks, capturing the operators' moves in the actual scenario is the crucial step. For this, the proposed solution integrates an artificial vision architecture that records human hand movements in a contactless manner. The fundamentals of stereoscopy (3D from multi camera arrangements) are used to track a luminous marker that is attached to the operator work tool; the marker is designed to endure industrial use, being particularly robust to lighting conditions — no artificial or conditioned lighting is needed. The stereo system, together with the marker, are designed to have minimum impact on the process; operators execute the usual tasks with no interference from the hardware apparatus; this allows to completely capture the essence of the human know-how.

Put together, all elements are industrial grade and by industrial grade is it meant that the solution shall attend to the major industrial requirements: sensors, actuators and auxiliary hardware paraphernalia must be able to endure the environment; all the apparatus must be the less intrusive possible, i.e, not requiring major changes to the processes; also, the operator must be able to act naturally without major cares towards the sensing setup; the framework must deliver in real-time meaning that there can be no overhead in the algorithms – as soon as the operation is demonstrated by the human, the robot must be ready to replay it; the whole system must be low cost, using standard industrial equipment.

On the end, the proposed solution integrates a set of industrial cameras and a cheap luminous marker. Together these devices grant a robust capture of operator's moves. Then, a set of computationally fast algorithms can determine the marker position and orientation. In a further step, the movement is analysed and segmented in order to be "written" in robot language. Finally, the robot is set to replay the same exact movement as made by the operator. From the users point of view, there is a complete abstraction of the robot programming language. The machine is programmed with no interaction with teach-pendants, code or any other sophisticated interfaces other than a "start/stop learning" button.

To test for the industrial capability of the framework, Flupol is used as the test bed. The skill transfer concept is applied to the spray coating application, where an experienced painter demonstrates the coating technique and the robot mimics him. The spraying over complex forms is replenished with technique and intricate wrist moves which serves the purpose of intensively validate the proposed framework.

### 1.3.1 A Previous Approach

One of the first attempts to improve Flupol's productivity through robotization was based on CAD programming and automatic object recognition. This system is thoroughly described in Marcos Ferreira et al. [4]. In short, the research focus on generating robot programs for the painting tasks through the use of a CAD interface. Along with it, an automatic recognition system is used based on camera–laser triangulation. The parts travel on a conveyor and are scanned by the laser; then, a machine learning algorithm identifies the part and uploads the correct painting program into the robot controller. With this system it was expected that both the program generation and the reconfiguration for different pieces would occur in a simplified manner without using the robot teach pendant. Nonetheless, it was concluded that the generation of accurate painting programs for complex shapes using CAD would be too difficult. So, the pursuit for means of collecting the know-how of the painter begun, and with that a simpler way to generate robot programs for any shape.

## 1.4 Thesis Outline

- Chapter 1 serves as the introductory chapter. It presents the panorama of using robotic manipulators at the industry level, particularly on SMEs. Discussion follows with the challenges of interfacing these machines and to allow factory operators easy interaction with them. The problem addressed by this thesis is exposed together with the proposed solution and major goals. The industrial demonstrator is described and framed with the research aims.

- Chapter 2 presents a review on the state of the art on programming by demonstration, human motion capture and intuitive human-robot interfaces. A relevant set of contributions has been selected to provide insight on current research work along this stream. In addiction, they are also meant to support and justify the line of research of this thesis.

- Chapter 3 provides the most basic tools for the understanding of the remaining of the document. It defines the notation used in equations and schemes, and presents some fundamental notions on cameras, stereoscopy, orientation of rigid bodies, etc. Also, some trivial algebraic relations are defined so that they can be skipped later and make the development sections more readable.

- Chapter 4 is the grand development chapter. It presents the design, implementation and methodology that make up the proposed solution to a novel human robot skill transfer framework. The chapter itself follows the structure of the motion imitation framework: starts with the sensing component, the module that deals with capturing human moves, down to replaying those exact movements with an industrial robot.

- Chapter 5 describes the implementation of the proposed solution in the industrial demonstrator. It comprises the installation and test routines that validate the developed framework. System precision is accessed and discussed.

- Chapter 6 provides the conclusions taken from this research work: a final word on the achievements, the novelty of the system and where it excels. Also, some lines of thought are addressed to future research work: how the framework can be improved, adapted or used for other ends.

# Chapter 2

# Related Work

This chapter presents the state of the art methods and achievements on robot programming-by-demonstration.

The universe of human-robot interaction is vast and it is hard to cover all the current streams of research in the area. Brenna Argall et al. [5] and Geoffrey Biggs et al.[6] offer a survey on robotics learning/programming from demonstration. According to the author of [5], the work on this thesis is best categorized as an imitation learning process. On this line, this literature review sets focus primarily on motion and tasks demonstration, which provide the necessary data for a robotic imitation. Also, the main goal is to achieve a system that allows transferring skill from a factory operator over to an industrial manipulator and, for that reason, this review is restricted to the interface with industrial robots.

Within this shorter scope, the reader starts by finding references on the actual methods for robot programming, mainly commercial-off-the-shelf products as simulation software and the teach pendant (which is the default programming tool provided by robot manufacturers).

From here, discussion follows through contributions found on programming by motion demonstration. In-between, a review on CAD based programming takes place since this is a major stream of research on the field of intuitive robot interfaces. On programming by demonstration there are also different approaches: gesture recognition, lead through programming and actual motion capture for robot playback. Each of those is discussed and, for the sake of completeness, a few contributions on humanoid/mobile robot interfaces are also analysed. Although these are not the type of platforms that this research aims to interface, some knowledge can be retrieved on limited sections of such contributions: methods and technologies for motion capture (which are commonly used in humanoid research and to build virtual reality models of humans), and methods for robot path planning, trajectory smoothing and collision avoidance (which are also used in mobile robotics).

## 2.1 Programming Industrial Robots: the Teach Pendant and Simulators

Industrial manipulators have their vary own programming language. Unfortunately, each brand has developed their specific version making it hard to develop an universal method or tool to program these machines. Another drawback of this scenario stands on the costumer side, who is forced to stick to the same manufacturer in order to prevent their operators to have to learn how to work with/program different machines. The first and most used method to teach industrial robots are the teach pendants — Fig. 2.1. The reasons are obvious: it is the default tool shipped with every robot, it comes with no extra costs, manufacturers support is guaranteed, it is well established as it has been around since the first manipulators. Furthermore, these tools have access to all the robot functionalities, are designed to endure industrial environment and provide an easy-to-learn interface for the final user.

Ease of learn, however, does not translate into ease of use or, more importantly, into efficiency. The method of programming through the teach pendant is tedious, time consuming and far from intuitive, mostly to the expected end user: an inexperienced factory operator. Through the pendant, the user moves the robot to the desired position, records the point, defines the type of movement, velocity and several other parameters; all those steps for a single point, repeated to every point that makes the desired path. In the end, the resulting program can hardly be used for anything else other than the task it was first meant to; adaptation or upgrades cost as much as redoing from scratch. In the SME scenario, specially with small series production, this is not a practicable solution. Therefore, the first approach on improving human-robot interaction has been on the enhancement of these existing interfaces. At the teach pendant level, KUKA has integrated a 6-D mouse, developed by the German Aerospace Center for space robotics projects [7]. Renzo Calcagno et al. [8] present a wireless

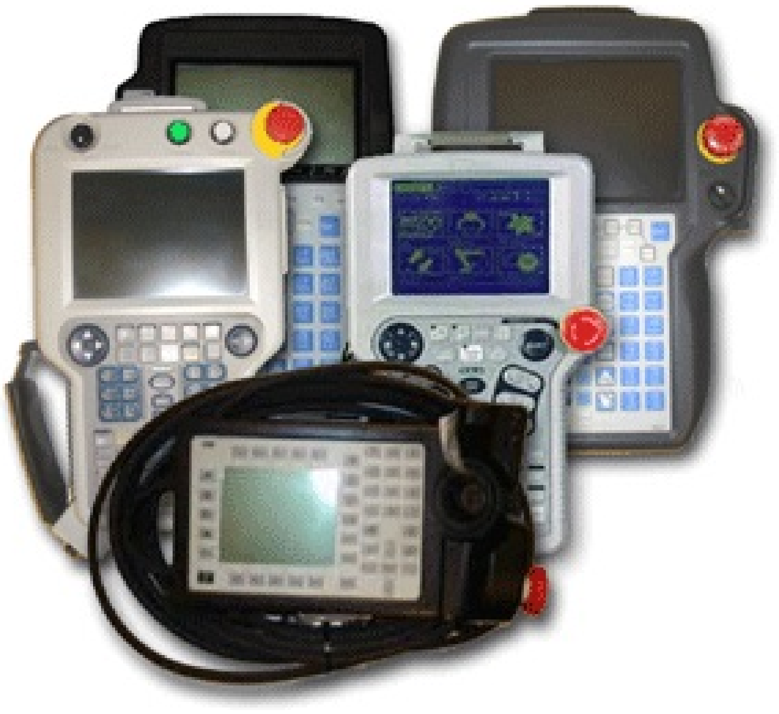


**Figure 2.1: Teach Pendants** - Each manufacturer has its own layout for the teach pendant

pendant aiming to improve flexibility and usefulness in tight shop floors but also to provide means for distributed supervision and multi-unit control. Even though these steps enrich indeed user interface, the programming method is still the same as are the proprietary programming languages. In a fairly recent study, as of 2011 (Lei Wang et al. [9]), it is obvious that the research community still seeks to overcome the limitations of such reality; in this case, the authors use case-based-reasoning to segment parts of the machine code in order to reuse them in similar applications avoiding to rewrite the code or to build a full point-by-point path all over again. Similar approaches dated old can also be found: replacing the code editor and interface by an icon-based flowchart [10] and the development a set of modular robot programs that adapt to different shapes provided the user is able to extract geometric data of the work-pieces [11] — then, a program variant is automatically created. All of these concepts cannot relinquish a skilled programmer with experience in robot specific language nor provide a way for a common factory operator to intuitively interact with the manipulator.

Another common reality of robot programming nowadays are the simulators. Some major dealers of industrial manipulators have their own simulation software, such is the case of RoboSim from Motoman and RobotStudio from ABB. Simulators provide the ability to test machine programs offline, saving testing time along with other key advantages: visual feedback, improved control over robot motion, security analysis, collision avoidance, tool/robot dimensioning, cell layout definition, etc. To cite a few, there are some projects that take full advantage of these virtual reality tools and try to further improve their usefulness and flexibility as reported by Xiongzi Li et al. [12] (the authors import CAD models of the pieces to generate and validate a painting path) and by Liwei Qi et al. [13](the paper analyses the reasons why robot simulations and offline programming are far from intuitive). One of the greatest drawbacks in simulations is the need of an accurate representation of the real world scene: representing the robot, the workspace, the work-pieces and every layout details with low precision grounds the effectiveness and interest of offline simulation. Also, the volumetric inaccuracy of the robot may compromise the virtual→real transformation: even if the virtual environment has accurate models, the robot may fail to respect the coordinates and still fail to deliver. Xiongzi Li et al. [14] present a solution for calibration between virtual and real scene increasing the rate of success when mapping simulated programs into the physical robot — [12],[13] and [14] all use ABB's RobotStudio. The work of

Haixiang Yu et al. [15] is conducted over Motoman's MotoSim, and advocates the use of simulation based programming in a palletizing application for a safe and efficient output.

Each and every of the discussed researches assume the default tools for robot programming and try to build more productive and/or intuitive layers on top of it though they do not close the gap between man and machine. Before us remains the scenario where the highly skilled (and expensive) programmer has to code for manipulators. Although the teach pendant and the simulators are indeed the *de facto* standard of industrial robot programming, while the former is not easy to use nor time efficient, the latter are dizzyingly expensive. Again, for SMEs, that is not an affordable solution. Besides, there's still no away to take the operator expertise and send it to a robot.

## 2.2 Taking advantage of CAD

On a similar line to simulation and modelling tools comes the offline programming using CAD. This stream alone has received notorious attention by researchers and can be considered one of the major fields of human-robot interaction.

Since companies are increasingly resorting to CAD software to design their products, this is an existing valuable asset that was waiting to be explored. Through the 3D data on CAD files it is possible to generate robot programs that can uniformly cover the object surface (useful for painting/coating applications), move along the contour (grinding and polishing tasks) or to plan ahead complex trajectories for collision free operations. Contributions to this area of expertise started long ago; Grier Lin *et al.*, back in 1995, have demonstrated the capabilities of CAD for object recognition and adaptive pick up applications with manipulators [16]: the authors used cameras to capture images from the workbench which where compared to CAD drawings for proper identification; robot programs where then generated to grab the objects in the messy workbench. On an extension of that application, the same authors used the CAD data, neural networks and torque sensors to further improve the grasping capabilities of the robot and the compensate position uncertainty [17].

In the fields of industrial painting and coating, CAD has also served numerous researches; Chen Heping is an author who provided a considerable amount of contributions in this area: [18] and [19] are examples where the author seeks methods for automatic robot path generation in order to evenly paint a work object based on its CAD drawings. The later contribution shows the application of the developed algorithms in an actual industrial situation of automotive painting. The same authors try to expand the knowledge of this CAD based painting to applications that require any kind of surface manufacturing. Although the basis are the same, the scope is widened [20] and a CAD based path planning framework is proposed. Years later, as of 2008, still the same authors release novel research on CAD based coating of composite materials [21], together with a review of industrial robot path planning in spray painting applications, [22]. This shows the usefulness of automatic path planning through CAD but also indicates that a generic approach is hard to accomplish. Furthermore it can be seen that industrial painting applications are still requiring major research work; the variability of shapes, applicable products and painting techniques make it harder to develop a single framework.

The industrial demonstrator for testing and validation of the research work presented on this thesis is itself a case of industrial spray painting. It is believed that the true know how lies on painters, experts with years of contact with the application; the CAD solutions do not take advantage of that expertise nor allow operators to transfer their skill to the robot.

The CAD employment obviously do not limit to painting application. To make reference to a few more applications, T. Pulkkinen et al. [23] use 2D CAD to automatically process metal profiles. M. Soron et al. [24] report on the field of welding — again, CAD models are used to generate continuous tool paths. Norberto Pires et al. [25] suggest an interface for easy mapping from the CAD to the robot program, also directed over welding processes. Another different application, polishing, comes from Fusaomi Nagata et al. [26] — other than for path generation alone, the CAD model is used to integrate data from the force sensor, and feed it to a force controller to prevent an over/under abrasive operation.

Since CAD data may not be as much accurate as needed for a given task, or the robot may not have the required accuracy, researches are conducted ahead of the simple path planning: P. Neto et al., [27] and [28], seek to compensate for poorly calibrated scenarios or for inaccurate/uncertain CAD models.

As mentioned before, CAD based robot programming have been seeing quite a respectful amount of research and the number of contributions is vast. As such, it is worthy of a mention in this state of the art overview. Nevertheless, this method of programming is more of a sophisticated interface than a true demonstration method. The user do not teach nor is able to pass on its expertise, he rather points out positions to where the robot should move. It is indeed an improvement over the standard robot programming interfaces but still lacks the ability of bringing together the robot and the actual factory operator who usually executes the tasks.

The following section provides an insight on current and previous projects that show how operator-robot interaction can or may evolve in the years to come. From the vast set of contributions on the PbD (Programming by Demonstration) field it is from programming by motion demonstration that most profit is taken from this state of the art analysis. As commented before in the introductory chapter, this thesis searches for a novel way to learn (even if only by imitation) from the human expertise in industrial processes: to find a way to capture know-how while operators do their tasks, instead of forcing them to learn how to use either software or hardware.

## 2.3 Programming by motion demonstration

The interaction with robots using the hand motion or gestures is arguably the most explored. Gestures and movements come as the most natural means for humans to express themselves. The programming by gesture recognition is, however, a mean of intuitive interaction rather than motion demonstration; for the sake of completeness of this review, some of the work on motion recognition is discussed: Stefan Waldherr et al. [29] present a framework based on video tracking that allows a mobile robot to follow an human; the same video data is used with neural networks and a template matching algorithm so that, if a given predefined gesture is captured, the manipulator on top of the mobile base executes pre-programmed tasks. The same approach is used by Matthias Strobel et al. [30], this time with hidden markov models and integrating context knowledge to better understand human intentions. Instead of a pure video inspection of human moves, Piyush Kumar et al. [31] developed a glove which allows a more precise tracking and identification of gestures.

The number of contributions on this area continues endlessly, most not directly related to industrial manipulators nor even robotics. Nevertheless, all these contributions share a common line: the human is given a way to interact with the robot but no means to program it or change its behaviour as needed. The recognition enables the activation of a corresponding robot program which must be programmed beforehand. There is no actual demonstration from the operator. In the same line of though comes the programming by voice commands, as exemplified by Norberto Pires [32]: the voice patterns are recognized and trigger tasks executions (which must also be pre programmed). This approach may play an important role in conjunction with solutions for actual programming by demonstration, liberating the user from any computer interaction. If the goal is to ever dismiss an experienced robot programmer at the factory, the previous solutions prove insufficient; they do hold a powerful way to interact with machines yet not teach them.

With focus on finding ways of demonstrating tasks, preferably with industrial capabilities, the following set of contributions have been analysed deeply. The techniques for capturing human moves or interfacing the industrial robot are close to what is expected from this research work.

R. Dillman has several contributions on this field. Using stereoscopic vision and a special glove, [33], the author proposes a framework for object handling where motion is captured and segmented as tasks; each task is classified according to its goal and is replayed using a linear-move, circular-move or free-move criteria. The proposed motion sensor, the glove (Fig. 2.2), besides the lack of industrial robustness, still presented a hard tracking problem. Most of the human move lies on the wrist and the glove does not directly measures it; the finger movement and inter-finger angles were used to perform a better estimate of the wrist movement. Even so, the neural network based processing had a low success rate, at least for industrial grade. To improve this system, the author upgrades the glove now to include tactile sensors also, [34]. This allows an easier task segmentation but does not improve the tracking ability. The same data of force on the fingertips is also used to upgrade the PbD system, and expand it to areas other than pick and place. The tactile sensors on the glove were used with an SVM (support vector machine) algorithm to identify grasp details and understand complex tasks such as screw bolting, [35]. The introduction of the new sensors, however, brought the problem of sensor fusion in order to merge camera and glove. This problem is addressed in [36]. Further work of this author have been carried out on this grasp-understanding framework. Recently the research has drifted towards humanoid robots but a hole has been left behind: actually exporting the results toward industrial applications; precision, repeatability and high success rates are the keys for successful scientific knowledge transfer to the industry market and little care is shown in past research work to these topics.

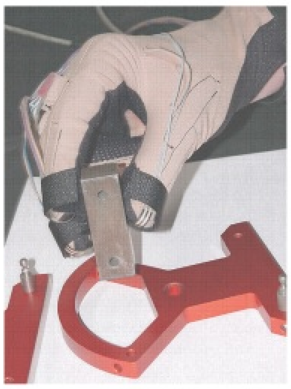


**Figure 2.2: Glove** - A glove with tactile sensors proposed by Dillman, R *et al* [33]

Skoglund et al. [37] also focus on pick and place tasks alone, using fuzzy clustering to identify the stages of a human demonstration. The robot playback is set to achieve the same goals but no care is taken that the trajectory is precisely the same. This serves well a pick-and-place task but is due to fail in a welding or painting operation. The work presented by Y. Maeda et al. [38] targets an industrial robot. It has a teaching-by-showing stage and a robot playback stage. Capturing the human motion is done with video images which are then fed to a neural network. After training, the robot achieves the correct manipulation task. This contribution, despite targeting an industrial manipulator, does not have a pure skill transfer core. The operator moves the robot in the teaching phase instead of doing it himself. Also, multiple teaching is needed until the robot performs acceptably. In the same line of the lead through programming, Germano Veiga et al. [39] report a system where the user also moves the robot end-effector, and at the same time a torque sensor records the force pattern along the path; this improves playback since an extra loop for force control is added allowing online corrections to the movement.

Another different approach to PbD is reported by M. Stoica et al. [40]: the user shows an action and the robot is set to repeat it in a loose fashion, i.e, the robot performs an action that is similar yet not precisely the same. Based on the robot behaviour it receives marks that are used in a reinforcement learning algorithm, until the desired motion output is achieved. Even though the authors do stick to industrial manipulators and point out the interest of interfacing with these machines, the developed system is hardly accepted as an industrial solution mostly due to the learning stage: the algorithm takes time and many failed repetitions until convergence on the desired behaviour.

Bjorn Hein has a pair of complementary contributions that deal in fact with movement demonstration. The author presents a set of new input devices [41], based on artificial vision and infrared markers; these are attached to different tools to cover distinct applications — Fig. 2.3. The second part of the system [42] integrates the tracking component in a more complex framework which deals with tele-operation, simulation and path planning. The hardware components are thoroughly described as well as the calibration stages of the system. Despite that, few tests are presented and there is no actual discussion on the precision and the industrial applicability of the apparatus. Furthermore, the major drawback lies on the robot interface: the authors use a Kuka motor controller interface which allows controlling each joint separately; the movement playback is achieved by doing inverse kinematics on the robot in order to replay the tracked trajectory. This type of approach is far from optimal since the high level controllers of the robot are disregarded — failing to provide a consistent set of joint angles and the robot may break apart; the robot model must be very accurate so that no precision is lost; finally, the standard off-the-shelf robotic manipulators do not provide such low level interface (for the same reasons discussed. The consequences of a software bug could be disastrous and wreak havoc on the surroundings and to the robot itself).

The contributions discussed so far focus on providing a PbD framework; yet, some profit can be found in exploring other fields of research that, for example, may not provide interface with robotic manipulators but deliver an interesting motion capture system or vice-versa. Field et al. [43] contributed with a fairly recent survey (2011) on motion capture sensors for robotic applications. From this study, one can keep track of the different technologies that enable motion tracking. Both Elgammal et al. [44] and Shon et al. [45] make use of passive markers (infrared), for outdoor body

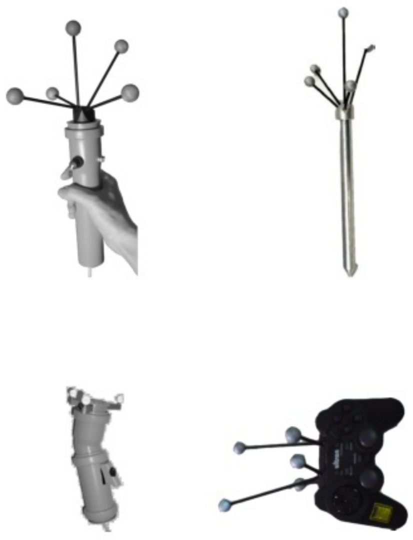


**Figure 2.3: Input devices** - New input devices proposed by Hein, B. *et al* [41]

tracking and indoor humanoid imitation, respectively. Naksuk et al. [46] and Leonid Sigal et al. [47] also use infrared markers but active ones. The former is an application directed towards humanoid control and the latter proposes an algorithm to evaluate articulated human motion. Both passive and active infrared markers have increased post processing times with the increasing number of used markers; differentiation of each light is usually done with a combination of inter-marker distance which becomes harder to achieve with a larger number of markers. Markerless options also exist: Azad et al. [48, 49] propose a stereo based human body tracking with no markers required — the precision of these systems fall short to the marker-based alternatives, nonetheless. Inertial [50] and magnetic [51] solutions (non vision based) are explored extensively. While magnetic sensors fail abruptly near metallic structures and provide noise full measures [52], inertial sensors do require double integration and, as such, position estimation is poor. Moreover, data fusion is often required as an extra step in post or online processing [53].

On the other end of the PbD framework lies the robot interface. This encompasses trajectory clustering, filtering, segmentation and planning. Some contributions excel in this area: Chien-Chou Lin [54] proposes a memetic algorithm to deal with path planning with collision avoidance taking into account the possible/allowable configurations of an industrial manipulator. Aleotti et al. [55] describe a spline smoothing technique to filter noise and human inconsistency on a set of repeated tasks. Joon-Young Kim et al. [56] present a minimum time trajectory planning algorithm also for industrial robots; a similar approach is followed by Wenguang Li et al. [57], using polynomial fitting.

From this set of discussed contributions it becomes clearer the areas where this research work can provide novelty. First, on the motion tracking component, active visible markers are seldom found; the typical problems with lighting conditions may have driven away researches but with the recently developed *sincrovision* concept there is space for a new kind of marker. This can improve detection and measurements, allowing a greater number of marker while maintaining low processing times. Another aspect that can be improved is the industrial capability: low cost, high precision and low latency times are fundamental for competitive production lines; these aspects seem disregarded in many contributions.

# Chapter 3

# Background and Notation

This chapter section summarizes some basic notions that help understanding the remainder of the document. It includes artificial vision concepts (camera model, calibration, reference frames, lenses,...), linear algebra relations for dealing with coordinate frames (rotations, translation, homogeneous coordinates,...), quaternion algebra and programming/interfacing industrial manipulators.

## 3.1 Matrix Notation

The most essential basics to understand every equation and procedure in this thesis are related to matrix operations and notation. Summarizing:

1. **matrices** are represented by bold upper-case letters, for instance, matrix $\mathbf{A}$;

2. the elements of $\mathbf{A}$ come in italic, lower-case letters — $a_{ij}$;

3. the indexes $i, j$ refer to matrix row $i$, column $j$;

4. the definition of a matrix appear in square brackets:

$$\mathbf{A} = \left[ \begin{array}{cc} a_{11} & a_{12} \\ a_{21} & a_{22} \end{array} \right] \tag{3.1}$$

5. vectors are Nx1 matrices, represented by bold lower-case letters, for instance $\mathbf{v} = [v_1, v_2, v_3]^T$;

## 3.2 Coordinate Systems, frames and transformations

In the remainder of this document, wherever is made reference to coordinate systems, without further specification, cartesian coordinate system is meant. Camera models and calibration, along with stereoscopic principles, make use of homogeneous coordinates (more on it ahead); this fact is highlighted in such cases.

Throughout the proposed motion imitation framework there are numerous frames (distinct cartesian coordinate systems) to be considered: the robot base frame, the robot's tip frame, the marker frame, the cameras frames, etc.

When a point in 3-D space is projected into a 2D image pixel, or when a point in robot base coordinates is mapped to the end-effector frame, a transformation of frames occur.

In Fig. 3.1, two frames are shown. The same point $P$ is represented in both frames. Concerning notation:

1. a frame is denoted by upper-case script capitals, e.g, $\mathcal{A}$ and $\mathcal{B}$;
2. a point is denoted by italic upper-case letters. E.g, $P$;
3. point $P$ on frame $\mathcal{A}$ is represented by ${}^{A}P$;
4. when a set of points is considered, the subscripted index in $P_i$ refers to point $i$;
5. ${}^{A}P_i$ has coordinates ${}^{A}P_i = {}^{A}(x_i, y_i, z_i)$;
6. alternatively, $P$ can be converted to matrix notation thus using the vector convention mentioned above: $P \rightarrow \overrightarrow{OP} \rightarrow \mathbf{p}$, where $O$ is the frame origin;

The transformation of frame $\mathcal{A}$ into $\mathcal{B}$ happens in two moments: a rotation ${}^{B}\mathbf{R}_{A}$ and a translation ${}^{B}\mathbf{T}_{A}$.

Rotations are represented by square matrices which rotate points about the origin of the coordinate system by a certain amount (an angle) — in Fig.3.2 (a-left), a rotation of $\phi$ degrees around the $z$ axis is applied, followed by a rotation of $\theta$ around the new $y'$ axis. More details on rotations matrices are given further ahead, in the angles and orientation section.

Translations in $\mathbb{R}^N$ are $N \times 1$ matrices and represent the offset between the two frames origins, ${}^{A}O_A$ and ${}^{A}O_B$ — Fig. 3.2 (b-right).

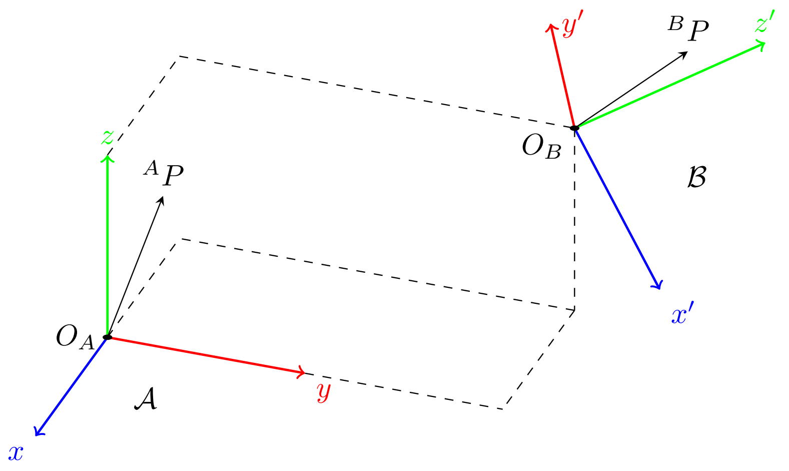


**Figure 3.1: Frames** - Notation for frames and points. Two frames are shown: $\mathcal{A}$ and $\mathcal{B}$. The later is obtained from $\mathcal{A}$ after a translation and a rotation

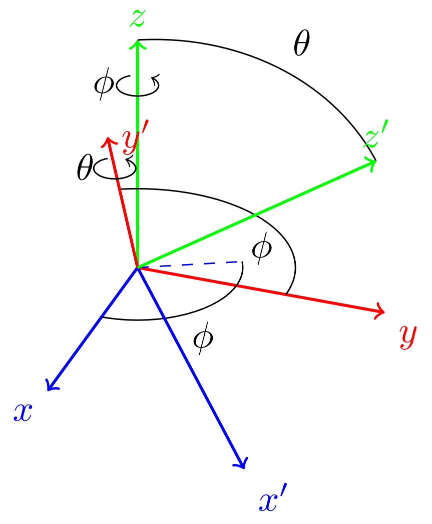
### 3.2.1 Homogeneous Coordinate System

At this point, homogeneous coordinates are introduced for two main reasons: first they simplify the algebra of frame transformation by allowing a more compact and computationally efficient representation of the pair rotation+translation and by making composition of successive transformations easier; secondly, homogeneous coordinates are also used to represent projective transformations which is the base for (3D) computer vision; as such, these notions will be used on the camera related subsystems.

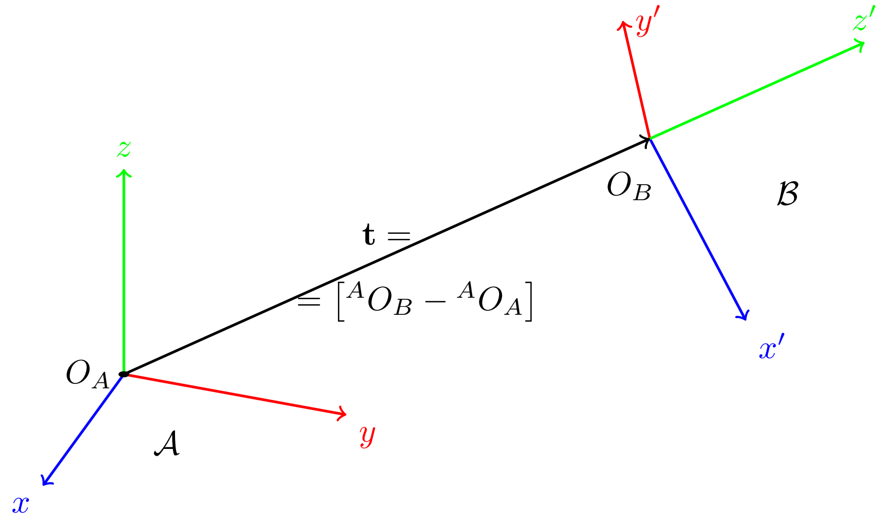
Concerning frame transformations (rotation plus translation), the homogeneous coordinates can be seen as an extra dimension, with value equal to 1. Simply put, a point in 3D space with coordinates $P = (x, y, z)$ has homogeneous coordinates $P = (x, y, z, 1)$. What are the implications? In the example of Fig. 3.2, if $^AP$ is represented in homogeneous coordinates, $^AP = (x, y, z, 1)$, then the whole transformation is written in a more compact manner:

$$^B\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \underbrace{\left[\begin{array}{c|c} ^B\mathbf{R}_A & ^B\mathbf{T}_A \\ 0_3 & 1 \end{array}\right]}_{^A\mathbf{H}_B} {}^A\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad , \quad \texttt{where } \mathbf{0}_3 = [0, 0, 0] \tag{3.2}$$

The homogeneous 4×4 transformation matrix $\mathbf{H}$ groups rotation and translation in



**(a)**



**(b)**

**Figure 3.2:** (a) **Frame Rotation**: The second frame $(Oxyx)'$ is obtained from the original by a rotation in the $z$-axis followed by another around $y'$. (b) **Frame Translation**: Vector $\mathbf{t}$ represents the offset from the two origins. Translation succeeds rotation. If translation comes first, then the rotation must be around the translated ${}^{B}O$.

a single form which is a more computationally efficient and clean representation. The composition of consecutive frame transformations is also made easier: it is a succession of multiplications by each $\mathbf{H}$ matrix — ${}^{D}\mathbf{H}_{A} = {}^{D}\mathbf{H}_{C}{}^{C}\mathbf{H}_{B}{}^{B}\mathbf{H}_{A}$.

The use of homogeneous coordinates in artificial vision problems is more than an added 1-value coordinate. In fact, homogeneous coordinates are useful to describe projective transformations and to have a non-singular definition of infinity. A point in 2D space still needs three coordinates: $P = (x, y) \rightarrow P_h = (x', y', w)$ where $x = x'/w$. From here it is easy to see that the mapping of cartesian coordinates to homogeneous ones is not unique: any value of $w$ can be set; the inverse mapping, on the other hand, is unique. When $w$ is zero the point is at infinity so homogeneous coordinates can in fact represent the infinity in the form of coordinates $(x, y, 0)$ instead of having to compute a zero division. In addiction, this added coordinate shows how it is impossible to go from a 2D frame into 3D, for instance, from an image frame into the world, without a complementary restriction — there are infinite possibilities (set $w$ any value); for the reverse transformation, a projection, it shows how there are many 3D points (along a line) that project into the same 2D point (whichever the value of $w$, $x$ and $y$ are the same when the normalization $w = 1$ occurs). Fig. 3.3 holds a geometric interpretation for the added dimension $w$.

### 3.2.2 Orientation of a Rigid Body: representations

There are several ways to represent the orientation (also called attitude) of a rigid body: rotation matrix (many times referred to as direct cosine matrix), euler angles, direction cosines, vector-angle representation and quaternions.

There are advantages in using some representations over the others. Rotation matrices are larger in size (9 elements) but their algebra is efficient and easier; nonetheless, interpolating between orientations is not possible. Euler angles are just 3 values but have discontinuities and the representation of a given attitude is not unique. Quaternion algebra is less common and more complicated but they offer means for interpolation that no other representation does.

The notation and some properties:

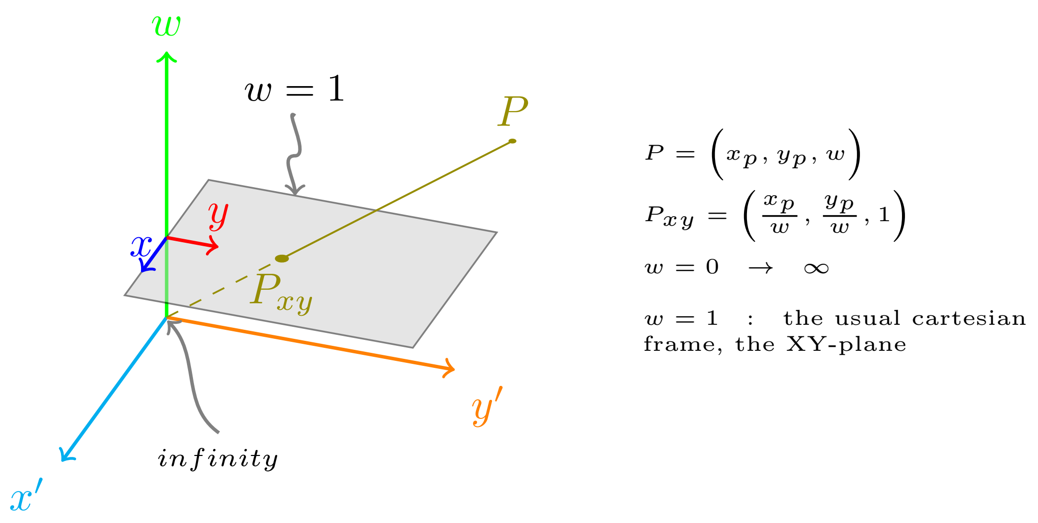

$w$
$w = 1$
$P$
$y$
$x$
$P_{xy}$
$y'$
$x'$
$infinity$
$P = \left(x_p, y_p, w\right)$
$P_{xy} = \left(\frac{x_p}{w}, \frac{y_p}{w}, 1\right)$
$w = 0 \quad \rightarrow \quad \infty$
$w = 1$ : the usual cartesian frame, the XY-plane

**Figure 3.3: Homogeneous Coordinates** - a geometric interpretation. A point $P_{xy}$ in cartesian $\mathbb{R}^2$ space ($w = 1$) can have infinite representations in homogeneous coordinates — all points along the line $P_{xy}$–$P$. The plane $w = 0$ represents infinity.

1. Rotation matrices use the standard matrix notation, upper-case bold letters, e.g, **R**.

2. The rotation around an axis $u$ by an angle $\theta$ is denoted $\mathbf{R}_{u,\theta}$.

3. If ${}^{B}\mathbf{R}_{A}$ rotates frame $\mathcal{A}$ to $\mathcal{B}$, then the inverse rotation ${}^{A}\mathbf{R}_{B}$ is defined by

$$ {}^{A}\mathbf{R}_{B} = \left({}^{B}\mathbf{R}_{A}\right)^{-1} \quad (\texttt{also},\ \mathbf{R}^{-1} = \mathbf{R}^{T}\ ) \tag{3.3}$$

4. Consecutive rotations (about the current axis) are obtained by pre-multiplication of the respective matrices:

$$ {}^{D}\mathbf{R}_{A} = {}^{D}\mathbf{R}_{C}{}^{C}\mathbf{R}_{B}{}^{B}\mathbf{R}_{A} \tag{3.4}$$

5. There are many conventions for use of the euler angles. In this thesis, euler angles were required to interface with a MOTOMAN manipulator and, as such, the convention followed is the one that best adapts to the robot programming.

   The euler angles define an orientation by the composition of three simple rotations, around the coordinate system axis. Assuming intrinsic rotations, i.e, around the moving axis, first there is a rotation around Z-axis by an amount $\phi$; then a rotation around the new Y-axis by $\theta$; finally, there is a rotation around the final X-axis by $\psi$. If fixed axis are considered (extrinsic rotations), then the order of axis rotation is reversed. MOTOMAN uses angles $(R_z, R_y, R_x)$ but this notation is easily confused with the matrices. Hence the adopted notation hereafter will be $(\phi, \theta, \psi)$; if a rotation matrix is associated with these angles and axis, then an orientation is described by

$$\mathbf{R} = \mathbf{R}_{z,\phi}\mathbf{R}_{y,\theta}\mathbf{R}_{x,\psi} \tag{3.5}$$

6. Lastly, there are quaternions. These are one of the most used tools to deal with orientations specially in the computer graphics field. Quaternions are an extension of complex numbers in 4D space: $\mathbf{q} = [q_r, q_i, q_j, q_k]^T$.

7. Unit quaternions represent orientation in 3D space; $q$ and $-q$ represent the same orientation while $q^*$, the conjugate, represent the inverse rotation;

8. unit quaternions live on the unit sphere of $\mathbb{R}^4$; interpolation between two orientations, which is not possible using rotation matrices, is simple in quaternion space — a ”linear” interpolation on the sphere surface (called *Slerp*, spherical linear interpolation).

The algorithms used in this thesis are mainly based on rotation matrices and quaternions. However, the interface with industrial manipulators is usually done using euler angles or quaternions. For this reason, and to simplify the presentation of the algorithms throughout the thesis, the methods for converting between each representation are here described. Later, the representations shall be used interchangeably without caring to present again the same basic calculations.

1. From euler angles to rotation matrix:

$$\begin{aligned}
\mathbf{R}_{Z,\phi} &= \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
\mathbf{R}_{Y,\theta} &= \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \\
\mathbf{R}_{X,\psi} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{bmatrix}
\end{aligned} \tag{3.6}$$

$$\mathbf{R} = \mathbf{R}_{z,\phi}\mathbf{R}_{y,\theta}\mathbf{R}_{x,\psi}$$

2. From axis-angle to rotation matrix — considering a rotation of $\theta$ degrees around axis $\mathbf{k} = [k_x, k_y, k_z]^T$:

$$\mathbf{R}_{k,\theta} = \begin{bmatrix} k_x^2 v_\theta + c_\theta & k_x k_y v_\theta - k_z s_\theta & k_x k_z v_\theta + k_y s_\theta \\ k_x k_y v_\theta + k_z s_\theta & k_y^2 v_\theta + c_\theta & k_y k_z v_\theta - k_x s_\theta \\ k_x k_z v_\theta - k_y s_\theta & k_y k_z v_\theta + k_x s_\theta & k_z^2 v_\theta + c_\theta \end{bmatrix} \tag{3.7}$$

where $c_\theta = \cos\theta$, $s_\theta = \sin\theta$ and $v_\theta = 1 - \cos\theta$.

3. From quaternion ($\mathbf{q} = [q_r, q_i, q_j, q_k]^T$) to rotation matrix

$$\mathbf{R} = \begin{bmatrix} q_r^2 + q_i^2 - q_j^2 - q_k^2 & 2q_iq_j - 2q_rq_k & 2q_rq_j + 2q_iq_k \\ 2q_rq_k + 2q_iq_j & q_r^2 - q_i^2 + q_j^2 - q_k^2 & 2q_jq_k - 2q_rq_i \\ 2q_iq_k - 2q_rq_j & 2q_rq_i + 2q_jq_k & q_r^2 - q_i^2 - q_j^2 + q_k^2 \end{bmatrix} \tag{3.8}$$

4. From rotation matrix to euler angles —considering $\mathbf{R} = [r_{ij}]$ and X,Y,Z angles as $\psi, \theta, \phi$:

$$\begin{aligned}
\phi &= \operatorname{atan2}\left(\frac{r_{23}}{-\sqrt{r_{11}^2 + r_{12}^2}}, \frac{r_{33}}{\sqrt{r_{11}^2 + r_{12}^2}}\right) \\
\theta &= \operatorname{atan2}\left(r_{13}, \sqrt{r_{11}^2 + r_{12}^2}\right) \\
\psi &= \operatorname{atan2}\left(\frac{r_{12}}{-\sqrt{r_{11}^2 + r_{12}^2}}, \frac{r_{11}}{\sqrt{r_{11}^2 + r_{12}^2}}\right)
\end{aligned} \tag{3.9}$$

5. From rotation matrix to quaternion — considering $\mathbf{R} = [r_{ij}]$ :

- If Trace($\mathbf{R}$)$\geq 0$

$$\begin{aligned} q_r &= \frac{\sqrt{1+r_{11}+r_{22}+r_{33}}}{2} \\ q_i &= \text{sign}(r_{32}-r_{23})\frac{\sqrt{1+r_{11}-r_{22}-r_{33}}}{2} \\ q_j &= \text{sign}(r_{13}-r_{31})\frac{\sqrt{1-r_{11}+r_{22}-r_{33}}}{2} \\ q_k &= \text{sign}(r_{21}-r_{12})\frac{\sqrt{1-r_{11}-r_{22}+r_{33}}}{2} \end{aligned} \tag{3.10}$$

- If Trace($\mathbf{R}$)$< 0$ then pick the largest leading diagonal element. Assuming it is $r_{11}$ (works equivalently if is another element):

$$\begin{aligned} s &= 2\sqrt{1+r_{11}-r_{22}-r_{33}} \\ q_r &= \frac{(r_{32}-r_{23})}{s} \\ q_i &= \frac{s}{4} \\ q_j &= \frac{(r_{12}+r_{21})}{s} \\ q_k &= \frac{(r_{13}+r_{31})}{s} \end{aligned} \tag{3.11}$$

6. From quaternion to euler angles:

$$\begin{aligned} \phi &= \text{atan2}\left(2\left(q_r q_i + q_j q_k\right), 1-2\left(q_i^2+q_j^2\right)\right) \\ \theta &= \arcsin\left(2\left(q_r q_j + q_k q_i\right)\right) \\ \psi &= \text{atan2}\left(2\left(q_r q_k + q_j q_j\right), 1-2\left(q_j^2+q_k^2\right)\right) \end{aligned} \tag{3.12}$$

7. From axis-angle to quaternion — $\theta$ degrees around axis $\mathbf{k}$:

$$\mathbf{q} = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}\hat{\mathbf{k}} \;\; , \texttt{with} \;\; \hat{\mathbf{k}} = \frac{\mathbf{k}}{\|\mathbf{k}\|} \tag{3.13}$$

## 3.3 3D computer vision

The artificial vision processes start from an image frame captured by a camera. The scheme on Fig. 3.4 exemplifies the notation for the axis of the image frame: horizontal and vertical axis $u$ and $v$, respectively, are the basis of the image space; a pixel $\mathbf{p}$ is defined by its coordinates $\mathbf{p} = [u_p, v_p]^T$. The value of the pixel is denoted $F(u, v)$ and can be one dimensional (grey-scale image) or three dimensional (colour image). The origin of the image space is at the top left corner. In the example, the image resolution is $8 \times 8$ pixels.

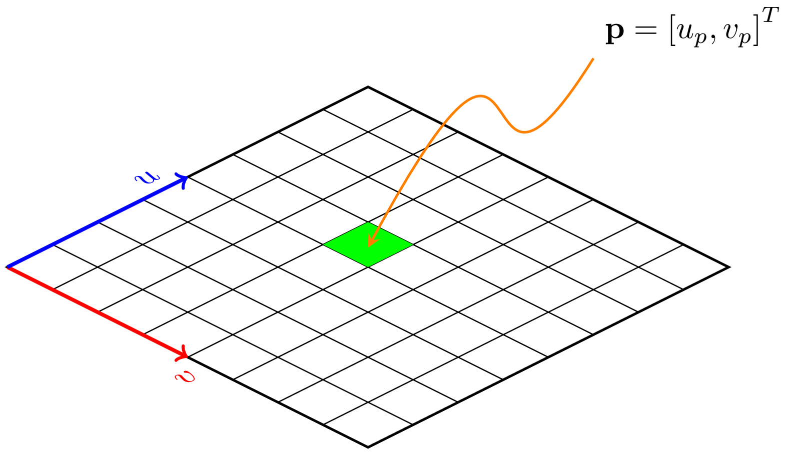


**Figure 3.4: Image Notation -** The standard frame used in image processing. Pixels are identified by their position $(u, v)$ in the image frame/matrix.

### 3.3.1 Camera Model

In order to understand the phenomena of capturing a scene into an image (using a camera and lenses) a model has to be established. The camera *pinhole* model is arguably the most commonly used to that end. Fig. 3.5 holds a scheme from which the *pinhole* model can be understood and derived. It synthesizes the projection of a world point into the image frame and how the different coordinate frames are related.

The required notation:

1. $M$ is a 3D world point and has coordinates $\mathbf{m} = [m_x, m_y, m_z]^T$, with respect to the world frame $\mathcal{W} = \{x_w, y_w, z_w\}$ ;

2. The corresponding pixel to $\mathbf{m}$ is $\mathbf{p}_m = [u_m, v_m]^T$;

3. The image plane $\mathcal{I}$ has a frame $\{\hat{x}, \hat{y}\}$ centred in $\mathbf{p}_0 = [u_0, v_0]^T$ ( the principal point); the usual frame for image processing (due to the matrix representation) is at the top left corner: $\{u, v\}$

4. The focal plane $\mathcal{F}$ has the camera coordinate system $\{x, y, z\}$, centred in $\mathbf{c}$ — called the optical centre.

5. The orthogonal distance between the planes $\mathcal{F}$ and $\mathcal{I}$ is $f$, the focal length; the line that connects $\mathbf{p}_0$ and $\mathbf{c}$ is the optical axis.

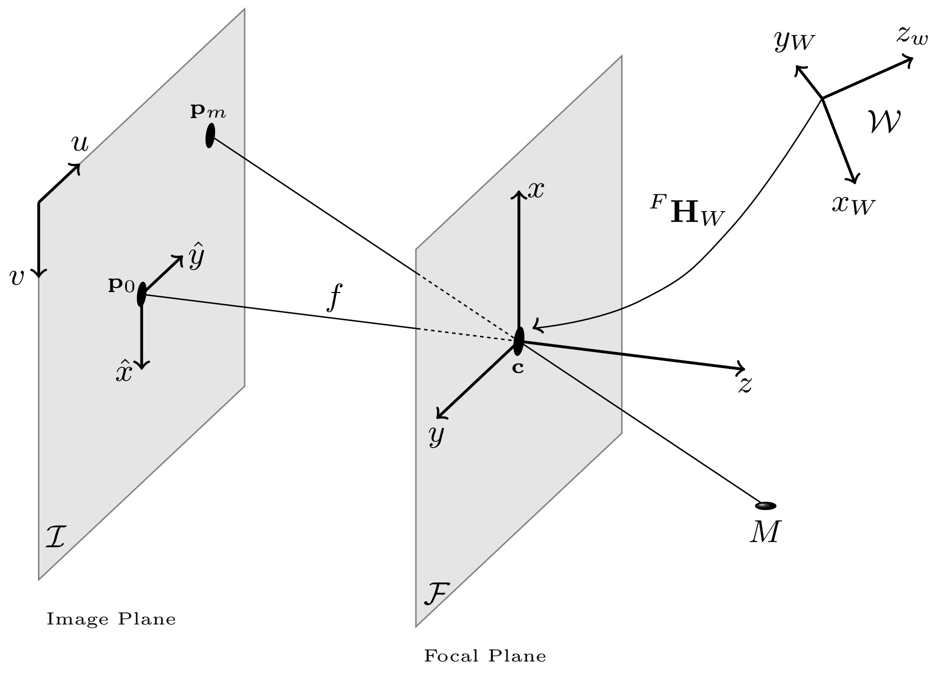


**Figure 3.5: Camera *Pinhole* Model** - The geometric relationship in a perspective projection. The *pinhole* model can be derived from this scheme (based on [58]).

The projection of $\mathbf{m}$ into $\mathbf{p}_m$ happens in two moments: first there is the transformation of world coordinates into camera's coordinates; then, the actual projection occurs. As for the former, it is necessary to describe the position of the camera in the world frame, i.e, specify $\mathbf{c}$ in terms of $\mathcal{W}$; this is accomplished by the translation vector ${}^F\mathbf{t}_W$. The alignment of the frames, i.e, $\{x, y, z\}$ matching with $\{x_W, y_W, z_W\}$, is achieved by a rotation: ${}^F\mathbf{R}_W$. The homogeneous transformation

$$
{}^F\mathbf{H}_W = \left[ \begin{array}{c|c} {}^F\mathbf{R}_W & {}^F\mathbf{T}_W \\ 0_3 & 1 \end{array} \right] \tag{3.14}
$$

holds the so called *extrinsic parameters* of the camera: how it is positioned with respect to the world.

Regarding the 3D→2D transformation, it is described by the camera's *intrinsic* *parameters* matrix

$$\mathbf{A} = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.15}$$

which uses the principal point,$[u_0, v_0]$, horizontal/vertical scale factors describing the actual size of the pixels,$\alpha$ and $\beta$, and $\gamma = \alpha \cos\theta$ – the parameter of non-orthogonality between $u$ and $v$.

The perspective projection model, through the pinhole scheme, is then given by the projection matrix $\mathbf{P}$, which takes into account both intrinsic and extrinsic parameters. Equation 3.16 (assuming homogeneous coordinates), shows the final $\mathbf{M} \rightarrow \mathbf{p}_m$ relationship.

$$\mathbf{p}_m = \mathbf{Pm} \quad , \quad \text{where} \quad \mathbf{P} = \mathbf{A}^F\mathbf{H}_W \tag{3.16}$$

### 3.3.2 Stereoscopy: 3D from two-view geometry

A single camera captures a world scene using an unequivocal $3D \rightarrow 2D$ mapping, i.e, world point $M$ can be seen at pixel $\mathbf{p}$ and at no other; on the other hand, pixel $\mathbf{p}$ can be the image of an infinite set of world points: those along the line connecting optical centre $\mathbf{c}$ and $M$. This means that the image frame has no depth information.

To recover depth, at least two cameras are needed. When a world point is simultaneously captured by two cameras(placed at distinct positions), a restriction arises that allows going from the 2D world into 3D. Fig. 3.6 shows a scheme of the stereoscopy principles. Point $M$ appears in image $\mathcal{I}$ as $\mathbf{p}$ and in image $\mathcal{I}'$ as $\mathbf{p}'$.

The required notation:

1. The left camera image is $\mathcal{I}$ and the right is $\mathcal{I}'$
2. The superscript "prime" ($'$) indicates variables of the right camera;
3. $M$ is a 3D world point;
4. $\mathbf{p}$ and $\mathbf{p}'$ are the pixels corresponding to of the image of $M$ in $\mathcal{I}$ and $\mathcal{I}'$, respectively;
5. $\mathbf{c}$ and $\mathbf{c}'$ are the optical centres. They are displayed behind the image plane for a matter of simplicity; it is mathematically equivalent to the real case where they lie in-between the world point and the image plane (as represented in the pinhole model, in Fig. 3.5);

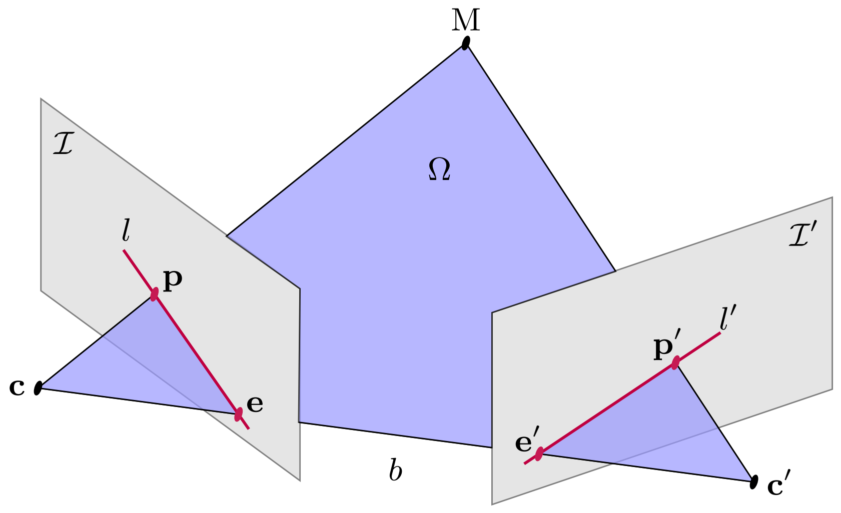


**Figure 3.6: Stereoscopy -** The geometric principles of two-view triangulation – known as epipolar geometry. The possible matching pixels to $\mathbf{p}$ lie on epipolar line $l'$ and vice-versa. This provides a useful geometric restriction used image analysis in order to find the corresponding pairs. *Remark*: The camera centres are drawn behind the image scene for simplicity and ease of interpretation. It is mathematically equivalent to the notation of Figure 3.5

6. $\mathbf{e}$ and $\mathbf{e}'$ are called the epipoles; they stand as the projection of the other camera's centre.

7. the line $b$ is called baseline; it connects the two centres and can be seen as the optical ray that simultaneously projects $\mathbf{c}$ into $\mathcal{I}'$ and $\mathbf{c}'$ into $\mathcal{I}$. The projection pixels are the epipoles ( $\mathbf{e}/\mathbf{e}'$).

8. Lines $l$ and $l'$ are called the epipolar lines. The potential matches to pixel $\mathbf{p}$(or $\mathbf{p}'$) lie on epipolar line $l'$ (or $l$);

9. The plane $\Omega$ is called the epipolar plane;

Stereo is possible when both the relative positioning of the two cameras is known, and when there is a corresponding pair of pixels $\langle \mathbf{p}, \mathbf{p}' \rangle$.

Epipolar geometry ([58] and [59]) provides the tools to relate cameras positioning and to generate 3D from the pixels: any corresponding pair of pixels $\langle \mathbf{p}_i, \mathbf{p'}_i \rangle$ satisfy 3.17:

$$\mathbf{p'}_i^T \mathbf{F} \mathbf{p}_i = 0 \tag{3.17}$$

$\mathbf{F}$ is the fundamental matrix. It encapsulates both camera's intrinsic parameters and their relative pose, i.e, the geometry relationships present on Fig. 3.6.

Finding a matched pixel pair involves image analysis with the help of epipolar geometry. Given a pixel $\mathbf{p}$, the corresponding pixel $\mathbf{p}'$ sits along the epipolar line $l'$. $l'$ is computed from the fundamental matrix $\mathbf{F}$:

$$\begin{gathered} l' = \mathbf{F}\mathbf{p} \\ \mathbf{p}' \in \mathbf{l}' \Rightarrow \mathbf{p}'\mathbf{l}' = 0 \\ \therefore \mathbf{p}'\mathbf{F}\mathbf{p} = \mathbf{0} \end{gathered} \tag{3.18}$$

So, Equation 3.17 resolves into a line equation which defines $l'$ given a pixel $\mathbf{p}$. All the pixels $\mathbf{p}'_i$ belonging to $l'$ are possible matches to $\mathbf{p}$. So, epipolar geometry gives a search constrain (a line among the entire image). Pixel features (as colour, texture, etc) must decide if the pairs match — this is called the correspondence problem.

With the above data it is finally possible to recover depth data from two camera views. The same concept is extensible to N-camera views, doing the possible 2-combinations of cameras.

### 3.3.3 Camera/Stereo Calibration

Camera calibration refers to the method of identifying/estimating the parameters of the projection matrix **P** – equation 3.16. A complete calibration holds the intrinsic parameters of the camera as well as its positioning according to a given world referential. Available options for calibration include (based on [58]):

1. Self-calibration: moving the camera in a rigid scenario and finding corresponding sets of points (on a least three images) provides enough data and constraints to estimate all the intrinsic and extrinsic parameters. The mathematical formulation of this problem is harder than those of the next techniques. Self calibration can be categorized as a 0D method.

2. Calibration based on a 1D line: the camera captures a set of calibration points that are displaced along a line; the points are moved around, maintaining the linear restriction; this can be achieved with a bar with a set of markers attached to it. Due to the linear restriction, this is categorized as a 1D method;

3. Calibration based on 2D patterns: this is arguably the most used method. Usually, a checker-board pattern is shown to the camera – Fig. 3.7 – at different poses. The edges of the squares are found and, since the size of the real pattern is known, the correspondence between pixels and real coordinates enables a complete calibration of the camera's parameters.

4. Calibration based on known 3D objects: as an upgrade to the previous method, this one relies on a 3D pattern or a 3D object whose dimensions are precisely known. The most simple form is to place two or three 2D-checker-board patterns orthogonally to each other and, again, retrieve the square corners and match them with the world coordinates.

Regarding stereo calibration, it can be achieved by using the estimated models of the cameras. If each has been previously calibrated (known **P** and **P′**), then **F** can be computed from a closed form solution — from Hartley and Zisserman [59]:

$$\mathbf{F} = \left[\mathbf{P}'\mathbf{c}\right]_{\times}\mathbf{P}'\mathbf{P}^{T}\left(\mathbf{P}\mathbf{P}^{T}\right)^{-1} \tag{3.19}$$

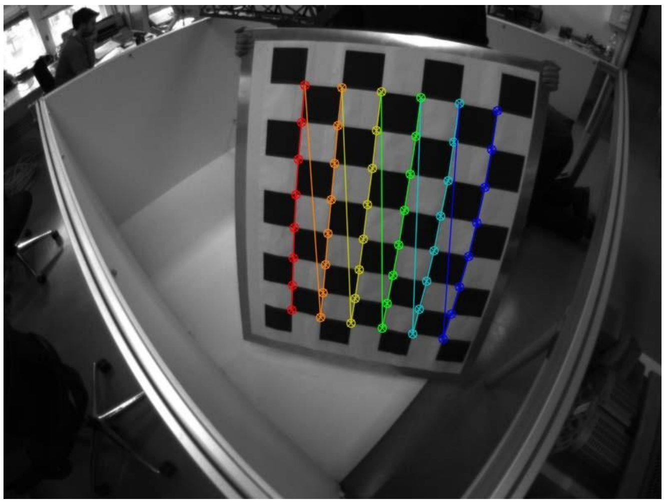


**Figure 3.7: Calibration Pattern -** A checker-board pattern used for camera calibration. It is the most used method as it simply involves printing a known-size black and white squared pattern. The required image processing is also minimal — finding the black-to-white transitions and detect the corner.

The $[\mathbf{a}]_\times$ operator denotes the skew-symmetric matrix corresponding to vector $\mathbf{a}$:

$$[\mathbf{a}]_\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}$$

The other option is to obtain a set of at least 8 corresponding pixel pairs. The techniques presented above remain valid to obtain a list of corresponding points. From the stereo equation 3.17 it is trivial to solve a set of homogeneous equations with respect to the matched pairs and directly estimate the fundamental matrix $\mathbf{F}$. Recovering $\mathbf{P}$ and $\mathbf{P}'$ from $\mathbf{F}$ is also possible [59].

A calibrated stereo system, working along with image processing methods that match pixel pairs, ultimately allows going from 2D images to 3D world coordinates.

### 3.3.4 Tracking Luminous Markers — The *sincrovision*

The *sincrovision* concept was developed and patented [60] in the University of Porto - Faculty of Engineering . It implements a system of 3D acquisition based on stereoscopic vision synchronised with high intensity luminous markers. The key idea is to turn on the markers as soon as the cameras start acquiring the image and turn them off after the camera exposure time has expired. Figure 3.8 shows a timing diagram of the system.

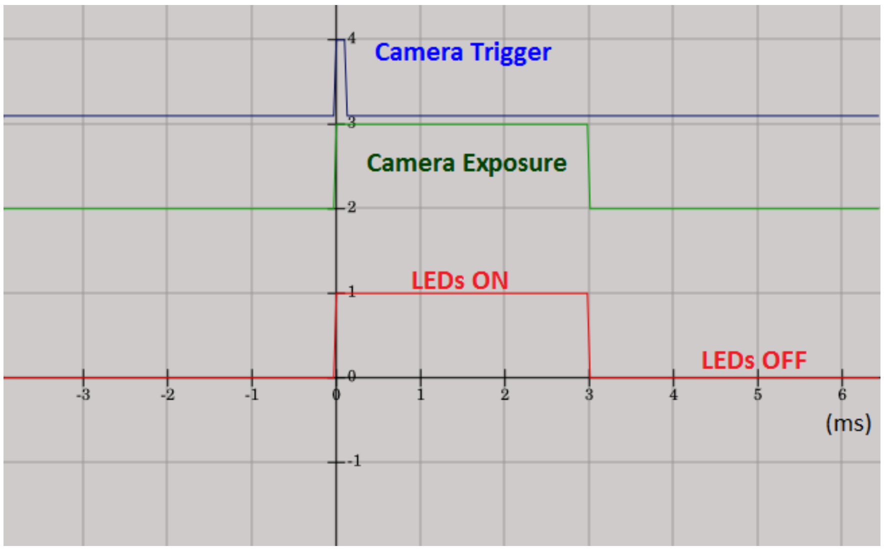


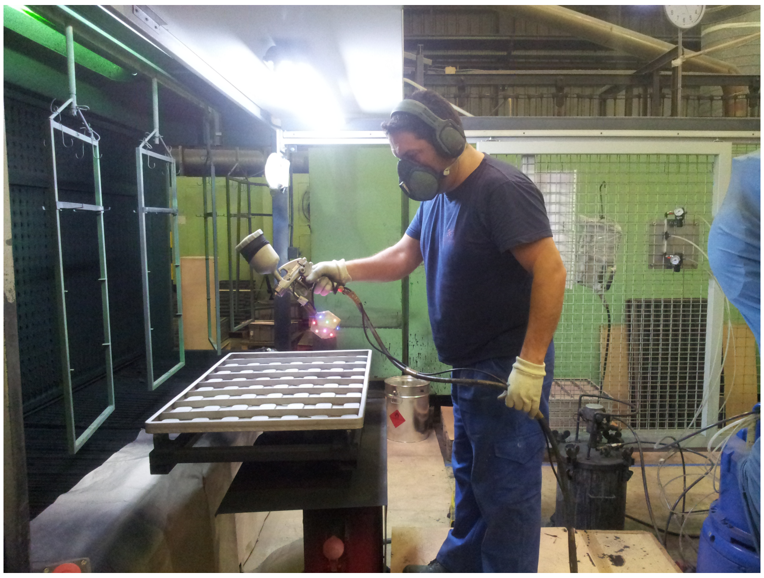
**Figure 3.8: *Sincrovision* Timings -** A timing diagram depicting the synchronous image acquisition. Both cameras are triggered at the same time. From that moment on they are acquiring the scene for a very limited time(3ms). The ultra-bright LEDs are triggered at the same instant the cameras were triggered too and remain lit only during the camera exposure period.

Together with a fully closed aperture of the camera's lenses, the high intensity lights will be very bright on the images whilst the background noisy data will have no time to be acquired by the camera. At the same time, blinking the markers for a short time makes it possible to stare at them; keeping them always on would cause eye damage. This setup makes it possible to triangulate the markers positions in space in a robust way, independently of lighting conditions in the scene and ignoring most of the common noise sources in artificial vision applications. Figure 3.9 shows a typical image captured by the cameras using this synchronous feature.

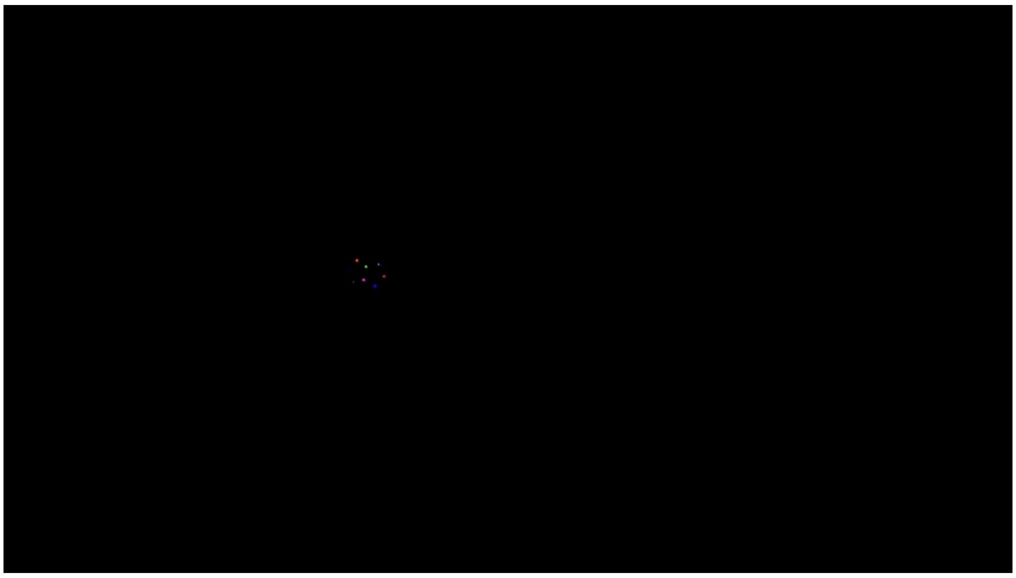
The *sincrovision* will be used as the basis of the proposed new input device for human movement capture, described in the next chapter.

## 3.4 Industrial Manipulators

The research reported in this thesis was supported by an industrial demonstrator, as pointed in the introductory chapter. The experiments and validation tests were performed on an industrial manipulator that has been acquired by this demonstrator,



(a)



(b)

**Figure 3.9:** (a) **Standard acquisition**: The scene is captured with no timings constraints. The exposure and lense aperture make the image too dirt to be processed. The background lighting obfuscates the marker. There are many noise sources, mainly the fluorescent lamp. (b) **Synchronous acquisition**: The lenses aperture are reduced to a minimal. Exposure is very low (around $3ms$). The background noise is eliminated and the markers shine robustly in the image. Under these conditions, clustering is straightforward.

exactly for the purpose of implementing the skill transfer framework. Without loss of generality, the methods for interfacing and programming the robot were developed aiming a MOTOMAN PX2050 robot.

This is an articulated manipulator, with 6 degrees of freedom (DoF) and a payload of 10Kg. Precision/repeatability (how close to the same position the robot end-effector is able to move to) is rated at $\pm 0.5mm$. In terms of accuracy (when the robot moves to a pre-defined point in space, how close it gets to the desired coordinates), manufacturers do not provide such data; this is some times referred to as volumetric accuracy, and usually falls down to a millimetre or two. It is due to this volumetric inaccuracy that online jogging and teaching is preferred over offline programming in high precision operations.

For a real-time control of the robot, a TCP/IP interface with the robot controller can be used. A specific network protocol must be implemented in order to control robot movements, read its state and upload offline generated programs.

As for the programming language, there are three types of movements which are also common across every brand of industrial robots:

1. Joint moves, or MOVJ: this instruction uses joint interpolation for moving the robot. This means that it travels from point $A$ to point $B$ without a particular path restriction; the controller moves each joint independently in order to achieve the desired position. This type of control must be used with caution since there is no *a priori* knowledge about where the tip or each link is going to move to. Only the final positioning is assured.

2. Linear moves, or MOVL: this type of control instructs the robot to go from position $A$ to $B$ using linear interpolation in the cartesian space, i.e, the robot will draw a straight line from $A$ to $B$;

3. Circular moves, or MOVC: this instruction controls the robot to perform a circular movement; it is usually fed by enough data points (3) to define a circumference in 3D space — the centre, the arc starting point and the arc ending point.

Together with the most basic movement instructions there are also a number of parameters that can be controller for a refined motion control such as acceleration, deceleration and tolerance (how close to the desired point the robot goes).

A robot program is but a collection of successive movement instructions (either of the presented) with sensor readings and actuation signals set in-between those instructions.

Chapter 4

# Motion Imitation Framework

This chapter presents the methodology, the design and implementation of the human-robot skill transfer framework.

The first section is a brief snapshot of the proposed solution. It exposes all the modules and how they interface each other. From here, the remaining sections provide an in-depth analysis and description of each subsystem. The structure and ordering of the sections resemble the actual system flow on the real application: the motion capture/tracking, the data processing and the interface with the robot.

## 4.1 System Architecture : Overview

This research is conducted towards a feasible industrial solution: the development of all system modules takes in concern implementation costs, reduction of hardware apparatus and cutback of setup and processing times. It is also important to minimize the impact on the production process, on the shop floor and ultimately on the operator task.

The proposed solution puts together a tracking framework and a set of routines for data processing which aim to enable/simplify the interface and playback with an industrial manipulator.

The *sincrovision* (see *Background* section 3.3.4) offers the possibility to develop a new kind of marker for gesture tracking. The technique is based on multi-view artificial vision and requires luminous markers. Using cameras to capture movement presents numerous advantages: it is a contactless method, operators do not have to wear extra devices (uncomfortable and unusual to their daily tasks), calibration is permanent as long as cameras remain unmoved. Also, there is a wide range of cameras with varying resolution and price. So, for a specific application there are many options to chose from available on the market. Furthermore, stereoscopy can be used starting with only two cameras (low cost) but more can be added as necessary to achieve the desired precision or to cover larger workspaces. This flexibility makes the stereoscopy based solution available to many industrial applications like painting, welding, bending, polishing, etc.

From an industrial cell point of view, the required changes to install a stereoscopy based tracking are small:

- Install a set of cameras;
- Attach the marker to the work tool;
- Connect the cameras to a PC and deliver power to the marker.

The added hardware apparatus is reduced to a cabinet that can hold the stereoscopic setup (a set of industrial cameras and the synchronization device). The robot and the human worker operate in the same area: first, the operator demonstrates then he is replaced by the robot that is going to mimic the operation. The marker must be attached to the operator tool. In the following sections, the design considerations are described in order to achieve a robust and easy to plug luminous marker.

The skill transfer framework limits how the human interacts with the robot, i.e, provides abstraction from the programming language but also from the teach pendant. A minimal interaction with a computer is needed nonetheless: to start and stop the demonstration, to calibrate the system and to playback desired programs.

The framework is divided into the following stages:

1. **Calibration**. In this step, two calibrations take place: the camera↔robot calibration and the human-tool↔marker↔robot-tool calibration. The former needs the operator to mount a calibration tool on the robot end-effector; after that, the robot executes a program with no further human interaction. When the program ends the stereo is calibrated. Human-tool↔marker↔robot-tool calibration needs the operator to hold the work tool in front of the calibrated cameras. Calibration in done from an image frame of the scene. From this point, the tracking system

is ready to perform (these steps are detailed in the forthcoming sections). It is important to note that the system calibration only needs to be done once as long as the cameras remain in the same spot relative to the robot. The procedures can be done during the actual installation of the system, so it is transparent to the end user.

2. **Demonstration**. Through a computer interface, the operator starts the demonstration process. It takes a single click of a button. Then, he executes the movements and, at the same time, the data is captured and processed. After finishing the demonstration, the final trajectory processing is carried out and the robot program is generated.

   In this stage, the operator needs only to start and stop the process through the PC interface. The demonstration happens in the most natural way, as the operator executes a task as he would normally do. There is no use of an extra tool or any change to the operators' behaviour.

3. **Playback**. Again, using only a software interface, the operator chooses a robot program. The code is transferred to the robot's controller and the movement mimic is executed. The operator needs only to evaluate the machine performance by inspection.

The motion imitation is achieved in a short time. The image processing is done in real time; the full trajectory analysis and robot code generation are also just a few seconds long which make the robot almost instantaneously ready to mimic the operator. The stereoscopy tracking requires camera calibration and a luminous marker; the marker itself must be able to follow the tool path and retrieve its position and orientation in space; the data must be processed in the shortest time possible; also, it must be optimized to interface with an industrial robot, i.e, limited to the movements the machine has available. These features, the design and implementation details that make this a feasible industrial system, are presented in the next sections.

## 4.2 Camera and Stereo Calibration

As depicted in Chapter 3, camera calibration is a vastly researched subject. There are many available tools (software libraries) that implement the required routines that estimate the camera model parameters. Most of them are based on a list of points retrieved from the checker-board patterns and implement either the Tsai calibration method [61] or the more recent Zhang's method [62].

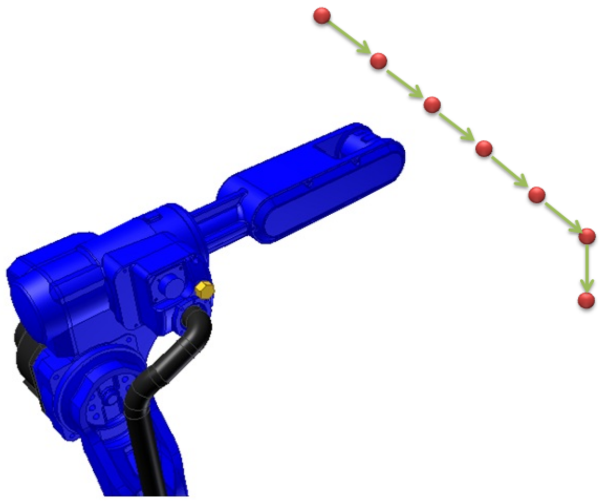
The *sincrovision*, however, poses a problem to the use of checker-board patterns. Since the lenses' aperture is shut to its maximum, it is very hard to observe the pattern. Even if the exposure is maximized, the images are too dark to be processed. It would require intense adaptive thresholds or added hardware apparatus to use light flashes.

A new method to retrieve $2D \leftrightarrow 3D$ correspondences has been developed. First, the *sincrovision* is expected to be used with intense synchronous markers. So, the options are to build a checker-board with as much LEDs as there would be square corners in a printed pattern or to show a single LED at different known positions. Additionally, 3D calibration methods (recall the review on 0/1/2/3D calibration techniques presented in Chapter 3, section 3.3.3) are known to be very efficient but are sometimes disregarded since they require more apparatus and precisely prepared setup. To overcome this limitation the proposed technique uses the industrial manipulator to calibrate the cameras and the stereo. The robot can move in 3D, can hold the synchronous LED required by *sincrovision* and has high precision.

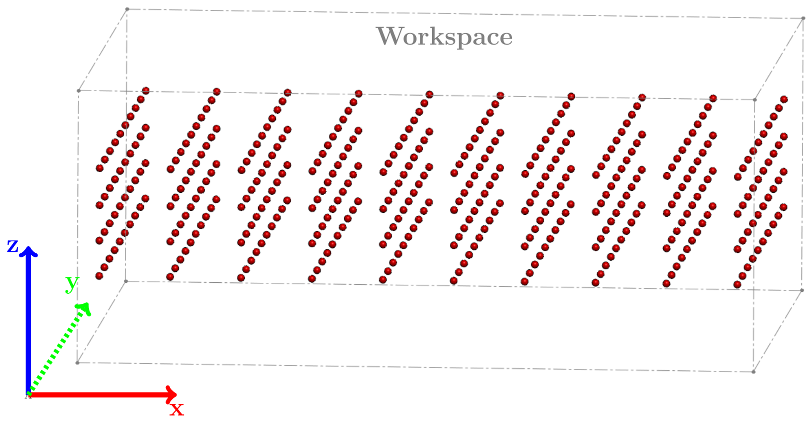
With this approach, the LED attached to the robot's end-effector describes a path with well known way-points. At these control points, the LED is lit and captured by the cameras. The world position is given by the robot controller and the LED is captured simultaneously by the cameras so building the $2D \leftrightarrow 3D$ correspondence list is straightforward. The robot has the ability to move and draw any shape (for instance, a cube), which simulates the use of two or three orthogonally positioned checker-board patterns. In addiction, the robot can move inside that cube which means the calibration method can have point/pixel correspondences from all over the workspace without extra hardware.

Figure 4.1(a) shows an industrial robot performing the calibration path. It moves in a grid pattern and the LED is lit at a regular interval. Figure 4.1(b) has the final calibration grid, after the robot has moved in the entire workspace. The grid density is easily adjusted to have extra points for a more robust calibration or to have few and take less time.

With the pixel–world-point correspondence list, the estimation method proposed by Faugeras [63] can be followed: it is a linear estimation of the camera's projection



(a)



A

(b)

**Figure 4.1:** (a) **Robot Path**: The manipulator moves in an incremental path and the LEDs are turned ON at regular intervals (red dots). Care is taken that all positions are visible to both cameras. (b) **The final calibration grid**: After the robot has moved around the entire workspace, the result is a calibration grid with 3D points with have also been captured by both cameras. This data can be used for camera and stereo calibration and is already described in robot coordinates.

matrix. Recall the camera model and the perspective projection equation (equ.3.16):

$$\mathbf{p}_m = \mathbf{P}M \quad , \quad \text{where} \quad \mathbf{P} = \mathbf{A}^F\mathbf{H}_W$$

The set of $n$ points in space, $M_i = (X_i, Y_i, Z_i, 1)$, captured by a pair of cameras generates a set of $n$ conjugate pairs of pixels $(\mathbf{p}_i, \mathbf{p'}_i)$ (corresponding pixels). For each camera we have $n$ $2D \leftrightarrow 3D$ mappings $\mathbf{p_i} = (u_i, v_i) \leftrightarrow M_i = (X_i, Y_i, Z_i, 1)$ which, according to equ. 3.16, originate:

$$\begin{bmatrix} X_i & Y_i & Z_i & 1 & 0 & 0 & 0 & 0 & u_iX_i & u_iY_i & u_iZ_i & u_i \\ 0 & 0 & 0 & 0 & X_i & Y_i & Z_i & 1 & v_iX_i & v_iY_i & v_iZ_i & v_i \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & u_nX_n & u_nY_n & u_nZ_n & u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & v_nX_n & v_nY_n & v_nZ_n & v_n \end{bmatrix} \mathbf{p} = \mathbf{0} \tag{4.1}$$

where $\mathbf{p} = [p_{11}, p_{12}, \ldots, p_{34}]^T$ and $\mathbf{0}$ is an $1 \times 2n$ null vector.

This system of homogeneous equations $\mathbf{Gp} = \mathbf{0}$ can be solved using linear estimation. The error produced by this procedure is sub-pixel and acceptable for human hand motion capture. The least squares solution is as usual the singular vector associated with the smaller singular value of $\mathbf{G^TG}$, i.e, the last column of $\mathbf{V}$ from SVD($\mathbf{G}$) $= USV^T$.

To calibrate the stereo from the known $\mathbf{P}$ and $\mathbf{P'}$ matrices, the relation from Equ. 3.19 is used:

$$\mathbf{F} = [\mathbf{P'c}]_\times \mathbf{P'P}^T \left(\mathbf{PP}^T\right)^{-1}$$

This calibration method has the added benefit that cameras and stereo are calibrated in the robot's coordinate frame. No additional frame translation and rotation is needed. When computing a 3D position from the images, the coordinates are directly used in the robot controller. Also, the manipulator is already supposed to exist, i.e, is an available asset as it integrates the motion imitation framework. Using it reduces the cost of having to develop hardware and complex mechanics that could provide a precise positioning of the 3D pattern.

More so, this calibration technique avoids the major positioning accuracy error on offline programming of industrial manipulators. These robots execute with great precision but a mediocre accuracy — see Fig. 4.2, from [64]. Precision, or repeatability, describes how close the robot can move back to a given position. Usually this indicator is provided by manufacturers and can be lower than 0.1mm. Accuracy defines the ability to move to some exact 3D coordinates in space. Manipulators can move to the same position over and over again with great precision yet they may still fail the desired position by a considerable amount. Usually manufacturers do not provide this indicator. Robot manufacturer ABB claims that their own robots may have a volumetric error of about 5 to 15 mm [65]. For offline programming solutions (where absolute positions/coordinates are used to generate programs) this feature may jeopardize the task performance as the robot moves to the simulated positions but adds a considerable error. By the proposed method for camera calibration, the camera model inherits the error pattern of the robot positioning. Thereafter, the offline programming based on coordinates read from camera images significantly reduces this interference from the robot inaccuracy.

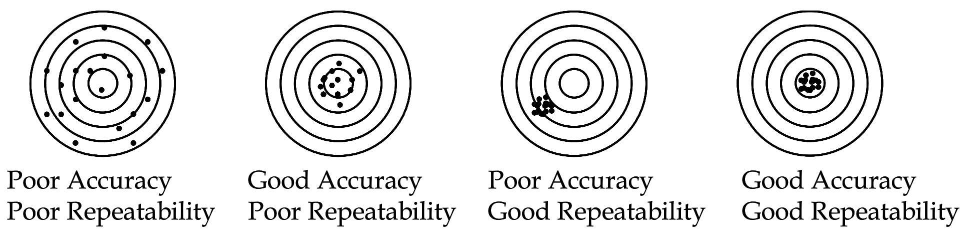


**Figure 4.2: Precision/Repeatability vs Accuracy** - Interpretation for the concepts of repeatability and accuracy. Industrial manipulators report to the 3rd example: they provide a good repeatability but volumetric accuracy is less feasible.

## 4.3 Motion Tracking

### 4.3.1 6-DoF Marker

This section describes a marker — hereafter referred to as luminous marker, icosahedron or simply marker — for a complete 6D (six degrees of freedom: position, $(x, y, z)$, and orientation, $(\phi, \theta, \psi)$ ) motion tracking.

As pointed earlier, the major concerns on the development of this tracking tool were to achieve a device that provides an accurate measure of the pose of the human hand/tool while maintaining costs low, reduced processing times and low impact on the process and on operator movements.

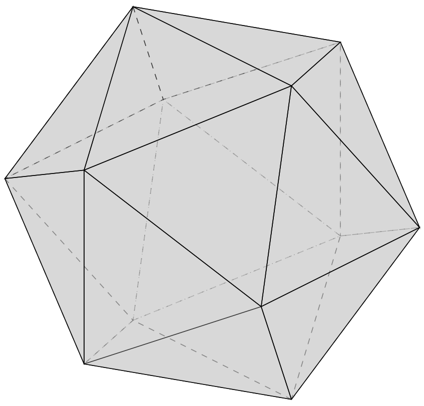
The description section that follows provides details on the choosing of the size and geometry of the marker. The later sections dissect the *modus operandi*, with an in-depth analysis of the tracking scheme — pose estimation and marker$\leftrightarrow$tool calibration. Finally, the limitations of this new input device are discussed.

#### 4.3.1.1 Description: Hardware and Properties

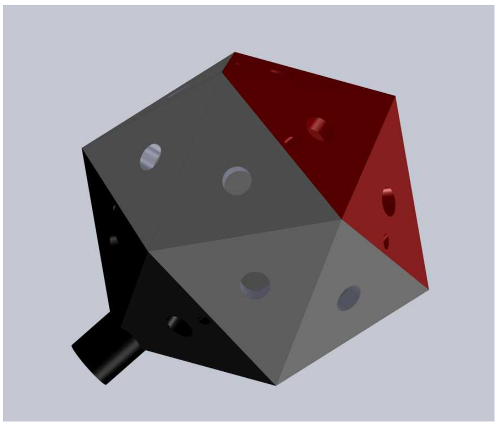
The *sincrovision* process demands high intensity luminous markers to be turned on simultaneously with the camera acquisition window. For that it was chosen to build a marker based on high-power/high-brightness LEDs. These devices are simple to command and offer high versatility.

In order to capture full 3-D orientation, at least three non-collinear LEDs are needed. Nevertheless, such a scarce number of lights would fail to provide a complete freedom of movements to the end-user: all of those individual markers should have to be visible at all times on both cameras otherwise pose estimation would fail due to occlusions. Increasing the number of cameras around the working area can fight back this problem but at a greater financial cost. In similar approaches ([44], [41],[46],[47] — already discussed in chapter 2) researchers used infra-red LEDs; they could be distinguished from one another through the distance as they were placed at different lengths from each other. As the number of individual LEDs increase to allow a full 360° cover on the 3 axis, playing with inter-LED distance is harder as it is detection of the lights on the image. On this line of reasoning, the proposed marker is based on 20 multiple visible-light (RGB) LEDs. These are distributed in a special manner, based on the shape of an icosahedron — see Fig. 4.3 — , as it showed to provide an interesting set of properties that aid in constructive and algorithmic aspects:

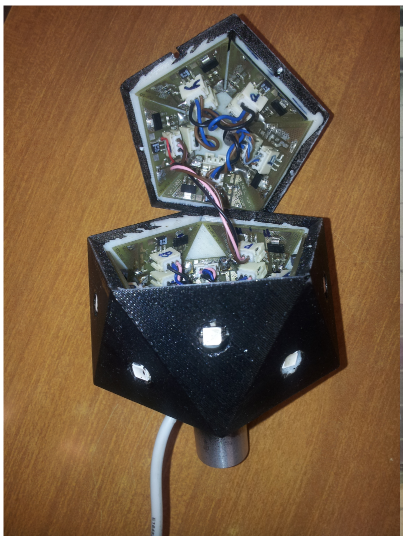
- Placing the LEDs on the centre of each face covers the intended full 360 degrees rotations in every axis; there are always enough visible lights on both cameras so that it is possible to compute orientation and position. The number of cameras can then be kept to a minimum (for stereoscopy) of two.
- The dodecahedron/icosahedron are regular (Platonic) polyhedra: all vertices/face centres lie on a sphere — Fig. 4.4 (b). Knowing the position of each vertex one can find the centre of the marker using sphere fitting (this is the circumsphere or circumscribed sphere). This property guarantees the symmetry of the LED



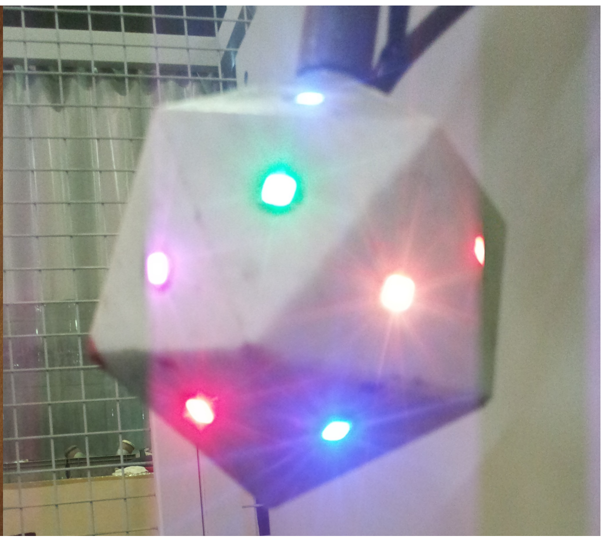
**(a)**



**(b)**



**(c)**



**(d)**

**Figure 4.3:** (a) **Polyhedron 3D view**: A 3D representation of an icosahedron. (b) **CAD**: the CAD model of the designed marker. (c) **Construction Details**: details on the developed marker; cabling, electronics and the LEDs are visible; the top hatch is held by screws to allow access to the interior. (d) **Lights ON**: real marker with LEDs in ON state. The camera which took the shot was not synchronized; also, the typical exposure time used in a daily use camera makes the LEDs very bright and the sensor reaches saturation. Is it hard to distinguish red from yellow or red from purple, even though the colours are quite different at naked eye. Moreover, they are perfectly distinguishable from the computational point of view.

positioning. The icosahedron dual polyhedron, the dodecahedron, has vertices touching the centre of icosahedron faces so it is equivalent to work with one or the other — Fig. 4.4 (a).

While the icosahedron shape provides housing for the hardware, the equations of the dodecahedron vertices are easier to work with so the properties of these two polyhedra are used simultaneously.

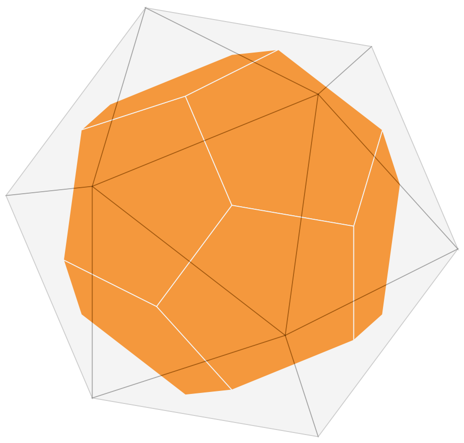
- The spherical symmetry is useful for algorithmic purposes as will be described ahead. Furthermore, it completely decouples the estimation of position and orientation of the marker; the translation is given by the sphere centre which is invariant under any rotation.

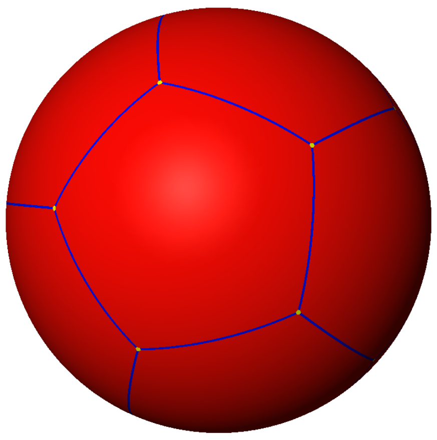
- Regarding construction issues, this polygon makes a balanced object that easily attaches to an industrial tool — Fig. 4.5 (a) shows the marker attached to an industrial spray-paint gun for coating applications; Fig. 4.5 (b) shows the marker attached to industrial suction cups for bending applications.

  Also, the empty space in its interior makes room for electronics, LEDs' supports and cabling. The polyhedron faces, unlike a sphere shaped surface, are ideal to pin a PCB holding one LED.

- Once the LEDs are found in the vertices of the dodecahedron, it is possible to see them on a grid/net pattern: the dodecahedron schlegel diagram — Fig. 4.4 (c). From this planar graph, one can colour de vertices using only 5 distinct colours and obtain unique sets: a vertex and its 3 neighbours are unique throughout the whole marker. This property is used to estimate orientation as detailed ahead.

Regarding electronics, each icosahedron face holds a triangular PCB with an RGB LED on its centre. Connections of the semi-conductor are standard to achieve the required 5 different colours: a series resistor on one input pin (primary colour) or in two (secondary colour), a supply capacitor to support the brief turn-on current surge and an electronic switch to enable the synchronized acquisition. Since the LEDs are ON only by a few milliseconds, no heat dissipation is needed reducing the required hardware and costs. The whole electronic apparatus (Fig. 4.6) is minimal and low-cost: it needs a synchronization device to generate the clock signal (a microprocessor


(a)


(b)

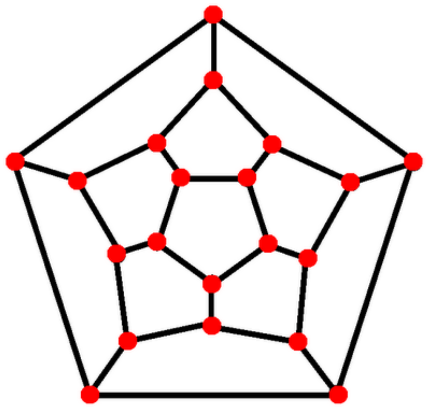
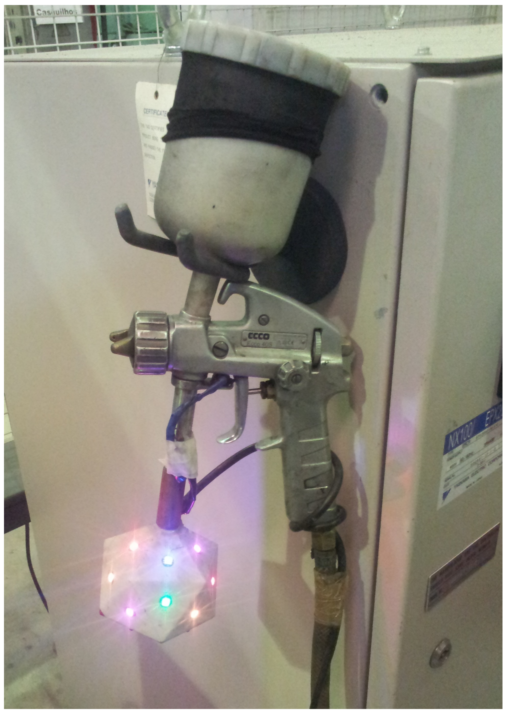
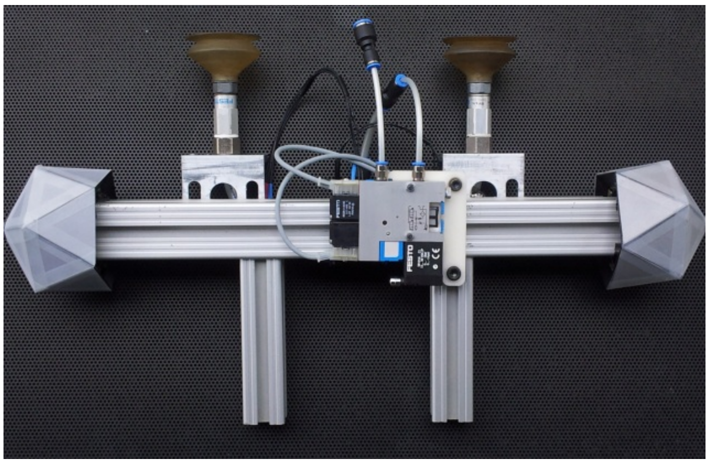

(c)

**Figure 4.4:** (a) **Dodecahedron**: The dodecahedron inscribed in an icosahedron. The marker outline follows the icosahedron shape while the LEDs are positioned on the vertices of a dodecahedron that touch the face centres of the former. (b) **Circumscribed sphere**: the dodecahedron circumscribed sphere. It shows how the polyhedron vertices lie on a spherical surface. (c) **Schlegel diagram**: a planar representation of the dodecahedron where no edges cross each other — called a schlegel diagram. It is useful for accessing the distribution of the LEDs are understand their relative positioning



**(a)**



**(b)**

**Figure 4.5:** (a) **Spray painting gun**: The marker is attached to an industrial spray gun. The balanced/symmetric shape does not change the handling of the tool neither does it hinder the line of view of the operator (b) **Bending suction cups**: two icosahedron shaped markers are attached to each side of an industrial tool bearing suction cups. This is intended to be used in bending operations. Since the metal sheets may easily cause occlusion of one marker, two are used to enforce a continuous tracking no matter the handling pose.

for instance), cabling to deliver the triggers to both cameras and cabling the marker to deliver both the control signal and power; inside the marker itself, there are small PCBs that hold the LEDs and discrete electronics for a total of 20 replicas that make all the lights around the icosahedron.

Also, due to the very short actuation period, the diodes are driven with well over their rated current. This produces the required powerful luminous flash, that can be captured by the camera regardless of the almost completely closed aperture of the lenses. This avoids the use of expensive ultra bright/power LEDs since cheaper high brightness ones fit the requirements using the short over current burst. Moreover, the smaller power ratings smaller semiconductors can be used, keeping the whole marker also small.

#### 4.3.1.2 Detection

The starting point for the tracking system is a pair of synchronized images. The *sincrovision* technique (recall section 3.3.4) provides very clean shots of the marker. Fig. 4.7 shows a real scene captured by a hand held camera and the two synchronous frames from the stereo pair.

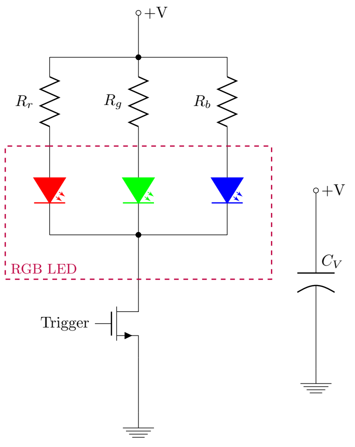
#### 4.3.1.3 *Per*-Image Analisys

The first stage of detection is based on a global threshold that takes in concern the characteristics of each LED for a fast and optimized algorithm. This is done over RGB images where each pixel $\mathbf{p}$ with image coordinates $(u, v)$ is composed by red, green and blue lights:

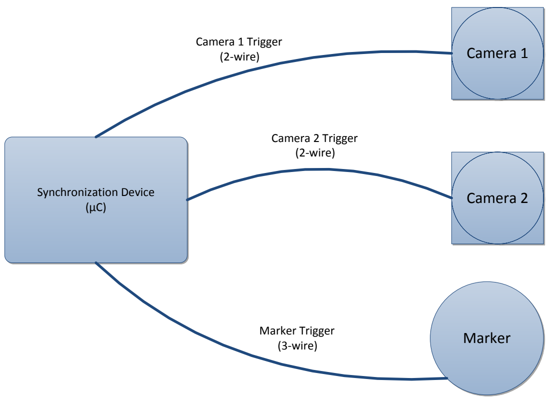
$$F(u, v) = \left[ \begin{array}{ccc} p.red & p.green & p.blue \end{array} \right] \tag{4.2}$$

To extract luminous clusters from the dark background despite the colour, the RGB model is not the most adequate so the evaluation is done over the HSV/HSL (hue, saturation, value/lightness) model. This is a cylindrical representation of the colour space (Fig. 4.8 (a))

whose hue component will come very handy in the cluster colour analysis further ahead; for now, the interest is on the *value/ lightness* component — also called *bright-*

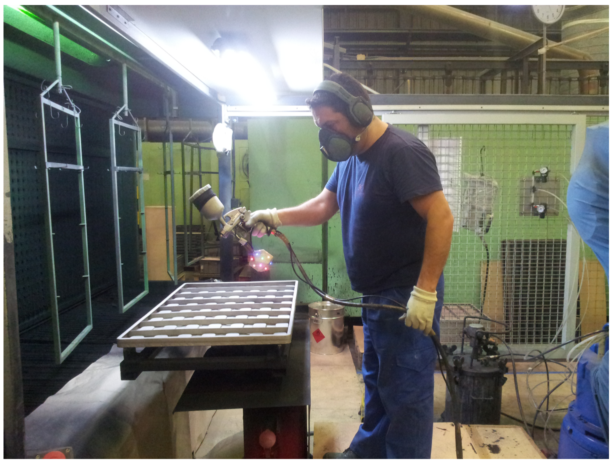


**(a)**

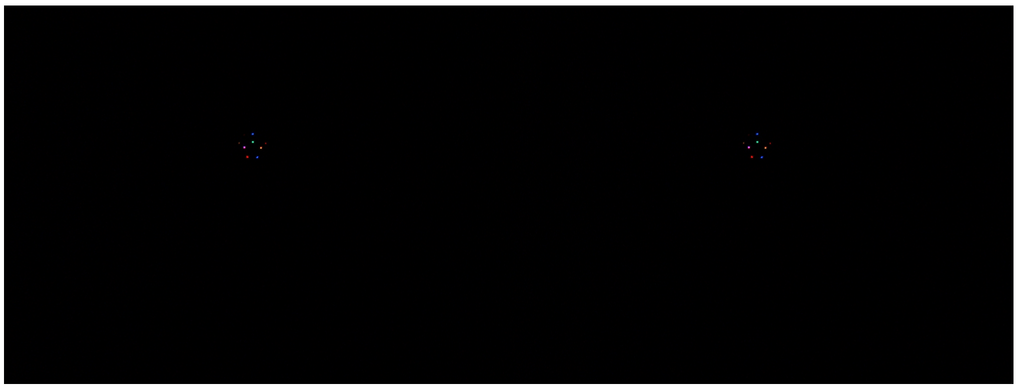


**(b)**

**Figure 4.6:** (a) **Schematic and Components**: the LED drivers for the synchronous operation are based on the most simple electronics; each colour has a series resistor in order to optimize each component individually. The mosfet does the triggering. (b): **Connections and Cabling Diagram**: the required connections and cabling are minimal. Each camera receives a pair with trigger+reference; the marker, in addiction to those, also receives power. The complete apparatus is minimal.

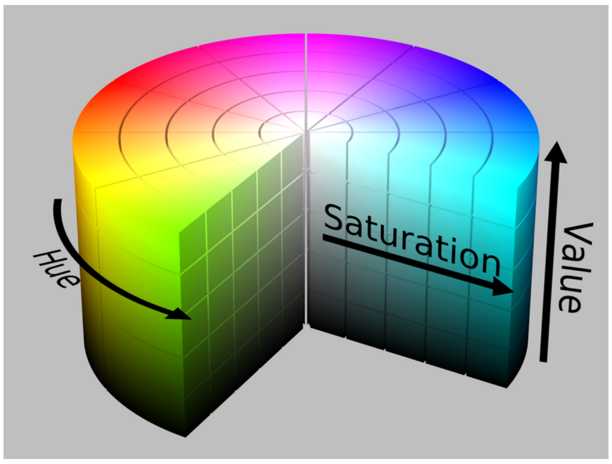


(a)

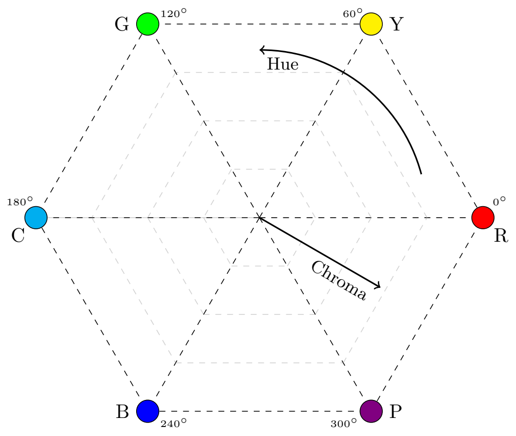


(b) (c)

**Figure 4.7:** (a) **Scene captured by a hand-held camera**: a normal photo of an operator holding the worktool with the luminous marker. (b,c): **Stereo *sincrovision* Capture**: the same scene of (a) captured by a pair of industrial cameras with the synchronization effect.



**(a)**



**(b)**

**Figure 4.8:** (a) **HSV model**: Cylindrical geometry of the colour space. The Hue component is the angular dimension that describes the colour itself; for a given colour, the value/lightness tells how close it is to white or black. (b) **Primary and Secondary colours in HSV space**: The hexagon representation of the HSV space gives a feel of how close the colours are to each other and how they combine. In order choose a set of different colours for the proposed marker, the hexagon scheme is used so that the colours are the most separated as possible. This improves image processing and cluster classification.

*ness* or even *intensity*, all refer to the same property even though each has a slightly different formula, which maps colour into a grey-scale representation.

The marker consists of red, green, blue, yellow and purple lights. The most simple definition of intensity (and respective threshold function) is the average of the three RGB components:

$$\mathrm{F}_{Grey}(u,v) = \frac{1}{3}\left(R+G+B\right) \quad \Rightarrow \quad \mathrm{F}(u,v) = \begin{cases} 0\ ,\ \text{if } \mathrm{F}_{Grey}(u,v) < \mathit{threshold} \\ \mathrm{F}(u,v)\ ,\ \text{otherwise} \end{cases} \tag{4.3}$$

but this is a poor evaluation when expecting only primary and secondary colours. Each LED light is composed by only one or two of the components; averaging the three components every time makes the intensity estimation lower then the actual value as at least one of the arguments will be always close to zero. The algorithm starts by thresholding each component then computes brightness from the active ones. This is possible because there is *a priori* knowledge from the light sources: the LED drivers are designed to provide strong intensity of a given primary colour; in the case of a secondary colour, each component intensity is reduced such that the sum of the two intensities is similar to the primary colour alone. In addiction, this has the effect that the size of all clusters in the image is fairly the same.

From the thresholded image follows the clustering algorithm. It is based on horizontal sweeps of the image frame: first, horizontal segments are detected; then they are merged together with segments from the rows below according to their relative positioning. Both large and small clusters (compared to the expected cluster size) are disregarded. Finally, the centre of mass of each light is computed.

The threshold and clustering stages are a single shot algorithm, meaning that only one entire image sweep is required. This reduces processing time keeping the whole tracking real-time capable.

After the cluster detecting comes the colour classification. Again, the HSV model is used. For reasons made clear further ahead, a total of five different colours are needed for a perfect unambiguous identification and tracking of the icosahedron.

The colour information is kept on the hue component of the HSV system which is an angular dimension. For an easy classification and distinction of the required five colours, these are chosen having the most hue distance from each other. Red, yellow, green, cyan, blue and purple and separated by at least 60° from one another — Fig. 4.8 (b). After some tests, cyan was dropped as it was harder to get using the RGB LED; the green/cyan and cyan/blue pairs caused ambiguous detections so the five icosahedron colours were finally set to red, yellow, green, blue and purple.

The HSV is then divided in 6 classification areas as seen in Fig. 4.9.

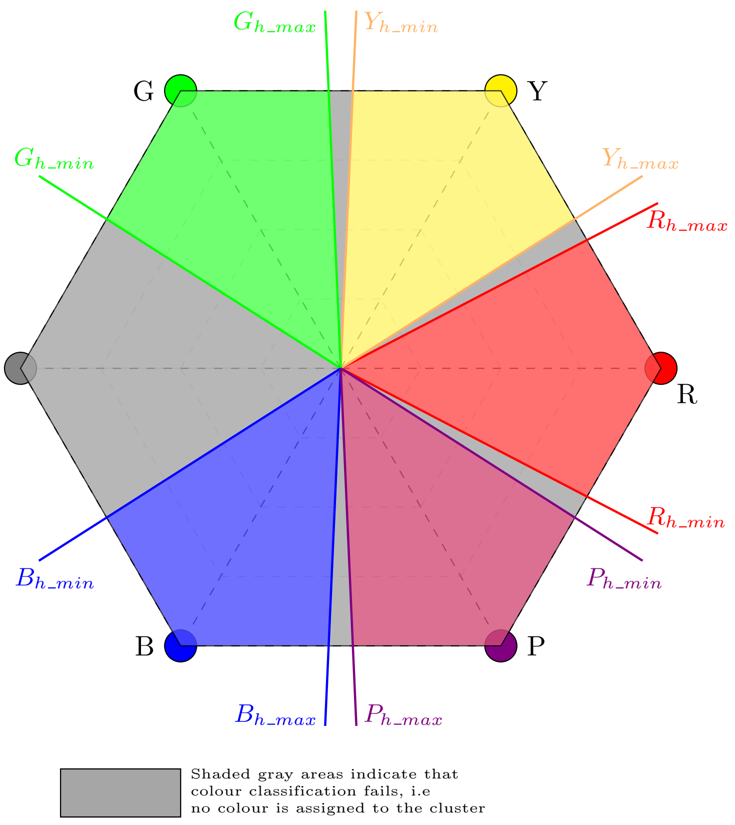


**Figure 4.9: Colour classification based on HSV** - The hexagon of HSV colour model is divided into 6 classification zones: one for each colour plus the zones of indetermination(gray).

For each cluster in the image, the hue quantity was computed as an average of each pixel hue, H, defined as:

$$\begin{aligned} \alpha &= \tfrac{1}{2}(2R - G - B) \\ \beta &= \tfrac{\sqrt{3}}{2}(G - B) \\ H &= atan2(\beta, \alpha) \end{aligned} \tag{4.4}$$

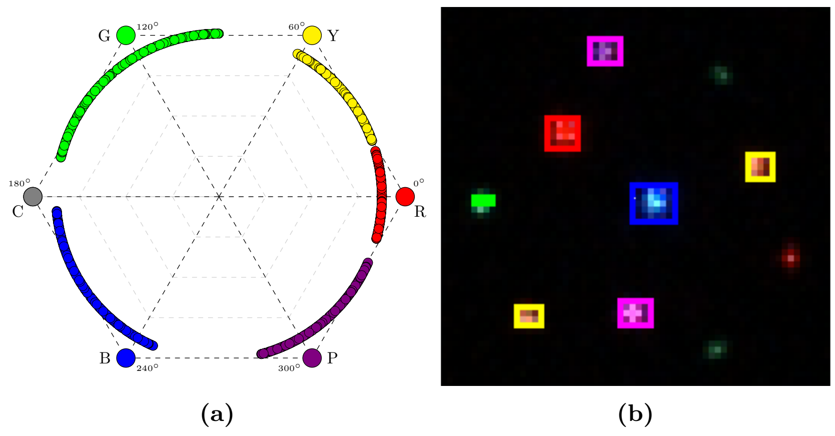


**Figure 4.10:** (a) **HSV Calibration**: Example of calibration of the colour classification algorithm. Samples are collected for each colour and the boundaries are set accordingly. This is a per-camera analysis. Remark: only the hue quantity is evaluated so all the samples are drawn in the maximum-chroma circle. (b) **Cluster Detection**: a zoomed image captured by one camera of a stereo configuration. The LEDs are detected and identified according to the average hue value of all pixels.

Equation 4.4 is a simplified and hence faster computation of hue, ignoring the peculiarities of the hexagon *vs* circle shaped HSL model (RGB cube projects into an hexagon HSV space; the formula assumes a circular space instead; the error of this approximation is smaller than 1.2° therefore negligible). Since there is a thick margin to the hue value of each colour, classification is robust to changes in the LEDs output, or even distortions from light sources pointed towards the marker. Fig. 4.9 shows a theoretical/perfect division of HSV space. In practice, the classification areas are not pre-defined but result from a colour calibration process: The colours are shown, each at a time, to both cameras. The samples are collected from the whole workspace. The boundaries between classes (hue regions) are then set individually and are different for each camera — Fig. 4.10(a) shows an example of the calibration of HSV colour classes for cluster identification; several samples are read from each colour and the limits to the hue values are set. On (b), the clusters are outlined in a real image.

#### 4.3.1.4 Stereo — cluster matching

Following the detection of individual clusters on each image, the next stage is matching them over a set of synchronized images from different cameras. For the sake of simplic-

**Table 4.1: Cluster data after processing images from both cameras**: The data is about the colour and the centre of mass of each visible LED. The stereo will have to pick clusters from both lists and match them.

Clusters

| Camera 1 | | | Camera 2 | | |
|---|---|---|---|---|---|
| Colour | Position u | Position v | Colour | Position u | Position v |
| Red | 448 | 462 | Red | 321 | 422 |
| Green | 430 | 464 | Green | 301 | 425 |
| | 475 | 463 | | | |
| Blue | 455 | 481 | Blue | 330 | 441 |
| Yellow | 474 | 481 | Yellow | 299 | 446 |
| | 428 | 484 | | | |
| Purple | 460 | 451 | Purple | 329 | 411 |
| | 444 | 494 | | 316 | 455 |

ity, the fundamentals of this step are presented for a two-camera framework; N-camera setups are processed in equal terms, considering two cameras at a time.

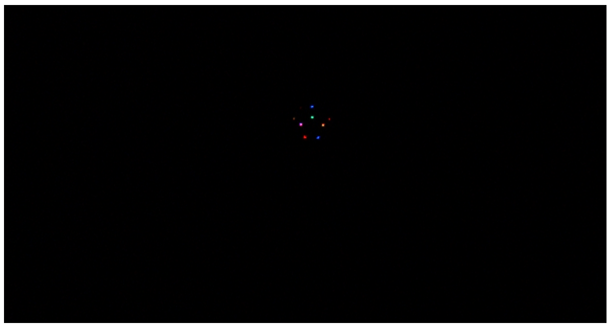
The starting point is a colour-ordered list of clusters from each image — an example is shown in Table 4.1.

The *position* entry refers to the centre of mass of the cluster. The stereo fundamental matrix, the one that defines the relationship of the two cameras to the 3D world, plays now its role: the relationship

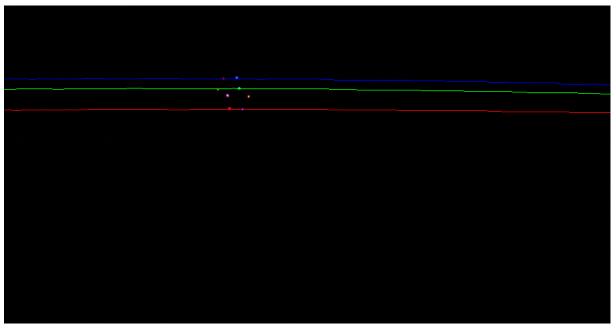
$$l' = \mathbf{Fp}$$

defines the epipolar line; for a given pixel $\mathbf{p}$ in camera 1, the epipolar line on camera's 2 image holds the possible corresponding pixels, $\mathbf{p}'$. Figure 4.11(c-right) shows three epipolar lines for a red, green and blue led on (c-left); with a precise calibration of the stereo setup, the epipolar lines go through the matching pixel on the other image. Image (a) and (b) are fullsize examples of two camera view.

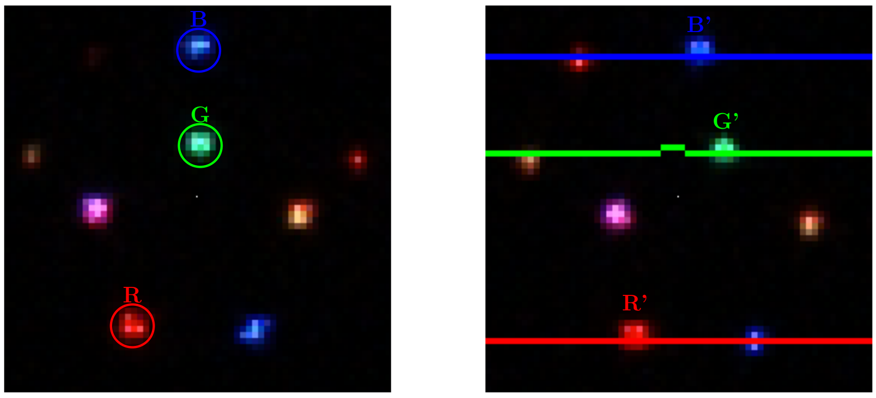
The matching algorithm picks every cluster (centre) on camera 1 and computes the



(a)



(b)



(c)

**Figure 4.11:** (a) **Camera 1 Image with clusters**: an image from camera 1 with a set of clusters. (b) **Camera 2 Image with Epipolar Lines**: the red, green and blue lines (not straight due to barrel distortion) are epipolar lines from some clusters in image (a). (c) **Zoom on Epipolar Lines and Cluster Correspondence**:the highlighted and labelled red, green and blue clusters — on the left, zoomed from (a) — need a matching pair in the other camera image so that $2D \leftrightarrow 3D$ is possible. Finding the correspondence is achieve by colour and the geometric constraint given by the epipolar lines — on the right, zoomed from (b).

corresponding epipolar, $\mathbf{e}'$; then, if any cluster of the same colour, on camera 2, lies at a parametrized distance to the epipolar, it is considered a good match.

It is possible, for some specific icosahedron poses, that two clusters of the same colour fall close to the epipolar:

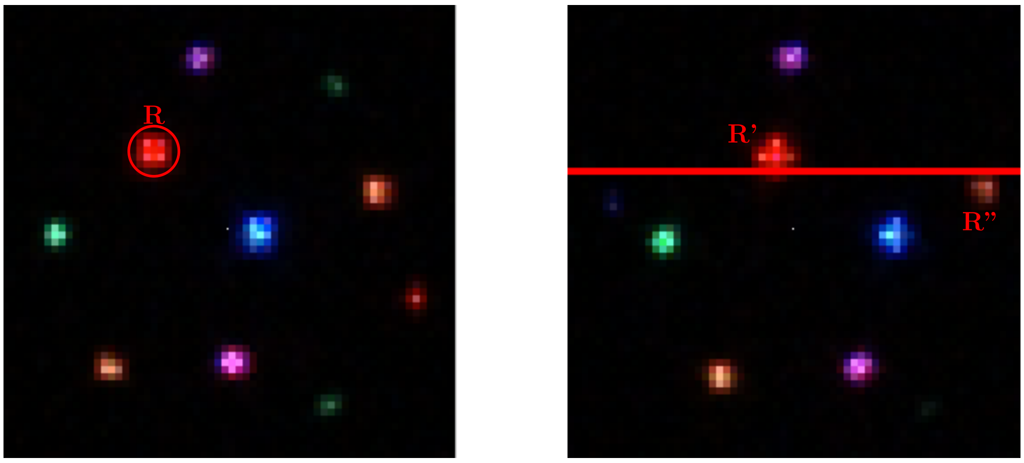


**Figure 4.12: Multiple matches** - In some cases, both the colour and the epipolar constraints fail to deliver an unique match of cluster pairs. In this example, the epipolar line of the red cluster on the left goes near two red clusters in the right image. Both hypothesis are considered.

In this case, more than one match is registered for a given cluster. At this point, it is not possible to reject either in favour of the other so both hypothesis are considered, i.e, $(R, R')$ and $(R, R'')$. On the next section, while estimating the marker position, these ambiguities are solved.

Still on Fig. 4.12, the $R''$ cluster is actually a yellow one. In such a narrow point of view, the HSV analysis misclassified it as being red. Even with the epipolar uncertainty and the misclassification, the pose estimation algorithms succeeds and proves robust to worst case scenarios of the marker detection.

#### 4.3.1.5 Estimating Position

The 3-dof related to the marker translation are computed taking advantage of the spherical positioning of the LEDs. Since the 3D coordinates of the matched clusters are already available, the world points are used to estimate the sphere shell in which they lie — Fig. 4.13 shows a 3D scene with the luminous markers and the sphere shell in which they lie at.

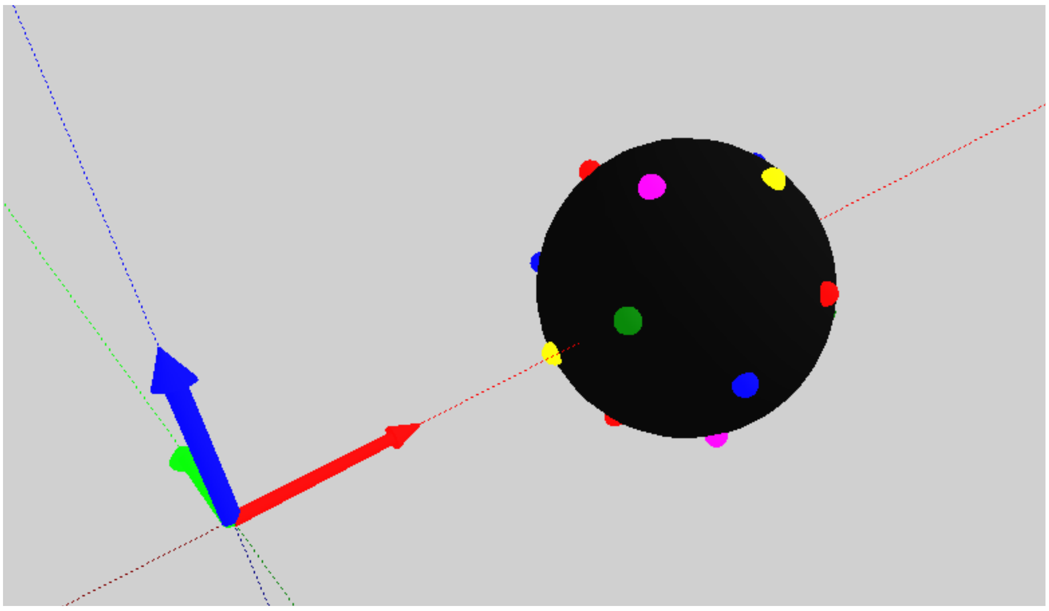


**Figure 4.13: 3D Scene with LEDs and a Sphere** - To estimate the whole marker position each individual LEDs contributes for a sphere fitting algorithm. The resulting sphere centre is used as the marker position in 3D space. The more LEDs are visible the more robust the estimation gets.

There are two main methods of data fitting to sphere surfaces, namely the geometric [66] and the algebraic fitting [67].

The former uses the most ”obvious” cost function; from the sphere equation 4.5

$$(x_i - x_c)^2 + (y_i - y_c)^2 + (z_i - z_c)^2 = r^2 \tag{4.5}$$

it minimizes the orthogonal distance, $d_i$, from each point to the sphere shell 4.6.

$$F(x_c, y_c, z_c, R) = \sum_i d_i^2 \tag{4.6}$$

The vector $[x_c, y_c, z_c]^T$ denotes the centre of the sphere and $R$ its radius. Expanding 4.6 since

$$d_i = r_i - R \ , \ r_i = \sqrt{(x_i - c_x)^2 + (y_i - c_y)^2 + (z_i - c_z)^2}$$

it resolves into a non-linear least squares minimization, 4.7, which has no closed form solution.

$$\min \sum_{i=1}^{n} \left( (x_i - x_c)^2 + (y_i - y_c)^2 + (z_i - z_c)^2 - r^2 \right)^2 \tag{4.7}$$

Methods for solving the geometric fitting all consist in iterative algorithms, with Levenberg-Marquardt optimization being one of the most common solutions. Since the goal is to keep processing time to a minimum, this solution has been avoided; iterative algorithms may take large amounts of time to converge which is also dependent of the initial estimate. Failing to converge would also compromise the output of the tracking system. Furthermore, the position estimation will make use of the sphere fitting at least once per observation. It may be called upon several times to eliminate stereo ambiguities, as explained ahead.

For these reasons, the algebraic approach is preferred. Starting over from the sphere equation 4.5, the expansion of the squared terms retrieves:

$$-2x_i x_c - 2y_i y_c - 2z_i z_c + x_c^2 + y_c^2 + z_c^2 - r^2 + x_i^2 + y_i^2 + z_i^2 = 0$$

With variable substitution

$$\kappa = {x_c}^2 + {y_c}^2 + {z_c}^2 - r^2$$

and for $n$ sample points, it results in (matrix notation):

$$\underbrace{\begin{bmatrix} 1 & 2x_1 & 2y_1 & 2z_1 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 2x_n & 2y_n & 2z_n \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} -\kappa \\ x_c \\ y_c \\ z_c \end{bmatrix}}_{\theta} = \underbrace{\begin{bmatrix} x_1^2 + y_1^2 + z_1^2 \\ \vdots \\ x_n^2 + y_n^2 + z_n^2 \end{bmatrix}}_{\mathbf{b}} \tag{4.8}$$

Equation 4.8 now solves directly using linear least squares — Eq. 4.9.

$$\hat{\theta} = \left( \mathbf{A^T A} \right)^{-1} \mathbf{A^T b} \tag{4.9}$$

This method, with a closed form solution with basic matrix algebra, suits a real time implementation of the movement tracking.

As for the position estimation itself, the collected data from the 3D analysis of the stereo pairs is organized in a list as depicted on Table 4.2 (a).

**Table 4.2: Marker Position Estimation**: an example of using the sphere fitting algorithm to eradicate bad matches. At each step, a different pair is tested; at the end, all possible combinations are used and each outputs an estimation for the sphere radius. The known true value is 40mm so the 3rd. step used all the good matches. The other possibilities are eliminated.

**(a)** Sphere Fitting Step 1

| $a$ | $b$ |
|---|---|
| $(R_1, R_1')$ | - |
| $(R_2, R_2')$ | - |
| $(G_1, G_1')$ | $(G_1, G_1'')$ |
| $(B_1, B_1')$ | - |
| $(Y_1, Y_1')$ | $(Y_1, Y_1'')$ |
| $(P_1, P_1')$ | - |
| $(P_2, P_2')$ | - |

**(b)** Sphere Fitting Step 2

| $a$ | $b$ |
|---|---|
| $(R_1, R_1')$ | - |
| $(R_2, R_2')$ | - |
| $(G_1, G_1')$ | $(G_1, G_1'')$ |
| $(B_1, B_1')$ | - |
| $(Y_1, Y_1')$ | $(Y_1, Y_1'')$ |
| $(P_1, P_1')$ | - |
| $(P_2, P_2')$ | - |

**(c)** Sphere Fitting Step 3

| $a$ | $b$ |
|---|---|
| $(R_1, R_1')$ | - |
| $(R_2, R_2')$ | - |
| $(G_1, G_1')$ | $(G_1, G_1'')$ |
| $(B_1, B_1')$ | - |
| $(Y_1, Y_1')$ | $(Y_1, Y_1'')$ |
| $(P_1, P_1')$ | - |
| $(P_2, P_2')$ | - |

**(d)** Sphere Fitting Step 4

| $a$ | $b$ |
|---|---|
| $(R_1, R_1')$ | - |
| $(R_2, R_2')$ | - |
| $(G_1, G_1')$ | $(G_1, G_1'')$ |
| $(B_1, B_1')$ | - |
| $(Y_1, Y_1')$ | $(Y_1, Y_1'')$ |
| $(P_1, P_1')$ | - |
| $(P_2, P_2')$ | - |

**(e)** Sphere Fitting Result

| Step No. | Sphere Radius $mm$ |
|---|---|
| 1 | 74.36 |
| 2 | 71.98 |
| 3 | 39.60 |
| 4 | 78.55 |

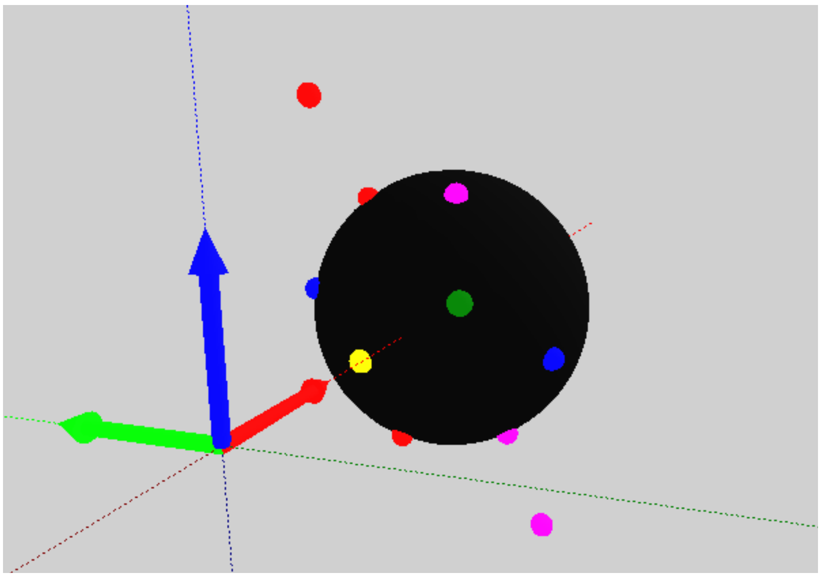
It's a list containing all matched clusters, their respective colours and the ambiguous pairs.

The fast computational nature of the sphere fitting allows, at this point, to investigate and eradicate the matching ambiguities. The algorithm will match every possible combination of the paired clusters. Then, even though the sphere radius is known *a priori* (a constructive measure), its estimation from the sphere fitting algorithm sheds light on the quality of the clusters: if the estimated radius largely deviates from the known measure, it indicates that the 3D point set is ill; one or more points do not integrate the expected sphere shell — Fig. 4.14(a) shows a good fitting whilst on (b) the two outliers ruin the estimation; the radius is larger and indicates that some clusters were wrongly paired.

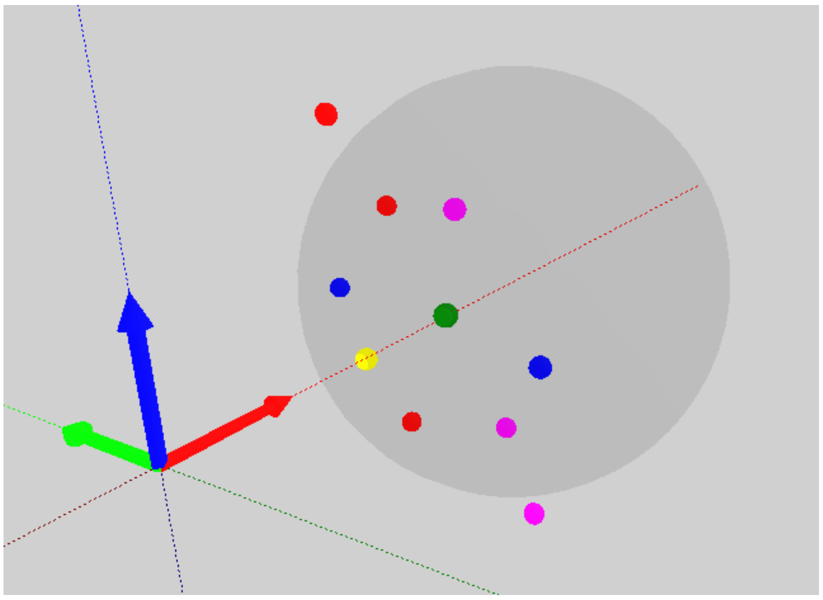
The algorithm then moves on to a different combination, using other matching candidate: this steps are shown on Table 4.2. If no configuration is considered good, then one 3D point is dropped and the previous search is done all over again. With, at most, twenty visible LEDs and two ambiguities (constructively it is not likely that more than 2 LEDs are aligned on the same epipolar), this iterative and recursive fitting takes no longer then a few milliseconds; typically, it runs below the millisecond mark.

Remark 1: in an offline implementation, or with supervised operation, the geometric fitting can be used as a final step to further enhance precision. After the correct set of matched clusters is found, the iterative optimization can be performed using the algebraic fitting estimation as a good initial guess. Despite this, it is not guaranteed that the non-linear method will converge.

As presented by Sung *et. al* [68], the orthogonal method can provide better estimates for increasingly noisy data. For the most basic tracking framework, with a minimum of two cameras, it is expected that the visible LEDs lie on the same side of the sphere. This tends to increase the bias of the centre estimation and the geometric fitting can prove a valuable tool in this scenario. Though, with a more complete setup (that may guarantee LED visibility all around the marker) the algebraic fitting do not fall short to the other method; the error improvement of the geometric fitting can be neglected in this case.



(a)



(b)

**Figure 4.14:** (a) **Good Fitting**: a sphere fits to the LEDs. There are two outliers, one red above the sphere and a purple below it. This is the correct output, ignoring the outliers. (b) **The Sphere Fitting Output Using all available LEDs**: if one or mode LEDs are take into consideration because a bad match occurred in the stereo stage, the sphere fitting retrieves an oversized or undersized sphere. Since the true radius is known, if the estimation is different than there are bad paired clusters that must be eliminated. The algorithm gradually eliminates outliers until a good fitting is achieved.

Remark 2: With proper placement of the minimum of two cameras, the marker guarantees, constructively, that at least 4 LEDs are visible are all the times. Also, these 4 lights are always non coplanar, which avoids the singularity (3 points alone are coplanar) of the fitting algorithm. Obviously, if due to unexpected behaviour less LEDs are visible then the position estimation is skipped; the pose corresponding to that particular time slot is filled by averaging its neighbours as described in the data filtering section further ahead.

#### 4.3.1.6 Estimating Orientation

After retrieving the translation vector, the missing 3-dof with respect to angular displacement are computed from the known well-matched 3D points.

Using again the advantages of the dodecahedron geometry, it is possible to build a list of the LEDs' coordinates in the 3D world. It is important to note that the actual position of the polyhedron where these coordinates are read does not need to be known. It is sufficient to know that there exists one dodecahedron pose to which a set of LED coordinates maps. For instance, there is a coordinate frame (or orientation) at which the dodecahedron vertices lie at:

$$\begin{aligned} &s\,(\pm 1, \pm 1, \pm 1)\\ &s\,(0, \pm 1/\varphi, \pm\varphi)\\ &s\,(\pm 1/\varphi, \pm\varphi, 0)\\ &s\,(\pm\varphi, 0, \pm 1/\varphi) \end{aligned} \tag{4.10}$$

where $s$ is a scale factor and $\varphi = \left(1+\sqrt{5}\right)\big/2$ is the so called golden ratio (if $a$ and $b$ are in golden ration, and $a > b$, then $\varphi = \frac{a+b}{a} = \frac{a}{b}$).

For the rotation estimation algorithm, this set of known 3D positions is designated by $P$, where $P_i$ is a 3D vector $[x_i, y_i, z_i]^T$ with the coordinates of dodecahedron vertex $i$. $P$ is considered the set of *stand-by* positions.

The rotation estimation is then achieved in two steps:

1. Finding which LED is which, according to its colour and neighbours; outputs a correspondence list between the observed position and the stand-by pair;

2. Compute the optimal rotation matrix that transforms the stand-by positions into the current ones.

Into the first step: the dodecahedron can be drawn using a schlegel diagram— recall Fig. 4.4 (c). A schlegel diagram is a planar graph that consists of a projection of $\mathbb{R}^n$ into $\mathbb{R}^{n-1}$ which, in this case, projects the 3D dodecahedron in a 2D equivalent (where lines never cross). The graph's lines and nodes represent the polyhedron vertices and edges, respectively.

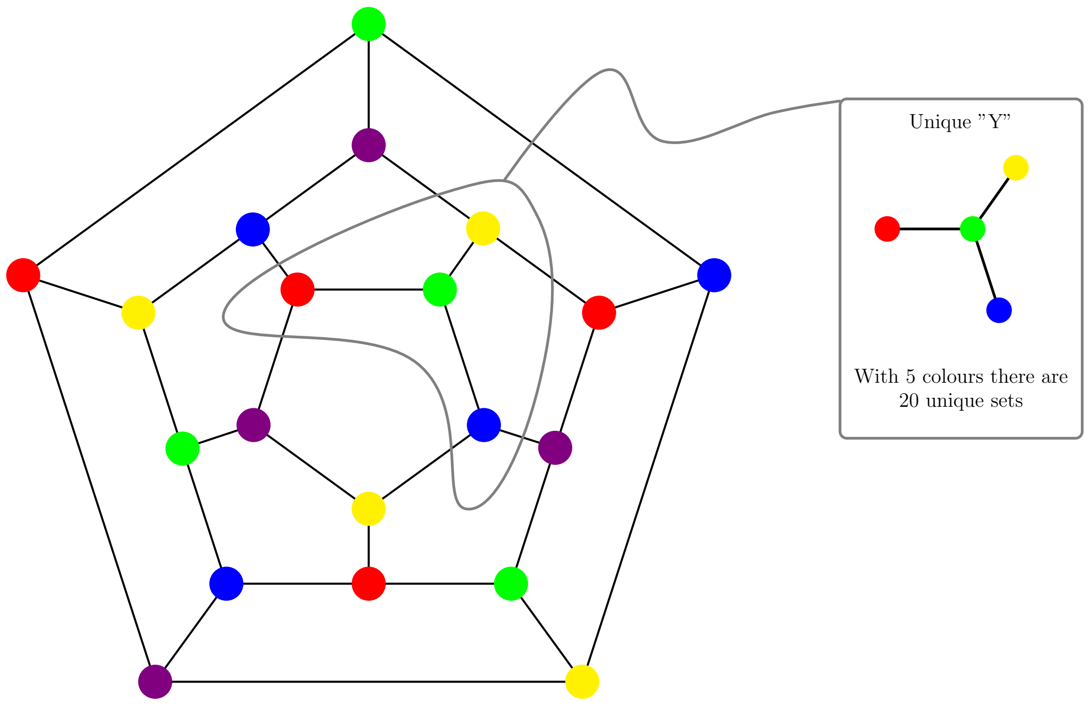
From this representation comes the interest and usefulness of having the vertices coloured by five distinct colours: there are twenty vertices with three edges connected to each; If no vertex has a neighbour of the same colour, then there are exactly twenty possible arrangements of colours that take one LED plus its three neighbours (hereafter such a set is also called a *star* or **Y**) — 5 possible colours for centre $\times$ 4 possible colours that will not appear in the neighbours; i.e, every *star* features all but one colour. Using the graph, these sets can be distributed such that the orientation of the sphere is unequivocally known given one complete *star* — Fig. 4.15(a). Fig. 4.15(b) shows the detection of the *star*/**Y** in a real image. There are many possible solutions to distribute the twenty **Y**s; the chosen configuration holds the sequence Red-Green-Blue-Yellow-Purple clockwise within each pentagon but it has no influence over performance (it may be more intuitive while assembling the marker).

The stand-by position list is augmented with the neighbour information as shown in Table 4.3.

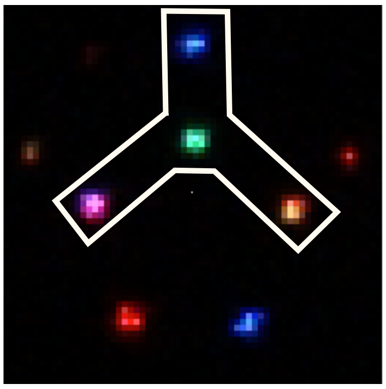
For every 3D point list from a stereo image pair, a distance matrix defined. It states whether or not a point is neighbour to the others. This step assumes that, by design, the distance between LEDs is known.

With the neighbour/distance matrix, is it possible to search for a complete *start*. The marker geometry and camera placement guarantee that such a set of four LEDs is always visible, one at least. If that *start* can be found, that it can be compared to the stand-by list. Once that vertex and the three neighbours are known (the correspondence to the original position), a recursive analysis of all the other visible lights in the image retrieves the complete point correspondence list. On one hand there are the stand-by positions and, from the observations, there are a set, $Q$, of measured positions. It is important to note that this comparison can only take place after removing the matching ambiguities on the previous step.

The second part of the orientation estimation algorithm lies on the computation of the optimal rotation matrix between the two sets, $P$ and $Q$, of paired 3D points — the



**(a)**



**(b)**

**Figure 4.15:** (a) **Coloured Dodecahedron Graph**:The schlgel diagram provides an intuitive way to view the distribution of LEDs around the dodecahedron. Using 5 colours, it is possible to colour the vertices with a pattern that allows a quick identification of the marker orientation. As soon as a *star*/**Y** is completely visible in the images, orientation is known. (b) **Y Detection**: Detection of a complete *star*/**Y** in a real image capturing the luminous marker. The **Y** is identified and it is enough to know the full orientation of the marker in world coordinates

**Table 4.3: Dodecahedron Vertices**: Coordinates, Colour and Neighbours

| Dodecahedron Vertices | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| No. | Colour* | Coordinates | | | Missing | Neighbours No. | | |
| | | x | y | z | Colour | Ngb 1 | Ngb 2 | Ngb 3 |
| 1 | 1 | -0.38197 | 0.525731 | 0.850651 | 4 | 2 | 5 | 8 |
| 2 | 2 | 0.381966 | 0.525731 | 0.850651 | 5 | 1 | 3 | 9 |
| 3 | 3 | 0.618034 | -0.20081 | 0.850651 | 1 | 2 | 4 | 10 |
| 4 | 4 | 3.28E-08 | -0.64984 | 0.850651 | 2 | 3 | 5 | 6 |
| 5 | 5 | -0.61803 | -0.20081 | 0.850651 | 3 | 1 | 4 | 7 |
| 6 | 1 | -2.29E-08 | -1.05146 | 0.200811 | 5 | 4 | 12 | 13 |
| 7 | 2 | -1 | -0.32492 | 0.200811 | 1 | 5 | 13 | 14 |
| 8 | 3 | -0.61803 | 0.850651 | 0.200812 | 2 | 1 | 14 | 15 |
| 9 | 4 | 0.618034 | 0.850651 | 0.200811 | 3 | 2 | 11 | 15 |
| 10 | 5 | 1 | -0.32492 | 0.200811 | 4 | 3 | 11 | 12 |
| 11 | 1 | 1 | 0.32492 | -0.20081 | 2 | 9 | 10 | 18 |
| 12 | 2 | 0.618034 | -0.85065 | -0.20081 | 3 | 6 | 10 | 19 |
| 13 | 3 | -0.61803 | -0.85065 | -0.20081 | 4 | 6 | 7 | 20 |
| 14 | 4 | -1 | 0.32492 | -0.20081 | 5 | 7 | 8 | 16 |
| 15 | 5 | 2.29E-08 | 1.051462 | -0.20081 | 1 | 8 | 9 | 17 |
| 16 | 1 | -0.61803 | 0.200812 | -0.85065 | 3 | 14 | 17 | 20 |
| 17 | 2 | -3.28E-08 | 0.649839 | -0.85065 | 4 | 15 | 16 | 18 |
| 18 | 3 | 0.618034 | 0.200812 | -0.85065 | 5 | 11 | 17 | 19 |
| 19 | 4 | 0.381966 | -0.52573 | -0.85065 | 1 | 12 | 18 | 20 |
| 20 | 5 | -0.38197 | -0.52573 | -0.85065 | 2 | 13 | 16 | 19 |

*Colours: 1-Red; 2-Green; 3-Blue; 4-Yellow; 5-Purple *nth* vector in $Q$ stores the current world position of the *nth* stand-by LED in $P$.

The kabsch algorithm [69] provides a mean to solve this problem. This method finds the rotation matrix that optimally describes (in a root-mean-squared-error sense) the rotation from two paired 3D point lists:

1. $P$ and $Q$ must have origin-centred vectors so the first step is subtracting both sets their respective centroid.

2. Compute the covariance matrix $A$ defined as: $A = P^T Q$

3. Compute the singular value decomposition of $A$:

$$A = USV^T$$

and the optimal rotation matrix $W^*$ comes from :

$$d = sign(det(VU^T)) \quad \rightarrow \quad W^* = V \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & d \end{pmatrix} U^T \tag{4.11}$$

(The auxiliary parameter $d$ is used to insure a right handed coordinated system.)

At this point we have the full (6-dof) characterization of the movement of the marker, with both position (3-dof) and orientation (3-dof).

#### 4.3.1.7 Limitations

The icosahedron shaped marker provides the most complete movement tracking, with full 6-DoF capture of the human operator. Most of the properties of this shape have proved very useful and effective regarding handling and algorithms. Even so, there are some known limitations:

- The smaller and most common hi-power LEDs, that can be driven bright enough to get captured using *sincrovision*, are around 3 to 5 mm in size — side of a square shaped LED. This restricts the minimum size of the marker (radius) at about 30mm. This restriction is required in order to prevent light superposition — in an undersized marker the lights get mixed up.

- The sphere fitting algorithm behaves well with observations from all around the full sphere shell. This means that cameras should be placed on two different/opposing sides for optimal performance. This way, the collected 3D points are balanced throughout the spherical surface. A single pair of cameras provides measures from just one side, which causes individual LED noise to have a greater effect on the centre estimation. This happens because the measure along the depth axis of the stereo is where the error is larger so the centre estimation tends to have greater variance and bias in that direction.

- The most cheap arrangement, with the minimum of two cameras, has the combined disadvantages of the previous two statements. On one hand it greatly influences the size of the marker: even though a proper choosing of lenses may provide a complete workspace coverage, the stereo precision is limited to how much the cameras can be placed apart; the depth precision can only be improved with more cameras rather than with better ones (more resolution). One of the must crucial and laborious tasks is to achieve a perfect trade-off between number of cameras, tracking precision and marker size that can satisfy the demanded system precision, work volume, shop-floor restrictions for camera instalment and the size/weight of the marker that will be attached to the operator's tool.

- The further the number of cameras is reduced, the more susceptible the system is to occlusions. The tracking algorithms work for a minimum of 4 visible LEDs; while the framework attempts to be the less intrusive possible in the process, the operator still needs to be cautious not to stand in-between marker and cameras.

- The synchronization process requires cabling. Even though some industrial tools are active (actively powered, electrically or by means of compressed air for instance) as in painting, welding, etc, some processes are passive (grinding, polishing, etc); in the case of active tools, the extra marker cabling may be an acceptable overhead while in passive processes it may present itself as an extra apparatus that the operator may have to learn how to cope with.

## 4.4 Interfacing an Industrial Manipulator

The tracking subsystem provides, in short, a set of time-stamped world coordinates. Due to the camera calibration process presented earlier, the coordinates are already referenced to the robot's base frame which simplifies the interface process.

Nevertheless, there is still much to an operators' task beyond the pure movement. The most critical point in order to achieve a perfectly timed mimic is the interface with peripherals. Capturing the wrist pose can be complemented with reading the actuation of the main control signals of the process such as a blowtorch on/off, the spray gun trigger, the suction cup enable/disable, etc. These main interfaces are synchronously captured and only then can the robot perform a timely imitation.

The developed synchronization device outputs the camera and marker triggers (synchronous signals) and is also able to keep the same synchronization across the peripherals in the industrial cell — Fig. 4.16. Should the operator interact with any of those commands (asynchronous signals), the action is recorded with a time stamp.

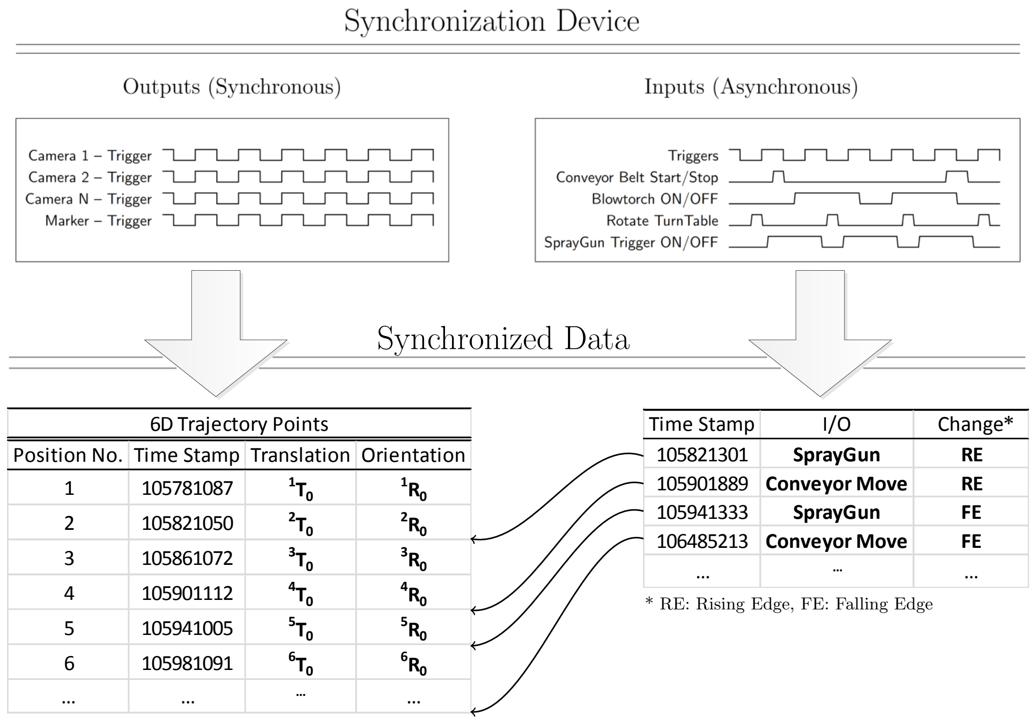
The robot interface then starts from two synchronized lists of human movements and human-device interfaces, as depicted in the bottom part of Fig. 4.16.

### 4.4.1 Data Filtering

This section deals with data smoothing, for both position and orientation. Rather than using online filters, the focus sets on using all points to create a smoother path. This means that a given 6D point $M_i$ is filtered using past and future poses $M_i^{filt} = f\left(\{M_i\}_{i-p}^{i+p}\right)$, with $p$ being the filter window width,i.e, how many past and future points are used. Position and orientation are treated separately but both approaches focuses on delivering a smooth path (either in position or orientation space).

#### 4.4.1.1 Position Smoothing

Position is described by a translation vector ${}^{k}T_0$. It represents the position of the marker at time $k$ in coordinates of the robot's base frame (because cameras where calibrated using the robot). In order to simplify notation, each translation vector will now be treated as a position $P_k$ in space. This change of notation is needed so to maintain a certain degree of standardization in the notation of the smoothing method.

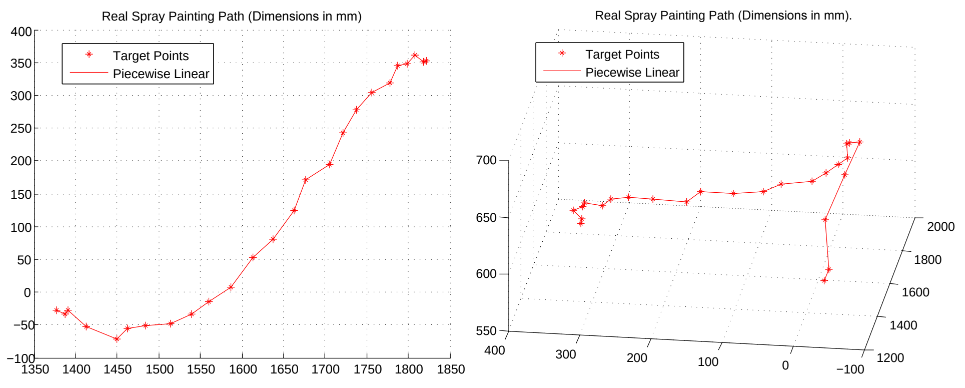


| 6D Trajectory Points | | | |
|---|---|---|---|
| Position No. | Time Stamp | Translation | Orientation |
| 1 | 105781087 | ${}^{1}\mathbf{T}_{0}$ | ${}^{1}\mathbf{R}_{0}$ |
| 2 | 105821050 | ${}^{2}\mathbf{T}_{0}$ | ${}^{2}\mathbf{R}_{0}$ |
| 3 | 105861072 | ${}^{3}\mathbf{T}_{0}$ | ${}^{3}\mathbf{R}_{0}$ |
| 4 | 105901112 | ${}^{4}\mathbf{T}_{0}$ | ${}^{4}\mathbf{R}_{0}$ |
| 5 | 105941005 | ${}^{5}\mathbf{T}_{0}$ | ${}^{5}\mathbf{R}_{0}$ |
| 6 | 105981091 | ${}^{6}\mathbf{T}_{0}$ | ${}^{6}\mathbf{R}_{0}$ |
| ... | ... | ... | ... |

| Time Stamp | I/O | Change* |
|---|---|---|
| 105821301 | **SprayGun** | **RE** |
| 105901889 | **Conveyor Move** | **RE** |
| 105941333 | **SprayGun** | **FE** |
| 106485213 | **Conveyor Move** | **FE** |
| ... | ... | ... |

* RE: Rising Edge, FE: Falling Edge

**Figure 4.16: Synchronization across images and devices -** The synchronization device delivers the triggers for the cameras and the marker. At the same time it reads the changes to I/Os. The video tracking outputs a list of 6D positions while the monitoring of the I/Os provides a synchronous list that can be merged into the trajectory using the time stamp.



**(a)** **(b)**

**Figure 4.17:** (a) **2D Upper View**, (b) **3D View**: These plots show a set of captured marker positions in a real spray painting application. The "*" represent the spatial position and the lines are but a linear piecewise approximation for the path. Despite the synchronous capture, the position estimation is still noisy.

Even though the tracking system is very robust to disturbances, the output path comes affected by noise. It has roots on the camera sensor, which captures noisy images despite the *sincrovision* effect, and on the position estimation algorithm, particularly when a small number of cameras is used and the marker is only visible from one of the sides (already discussed in section 4.3.1.5). Figure 4.17 shows an example of a real trajectory where each position is connected by a line. In order to remove some of the noise, and achieve an equivalent smoother path, a spline based smoothing is proposed.

Splines are polynomials defined piecewise, i.e, a set of $n$-degree polynomial functions connected together in a smooth fashion. The motivation for the use of splines comes from their extensive and successful use in computer graphics and CAD systems, to approximate arbitrarily complex curves and surfaces. Also, there are already numerous contributions on industrial machining problems and even industrial robotics (for instance, Jia Pan et al [70]) that take advantage of splines.

The main advantages of this smoothing/modelling tool are the numerical stability, easiness and flexibility of construction and the ability to fit complex curves (like a free hand movement). The numerical stability is closely related to the possibility of interpolating data (as in polynomial interpolation) with low-degree polynomials which avoids the Runge's phenomenon for single high-degree polynomial interpolation. For this reason, the most commonly used splines are of 3rd degree, i.e, the so called cubic splines. Moreover, there are numerous forms of splines which vary on how they are constructed or which features they offer. The most commonly used (generally, but also particularly in curve fitting) are B-splines: a spline that is itself a (linear) combination of splines and offers means to easily ensure continuity across a set of chained splines used approximate a whole curve.

This introduction leads to the definition of a B-spline as a linear combination of splines:

$$S\left(\mathbf{u}\right) = \sum_{i=0}^{n} P_i N_{i,p}\left(\mathbf{u}\right) \tag{4.12}$$

The points $P_i$, for a total of $n + 1$,

$$P = \{P_0, P_1, ..., P_n\}$$

are called the *control points*. It is to these positions that the final spline curve tries to stick to. $N_{i,p}\left(u\right)$ are the splines which are linearly combined to obtain $S$; they are called the basis functions. From the definition above, equ. 4.12, it is easily seen that the final B-Spline is not defined as a function of $(x, y, z)$ positions. Instead, $S$ stands as a parametric function with parameter $u$ taking values in the interval $[0, 1]$. Yet, $\mathbf{u}$ was represented as lower-case bold letter because it is a vector

$$\mathbf{u} = \{u_0, u_1, ..., u_m\}$$

where each $u_j$ is a value in $[0, 1]$ representing break points on the curve, that is, the whole curve is segmented in smaller curves. Each of these is itself a linear combination of splines. So it is easy to understand that the $u_j$ sequence must be crescent (moving along de curve, from 0 to 1, and breaking it along the way):

$$u_j \leqslant u_{j+1} \quad , \quad 0 < j < m - 1$$

Vector $\mathbf{u}$ is known as the *knot* vector.

The last component that makes $S$ are the basis functions $N_{i,p}\left(u\right)$. $p$ is the degree of the basis functions and, as such, is the degree of the B-Spline because the latter is a linear combination of the former.

According to Krebs, H. et al. [71], a human-like movement can be well described by a combination of bell shaped basis functions. This is the shape of a cubic basis function — see Figure 4.18. For this reason, cubic B-splines are used in the smoothing algorithm.

The zero-order basis function (constant), $N_{i,0}\left(u\right)$, is defined as — from [72]:

$$N_{i,0}\left(u\right) = \begin{cases} 1 & \text{if } u_i \leqslant u \leqslant u_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{4.13}$$

The next higher order basis functions are recursively computed from $N_{i,0}\left(u\right)$:

$$N_{i,p}\left(u\right) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}\left(u\right) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}\left(u\right) \tag{4.14}$$

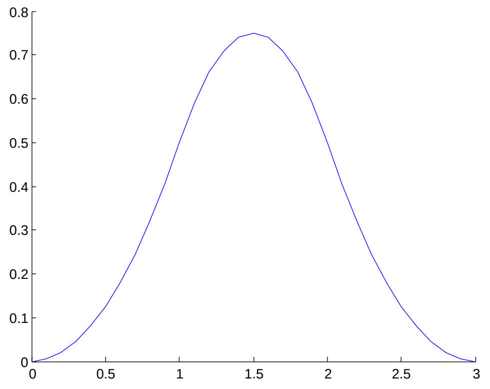


**Figure 4.18: B-Spline Cubic Basis Function** - Illustration of the cubic basis function for a 3rd degree B-Spline. It has the so called bell shape.

At this point, where a path with $P_i$ points awaits for smoothing, the only thing that is not already available is the *knot* vector. As pointed before, this is where the entire curve gets broken into smaller curves. As such, the *knot* positions have a major influence on the final layout of $S$. First, the number $m$ of *knots* must respect the relationship: $m = n + p + 1$, i.e, the number of points, $n + 1$, plus the degree of the polynomial. To define values for $\mathbf{u}$ there are different strategies (covered in [73] ); in short, the sequence may be uniform, $\mathbf{u} = \left[0, 1/\left(m-1\right), 2/\left(m-1\right), ..., 1\right]$, or non-uniform (which result in the so called uniform or non-uniform B-Splines, respectively); additionally, the sequence can be closed, clamped or open which result, respectively, in: a closed curve where the last point is the same as the starting one (not useful in smoothing an unpredictable human move); a curve clamped to the first and to the end points, smoothing all positions in-between; an open curve which has no restrictions. In this case the clamped method is used as it is desirable that the robot playback starts and ends at the same place as the human did. In addiction, a non-uniform sequence must be used because data is not equally spaced (it is in time dimension but not in spatial dimensions). The method for clamping both ends is to set the first and last values of **u** to 0 and 1 respectively. The number of 0s and 1s is related to the degree of the polynomial — $p+1$ repetitions at each end:

$$\mathbf{u} = [u_0, u_1, \ldots, u_p, \ldots, u_{m-p}, u_{m-p+1}, \ldots, u_m] = \left[\underbrace{0, 0, \ldots, 0}_{p+1}, \ldots, \underbrace{1, 1, \ldots, 1}_{p+1}\right] \tag{4.15}$$

To fill in the middle of the *knot* vector the method of *chord length* was used: each point $P_i$ is given a parameter $t$ (actually this is still a $u$ value but it is named $t$ not to be confused with the $u$ values that belong to the *knot* vector). This parameter $t$ is proportional to the length of the lines connecting the control points $P_i$; that is, $t$ parameters are as spaced as points are:

$$\begin{aligned} d &= \sum_{k=1}^{n} \|P_i - P_{i-1}\|_2 \\ t_i &= \frac{\sum_{j=1}^{i} \|P_j - P_{j-1}\|_2}{d} \end{aligned} \tag{4.16}$$

Then, the $u$ values that compose the *knot* vector are defined as:

$$\begin{aligned} &u_0, \ldots, u_p = 0 \\ &u_{p+1}, \ldots, u_{m-p} = u_{h+p} = \frac{1}{p} \sum_{k=h}^{h+p+1} t_k \\ &u_{m-p}, \ldots, u_m = 1 \end{aligned} \tag{4.17}$$

where the middle values, $u_{h+p}$, are generated from the average of $p$ parameters.

With this formulation, the B-Spline is completely defined. Points $P_i$ from the path serve as control points, and the ratio of their relative distance to the whole path gives rise to the *knot* vector. From the definition of $S$, recovering points along the curve (also called *evaluating* the spline) is done by giving values to $u$, from 0 to 1. This step shows the major disadvantage of spline fitting: as discussed above, B-Splines are defined in a parameter space of $u$ and not in actual 3D space; this means that it is quite hard to know which $u$ to use in order to go to some well-defined 3D point. In other words, there is no direct correspondence from an original point path $P_i$ to a $u$ value; there is no saying that $P_i$ corresponds to the new filtered $S(u_i)$.

Another subject that influences the outcome of the B-spline smoothing is that using all the trajectory points also causes the B-Spline to follow the noisy data to closely; furthermore, too many points greatly increases the computation time. To address this issue, the implemented solution samples the path $\{P_i\}$ so that less points are used to compute the B-Spline. After inspection of a number of runs, and trying different sampling rates, it can be concluded that choosing every other point or every three points has acceptable results. Acceptable in the sense that the curve does not over-smooth the path, does not short-cut corners, while eliminates much of the high frequency noise. These tests shall be presented later on section 5. Nonetheless, this choosing has much to do with the characteristics of the demonstration: the sampling can be adjusted as a function of the camera frame rate and the velocity of the demonstration; for instance, slow movements may require less points whilst highly dynamic ones, with sharp turns and edges, may be more demanding. The critical situations are U-turns as it is important that the sampling does not miss narrow arcs. This way the spline does not over smooth the edges and sharp turns. In short, from the set of original tracked points

$$\{O\}_0^n = \{P_0, P_1, \ldots, P_n\}$$

the spline is defined with a set $\{D\}_0^k$ sampled from $\{O\}_0^n$ at a frequency $s$, such that

$$\{D\}_0^k = \{D_0, D_1, \ldots, D_k\} = \{P_0, P_s, P_{2s}, P_{3s}, \ldots\}$$

At this stage, evaluating the B-Spline, $S$, at some $u$ values gives a smooth path from start to end, $S(u)$. The problem of choosing which "$u$"s to evaluate is again solved using the *chord length* method for every point in the original trajectory. This retrieves a vector $\mathbf{u_{eval}}$ (which is equal to the set of $t$ parameters defined above in Equation 4.16) which evaluates into 3D positions over the spline curve. This positions may be considered the smoothed counterpart to the original ones:

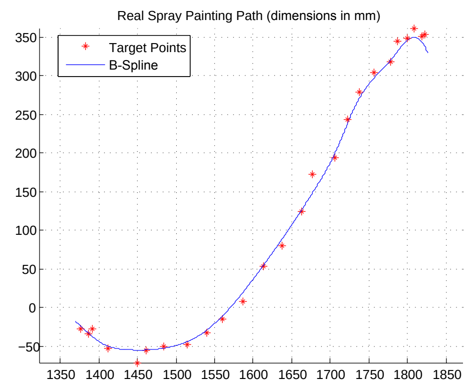
$$\{O\}_0^n \xrightarrow{\text{Parameter Space}} \{u_{eval}\}_0^n$$
$$S\left(\{u_{eval}\}_0^n\right) = \{P_{filt}\}_0^n$$

This notion of a matched pair between original and curve point is needed because the velocity information exists on the original points but not on the B-Spline — the original points hold the time stamp. Sampling the spline at these same locations allows to associate the resulting 3D position with the same velocity of the original points:

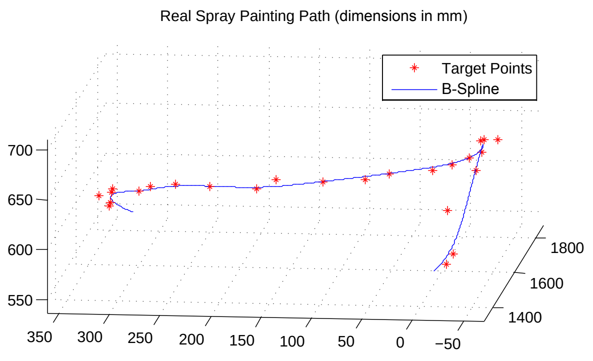
$$\{O\}_0^n \leftrightarrow \{V_o\}_0^n \Rightarrow S\left(\{u_e val\}_0^n\right) \leftrightarrow \{V_o\}_0^n$$

Additionally, it may be of interest to evaluate the B-Spline in more parameters than those of the initial positions; this is necessary to further improve smoothness in the robot playback. Since the manipulator uses linear interpolation of each point, the more points are sampled/evaluated from the B-Spline the smoother the final trajectory. Concerning position, the curve can be evaluated in as much points as needed by choosing a longer $u_{eval}$. In this case, $u_{eval}$ has a higher dimension than $u_o$ ($\{O\}_0^n \rightarrow \{u_o\}_0^n$ ). Regarding velocity, there is no information outside the initial set of tracked points so an interpolation must be done. If the velocity is assumed constant between each point than evaluating the B-Spline outside $\{u_o\}_0^n$ translates into a discontinuous velocity — it will only change when some $u_e val$ equals some $u_o$. It practice, this means that infinite acceleration is prompted to the robot at each point $u_o$. The truth is that it will always be so because the final robot path will always be a discretization of the smooth curve. Despite that, every industrial robot controller implements an acceleration and deceleration limitation to deal with this situation. Yet, it makes sense to improve velocity smoothness on pair with the position. So, if the B-Spline is being oversampled (retrieving more points than the originals) it also makes sense to at least linearly interpolate velocity. This way, the final B-Spline synchronized with the tracked velocity is at least $C^1$ continuous (obviously, after discretization the path will always be $C^0$ because the robot program is built on linear interpolations. Even though, as will be discussed further ahead, the robot controller tweaks this by implementing a smooth junction at each path point).

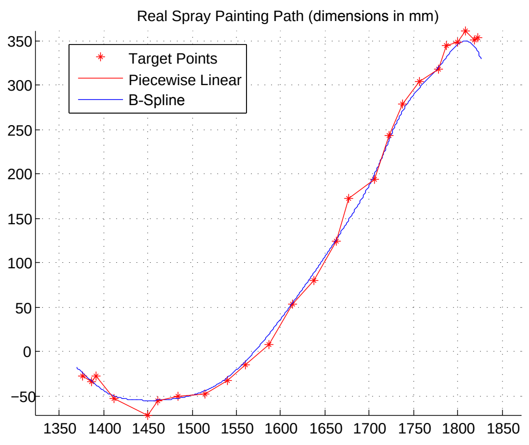
Finally, for comparison, Figure 4.19 shows the smoothed B-Spline fitted to the same data points shown previously in the beginning of this section, Figure 4.17. Much of the noise is clearly eliminated,the path is smooth, and it is close to the original points even though only one third of them have been actually used to construct the curve.



(a)



(b)



(c)

**Figure 4.19:** (a) **B-Spline 2D Upper View**, (b) **B-Spline 3D View**: These plots show the approximated B-Spline curve to the data points marked with "*". Actually only one third of the points are used in the construction of the curve. (c) **Comparison Between Piecewise Linear Approximation and B-Spline**.

#### 4.4.1.2 Orientation Smoothing

Orientation is described by a rotation matrix or, equivalently, euler angles or quaternions. Interpolation of the rotation matrix can result in degenerate solutions: the gimbal lock may occur, where one rotational degree of freedom is lost; also numerical inaccuracies can cause problems since matrices must remain orthogonal after interpolation with forces each row to have unit length and the columns to be mutually orthogonal — these conditions must be kept during interpolation. Quaternions, on the other hand, can also represent orientation and present some advantages: the mapping from angles to quaternions is unambiguous ($q$ and $-q$ do represent the same orientation but this does not bring any inconveniences); most importantly, interpolation of orientations is easy to achieve, even smooth interpolation, using simple quaternion algebra.

An unit quaternion represents orientation. Since quaternions are $\mathbb{R}^4$ entities, this means that every vector in $\mathbb{R}^4$ unit sphere represent orientation. To interpolate between two quaternions a linear interpolation can be used – Lerp; yet, the result would not lie on the unit sphere and therefore would not represent an orientation. The correct method and the most commonly used for quaternion interpolation is SLERP — spherical linear interpolation. Here, the interpolation is done across the unit sphere yielding a unit quaternion as result — Figure 4.20 shows a geometric interpretation for *lerp* and *Slerp*. One of the possible implementations [74] to interpolate between quaternions $q_0$ and $q_1$ is given by:

$$\text{Slerp}\left(q_0, q_1, t\right) = p_0 \left(p_0^{-1} p_1\right)^t \tag{4.18}$$

In short, Slerp draws a path on a sphere equivalent to a straight line in affine spaces, which is a spherical geodesic. As such, the above equation can be re-written as:

$$q_t = \text{Slerp}\left(q_0, q_1, t\right) = \frac{\sin\left(\left(1-t\right)\theta\right)}{\sin\theta} q_0 + \frac{\sin\left(t\theta\right)}{\sin\theta} q_1 \quad , \quad 0 \leqslant t \leqslant 1 \tag{4.19}$$

where $\theta$ is the angle between the two vectors — $\cos\theta = q_0.q_1$.

*Slerp* is optimal [75] to interpolate between two quaternions. Despite that, when a sequence of rotations is available (like the 6D trajectory retrieved from the tracking) *Slerp* causes the orientation to be non-differentiable. Similarly to linear interpolations in affine spaces, the resulting path has sharp edges — Figure 4.21 (a). In order to create a smooth interpolation along a set of consecutive orientations *Squad* was used — *Spherical*

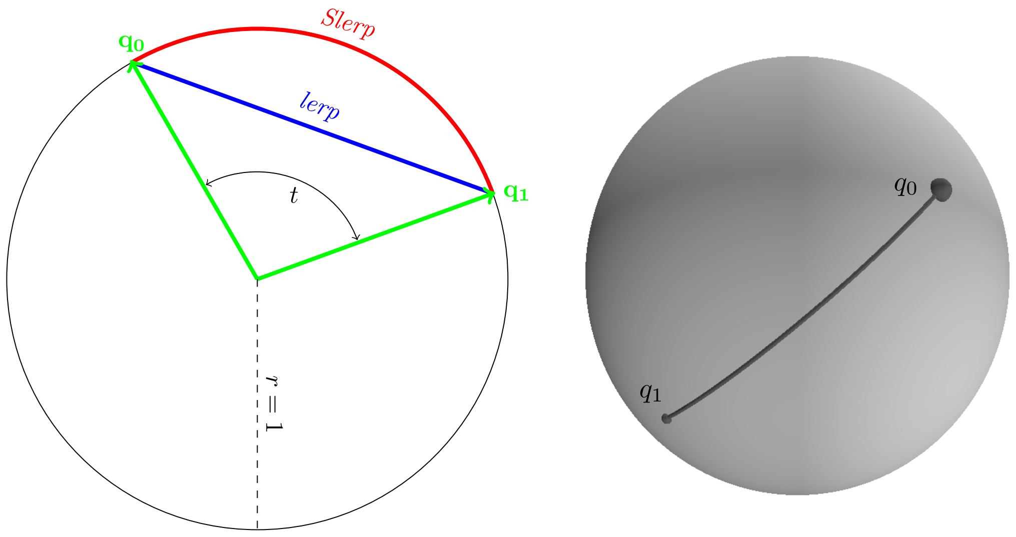


**Figure 4.20: *lerp* and *Slerp* interpolation -** On the left side there is a 2D representation of *lerp* and *Slerp*: the two quaternions $q_0$ and $q_1$ have unit length thus representing some orientation; the blue line represent the *lerp* interpolation and, as a result, blue quaternions do not have unit length; *Slerp*, on the other hand, represented by the red arc, result in unit quaternions. On the right side picture it is shown the *Slerp* interpolation on a sphere surface which gives rise to a spherical geodesic.

and *quadrangle* interpolation. *Squad* is the spherical equivalent to a cubic interpolation, and resembles a bilinear interpolation as it depends on *Slerp*. The derivation of *Squad* can be found in [76]. For what matters, *Squad* is defined by:

$$\mathrm{Squad}\left(q_i, q_{i+1}, a, b, t\right) = \mathrm{Slerp}\left(\mathrm{Slerp}\left(q_i, q_{i+1}, t\right), \mathrm{Slerp}\left(a, b, t\right), 2t\left(1-t\right)\right) \tag{4.20}$$

where $a$ and $b$ are control points. The cubic curve starts on $q_i$ and ends on $q_{i+1}$ but does not pass through $a$ and $b$. The choosing of the control points affect the layout of the curve, as it was discussed for the B-Splines, presented on the position smoothing section. To smoothly interpolate between a set of consecutive quaternions, a set of *Squad* curves can be computed: $S_j\left(t\right)$=*Squad*. Yet, each *Squad* retrieves a cubic curve, and it is necessary that consecutive curves meet at the start and end so that the final entire path is smooth. In short, $S_{j-1}\left(1\right) = S_j\left(0\right)$; and for velocity, $S'_{j-1}\left(1\right) = S'_j\left(0\right)$. With these constraints, $a$ and $b$ are determined by — details can be found at [75] and [77]:

$$a_i = b_i = q_i \exp\left(-\frac{\ln\left(q_i^{-1}q_{i+1}\right) + \ln\left(q_i^{-1}q_{i-1}\right)}{4}\right) \tag{4.21}$$

So, the definition of a spherical spline interpolation of quaternions is given by:

$$S_i\left(t\right) = \mathrm{Squad}\left(q_i, q_{i+1}, s_i, s_{i+1}, t\right) = \mathrm{Slerp}\left(\mathrm{Slerp}\left(q_i, q_{i+1}, t\right), \mathrm{Slerp}\left(s_i, s_{i+1}, t\right), 2t\left(1-t\right)\right) \tag{4.22}$$

$$s_i = q_i \exp\left(-\frac{\ln\left(q_i^{-1}q_{i+1}\right) + \ln\left(q_i^{-1}q_{i-1}\right)}{4}\right) \tag{4.23}$$

This method of spline-ing through orientations is common in computer animation to generate smooth transitions between image frames. *Squad* ensures $C^2$ continuity in the interpolated orientations. Figure 4.21 (b) shows the smooth orientation path between the target points resulting from *Squad* interpolation.

The steps to apply a *Squad* filter to the entire path are:

- Take the path $q_0, q_1, q_2, ..., q_n$ and for all $q_i$ with $0 < i < n$ (the start and end points remain unfiltered):
- compute: $q_i^s = \mathrm{Squad}\left(q_{i-1}, q_{i+1}, s_{i-1}, s_{i+1}, t\right)$; $q_i^s$ is the best orientation for point $i$ in a smoothness sense, and uses two points before and two points after: $q_i^s =$

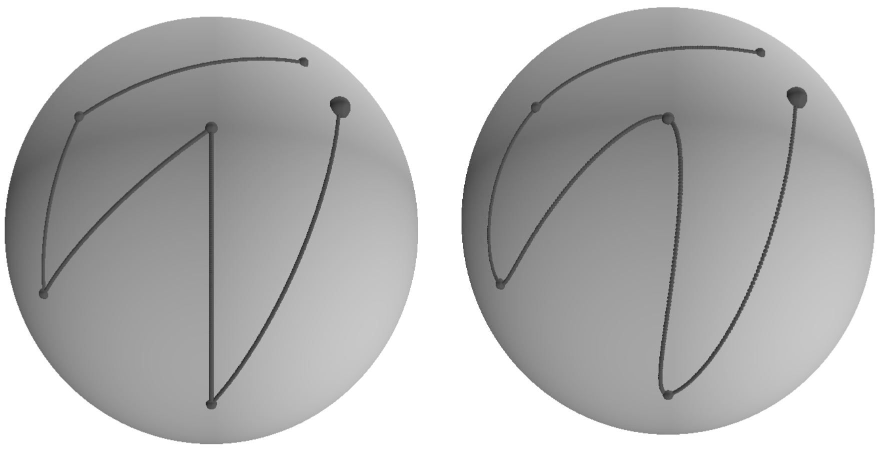


**Figure 4.21:** (a) **Slerp**: Interpolation of a consecutive number of orientations (a path of orientations) with *Slerp* yields an edged path, non-differentiable. This is similar to affine space's linear interpolation. (b) **Squad**: The same set of nodes interpolated using *Squad* gives birth to a smooth path. It is a differentiable curve and, additionally, angular velocity is also differentiable.(images credits to Erik B. Dam et all [75] )

$f\left(q_{i-2}, q_{i-1}, q_{i+1}, q_{i+2}\right)$. $t$ can be computed using the chord length method presented in the previous section; the euclidean distance is replaced by an angular measurement:

$$t = \frac{\cos^{-1}\left(q_{i-1}q_i\right)}{\cos^{-1}\left(q_{i-1}q_i\right) + \cos^{-1}\left(q_i q_{i+1}\right)}$$

- The final filtered orientation is a weighted interpolation between the smooth $q_i^s$ and the measured orientation $q_i$. For this, *Slerp* can be used. So: $q_i^{filt} = \text{Slerp}\left(q_i^s, q_i, t\right)$. As $t$ takes values from 0 to 1 it gives more weight to the *Squad* result or to the measured orientation.

### 4.4.2 Path Segmentation

This stage of data processing focus on breaking the whole trajectory in smaller segments. Industrial robots programming is about chaining movement instructions and I/O control commands. As such, path segmentation is performed on 2 steps: I/O based segmentation and movement-type segmentation.

The first step splits the path according to the synchronized I/O list (recall the synchronized lists from Fig. 4.16). This is accomplished by looking at the I/O signal time-stamp on the list and then grouping the 3D trajectory points between two I/O changes – Fig. 4.22 illustrates this procedure. This is a necessary step due to how industrial robot controllers deal with external I/O interfaces. The use of these signals, either being it reading an input or writing an output, takes place in-between movement instructions. As such, the move-type-instruction chaining must pause to insert the I/O control; then it makes sense to break the path when such event occurs is it follows the natural way robot controllers deal with path points and I/Os.

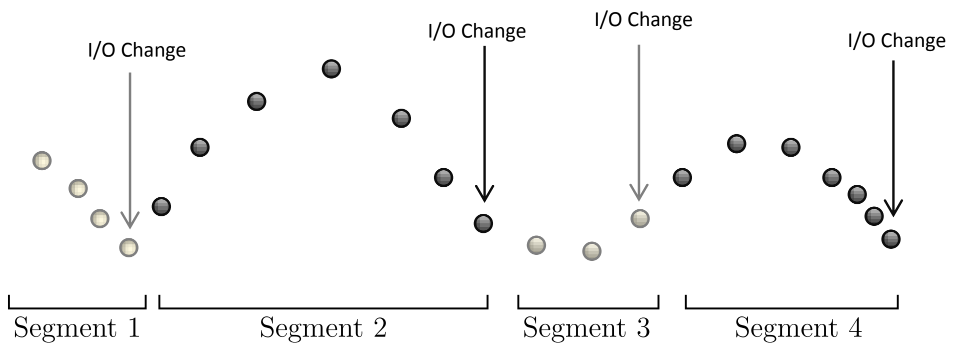


**Figure 4.22: Trajectory Segmentation Based on I/O Changes -** The first stage of segmentation is based on the I/O change list. The 6D points are grouped into sets of consecutive points in-between two I/O changes. This is a mandatory action since robot controllers execute movement instructions and I/O control sequentially.

The second step for segmentation is the interpretation of trajectory points as well defined geometric movements. Programming interfaces from standard manipulators already implement linear interpolation of both position and orientation between two 6D points; so, it is the job of segmentation to cluster a set of 6D points and write them as a single linear movement. Doing it for the entire path makes possible to chain consecutive move-linear instructions and automatically create a robot program. First, parts of the trajectory that do not show any type movement must be detected, i.e, it is necessary to identify motion stops. This is accomplished by analysing the velocity between to consecutive points: if the value is smaller than a given threshold, $\|v\|_2 < v_{thres}$, than the marker is assumed to be stopped in the same position. Computing velocity from the trajectory data is straightforward since both position $(M)$ and time$(t)$ are known are each instant $i$:

$$v_i = \frac{M_{i+1} - M_i}{t_{i+1} - t_i} \tag{4.24}$$

The remaining of the trajectory points represent a motion of some kind and are now processed in order to achieve the desired segmentation. As there are different applications and requirements, two methods are proposed:

**1. Segmentation of a free hand movement**. This method is adequate for any application where the operator may need to perform free movements, i.e, movements with high degrees of liberty. For instance, a spray painting of elaborated shapes where the spray gun needs to execute complex, unstructured moves. This method can be the starting point for any application since it is assumed that the output path is very unpredictable. This type of trajectory segmentation uses all the points collected from the tracking system and simply creates a robot instruction that moves to each point — Fig. 4.23. Velocity is computed using the 3D position and time stamp as referred above on equation 4.24. A set of move-line instructions are used to reproduce the path, making the robot perform a straight line between each point.

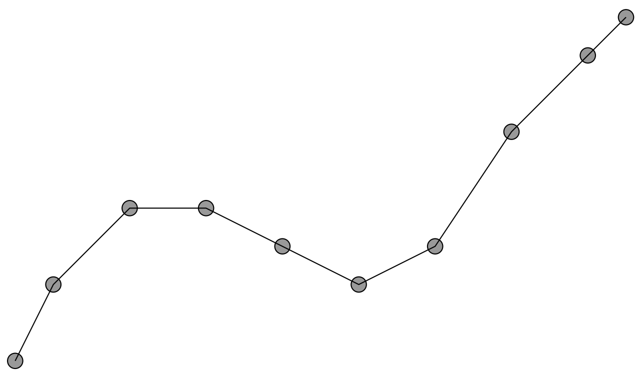


**Figure 4.23: Moving linearly to each point -** In this method of trajectory segmentation all the points are used. A move-linear instruction is used to follow the path by going through every point. The corresponding robot code is extensive but the resulting path is more faithful to the captured trajectory

**2. Path Segmentation into Move-Linear Instructions**. This method fits better to applications where there is some *a priori* knowledge about the type of movement the human operator is supposed to perform. For instance, welding of metal sheets; unlike the previous method, here it is expected that the operator performs a set of straight lines instead of a complex free movement through the whole workspace. This path segmentation method uses 4D-line fitting to break the trajectory into line segments. Given a set of points in space, $M_i = (x_i, y_i, z_i)$, there is a line that best fits the points, in a least squares sense [78]. The line $l\,(t)$ is defined by

$$l(t) = \bar{M} + t\mathbf{d} \tag{4.25}$$

where $\bar{M}$ is a point on the line and $\mathbf{d}$ is the (unit) direction vector. Each point $M_i$ can be decomposed into two components: one along the line ($m_d$) and another orthogonal to it ($m_\perp$):

$$M_i = \bar{M} + m_d\mathbf{d} + m_\perp\mathbf{d_{i\perp}} \tag{4.26}$$

where $\mathbf{d_{i\perp}}$ is the unit vector orthogonal to the line passing through $M_i$. $m_\perp$ is the component of $M_i$ that contributes to the fitting error, i.e, the orthogonal error. The squared distance is

$$e_i = (\|m_{i\perp}\mathbf{d_{i\perp}}\|_2)^2 = m_{i\perp}^2 \tag{4.27}$$

Then, the least squares minimization sets its goal to

$$\min\sum_{i=0}^{n} e_i = \min\sum_{i=0}^{n} m_{i\perp}^2 \tag{4.28}$$

From Equation 4.26:

$$m_{i\perp}^2 = \left(M_i - \bar{M} - m_i\mathbf{d}\right)^2 \tag{4.29}$$

The solution for this minimization [78] sets $\bar{M}$ as the average of the points $M_i$

$$\bar{M} = \begin{pmatrix} \bar{x} & \bar{y} & \bar{z} \end{pmatrix}\ ,\ \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{4.30}$$

that is, the line goes through the average of points. The direction $\mathbf{d}$ is given by the singular value decomposition of the covariance matrix $\mathbf{Q}$:

$$\mathbf{A} = \begin{bmatrix} M_1 - \bar{M} \\ \vdots \\ M_n - \bar{M} \end{bmatrix} = \begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} & z_1 - \bar{z} \\ \vdots & \vdots & \vdots \\ x_n - \bar{x} & y_n - \bar{y} & z_n - \bar{z} \end{bmatrix} \tag{4.31}$$

$$\mathbf{Q} = \mathbf{A}^T\mathbf{A} \tag{4.32}$$

$$\begin{bmatrix} \mathbf{U} & \mathbf{S} & \mathbf{V}^T \end{bmatrix} = \mathrm{SVD}(\mathbf{Q}) \tag{4.33}$$

**d** is the eigen vector $\mathbf{v_i}$ corresponding to the largest singular value $\sigma_i$ of **Q**

$$\mathbf{V} = [\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}]\,; \quad \mathbf{S} = \mathrm{diag}\,(\sigma_1, \sigma_2, \sigma_3) \tag{4.34}$$

Since each point of the trajectory has a time stamp, it means that the path can be seen as a 3D-positions time series. To split the entire path into lines, the line fitting is recursively used in the following steps:

- Fit all data available to a 3D line and compute the error:

  $$e = \sum_{i=1}^{n} \mathrm{dist}_{\perp}\,(M_i, l)^2 \tag{4.35}$$

  $\mathrm{dist}_{\perp}$ is the procedure that computes the orthogonal distance from the point to the line.

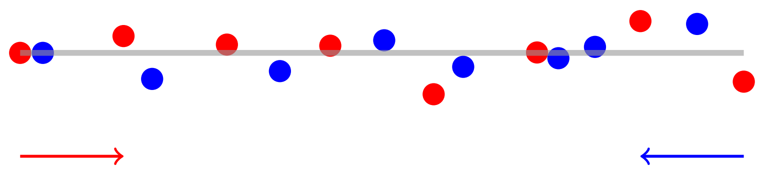
- If the error is too large, $e > \sigma_{max}$ ($\sigma_{max}$ is an adjustable parameter, defining how rough the approximation can be), find the point $M_{me}$ that has the largest error $\max_i\,(e_i)$.

- Take all points up to $M_{me}$ (up to its time) and start again.

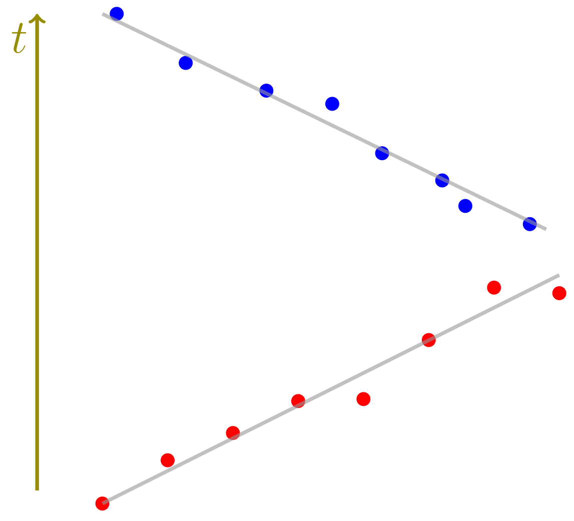
- When a line is fitted, re-run the algorithm for the remaining points (from $M_{me}$ onwards).

The 3D fitting has a problem with backtracks, i.e, if the path follows one direction and then comes back (going back and forth in the same direction), the whole set of points are approximated by a single line. Fig. 4.24 (a) shows an example of this situation; red points move from left to right and blue points from right to left. A simple way to solve backtracks is to extend the 3D line fitting into 4D where the added dimension is time. As such, equations 4.25-4.34 are extended into 4 dimensions with $M_i = (x_i, y_i, z_i, t_i)$ — those equations remain valid for $n$D. The geometric interpretation is shown in Fig. 4.24 (b): backtracks clearly become distinct directions.

The previous algorithm is computationally fast given that the trajectory has been previously split by I/O changes and motion stops — so that the number of points to consider in each step is already reduced.



(a)



(b)

**Figure 4.24:** (a) **A back track in 2D/3D**: using only spacial information backtracks are not detected. The red dots represent positions of a movement going from the left to the right; blue circles are a path in the opposite direction. Using all the points in a 3D line fitting method comes up with the grey line. (b) **Adding time as an extra dimension to data**: the image represents a scheme of adding a 4th dimension (time) to each point. Backtracks now clearly point in a different direction. Note: the noise on red/blue dots are not in the time axis; this is still a variation on the $xy$ plane; the visual feeling results from the 2D view.

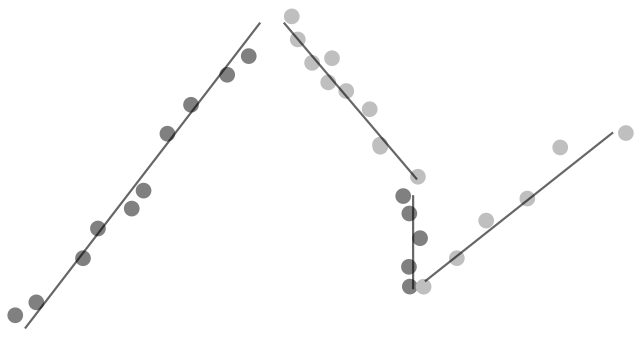


**Figure 4.25: Trajectory Segmentation in Linear Movements -** After the recursive 4D line fitting algorithm the trajectory is composed of consecutive lines. This method is useful for applications where it is known that the operator wants to execute well structured movements; this segmentation removes much of the noise associated with hand jitter.

The resulting robot path is more ”rigid” than the previously presented method since there are now less and longer segments — Fig. 4.25.

Finally, the sequence of line segments $l_i\left(t\right)$ defined by the start and end points, $l_i\left(0\right)$ and $l_i\left(1\right)$, make up the path. Yet, in long lines, care must be taken for non-uniform linear and angular velocity. To address this issue, each line segment is evaluated at different positions $t$, computed using once again the chord length of the original points. This way, there is no averaging the velocity, and linear interpolating rotation, between the start and end points. Instead, the segment is divided and, for each piece, the velocity is computed from the original points, and orientation is linearly interpolated within each piece. This method forces the whole line to have different linear and angular velocities, and does not assume the shortest path of orientation between ends. If the original path does not have such dynamics than this is just a point over-head in the final program; if such changes do occur, the playback is able to mimic it.

The line-segmentation has the added effect of smoothing both position and orientation. Since a group of consecutive points is replaced by a line, the noise between the start and end points are substantially reduced. Additionally, the move-line instruction of industrial robots also implements a linear interpolation of orientation so rotation is also smoothed using this approach.

At this point, by either method, the list of consecutive linear movements can be used to automatically generate a program for the industrial manipulator.

### 4.4.3 Automatic Code Generation

The last stage on the programming by demonstration framework is to generate the robot program. Taking the points/lines obtained from the previous algorithms and stacking up a list of strings with robot instructions is rather simple. Yet there is one last issue that must be addressed: velocity smoothness. It is not related to the data itself but to how robot controllers interprets the program and do the joint control. When the robot travels from point $M_0$ to point $M_1$ by a move-linear instruction

$$\text{instruction:} \quad \text{move-linear}, \quad M_1, \quad V = V_1, \quad \text{Tolerace}{=}Tol \tag{4.36}$$

the linear velocity $V$ must be specified along with a tolerance, *tol* parameter. As already discussed, velocity is computed from the current and previous point and their respective time stamp: $v_i = (M_i - M_{i-1}) \,/\, (t_i - t_{i-1})$ (or the start and end points of line segments). Tolerance sets how close to position $M_1$ the robot actually moves to. Usually this parameter is specified in millimetres. Figure 4.26 (left) shows a geometric interpretation of this parameter (in 2D): as soon as the robot's tool tip is at a distance *tol* from the target $M_1$ the controller assumes the movement has been accomplished and stops the robot or starts the next instruction. Additionally, the robot controller implements a trapezoidal control for robot velocity, which means that when it leaves point $M_0$ it starts accelerating then it reaches nominal velocity $V_1$ during the line, and finally, starts decelerating until $M_1$. If the tolerance parameter is kept too low, for instance zero (the robot moves to $M_1$ with full accuracy) then the robot will decelerate until a stop and only then moves on to the next target. This generates a "hiccup-like" movement due to the full stop at each point. If the target points from the tracking system are directly used in the movement instructions then two situations can happen:

1. The tolerance parameter is set to zero, *tol* $= 0$, to insure the robot goes exactly through the points captured by the vision system. This is the ideal in terms of positioning. Yet, due to the full stops at each point, the movement has no velocity continuity; instead of continuously flowing through the points, the robot stops at each line-end causing the sobbed behaviour.

2. The tolerance parameter is set to some value grater than zero, in such a way that when it enters the tolerance sphere it has not started to decelerate (the tolerance value must be accessed by investigating the acceleration and deceleration

parameters of each robot type/brand). If there are any further instructions, the robot will start executing them before coming to a stop. This ensures velocity is smooth as lines are performed one after the other with no deceleration due to the positioning controller. This smoothness comes with the price of adding error to the position; the target points from the tracking are not fully respected.

To overcome this issue additional control points are computed from the desired path. So, for a sequence of moves $M_0 \rightarrow M_1 \rightarrow M_2$, control points $M_1'$ and $M_1''$ are created as an extension in the direction of the movement — Fig. 4.26 (right) — such that

$$M_i' = M_i + tol * \frac{M_i - M_{i-1}}{\|M_i - M_{i-1}\|_2} \tag{4.37}$$

It means that instead of moving to $M_1$ the robot will move to $M_1'$ which is further away from $M_1$ by the distance *tol* in the same direction of the movement. Then, the tolerance parameter can be effectively applied because the robot will not reach $M_1'$: it will start moving in that direction and will stop (or proceed to the next step) when it is at a distance *tol* from $M_1'$ therefore being over $M_1$. $M_1''$ is a point in the direction of $M_2$ but closer to $M_1$ so that the robot does not follow the line $M_1'$ — $M_2$ but stays on $M_1$ — $M_2$.

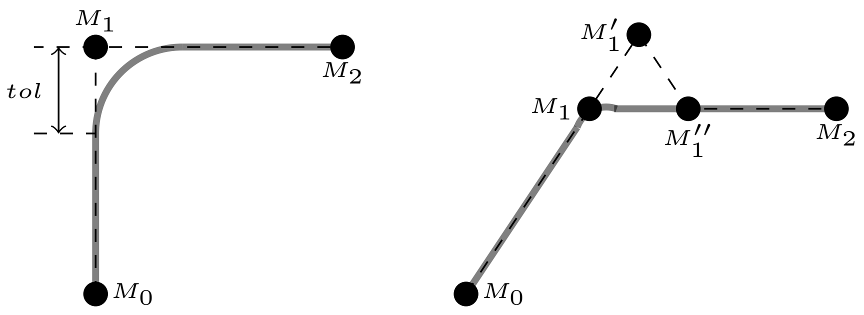


**Figure 4.26: Tweaking the Robot Trajectory Controller Tolerance -** On the left side of the picture there is a geometric interpretation of the robot tolerance parameter: upon reaching at a distance *tol* of the target point the robot starts executing the next instruction; it does not go exactly over $M_1$. On the right side there is a scheme for the compensation of this feature. The robot is given a target point a little bit further ahead from the real target in order to assure that it actually goes through $M_1$.

This tweak ensures velocity smoothness along the path as well as smoothness in the transition between consecutive lines due to the small arc the robot performs in order to maintain velocity constant.

From here, the code generation is straightforward as points give rise to robot instructions:

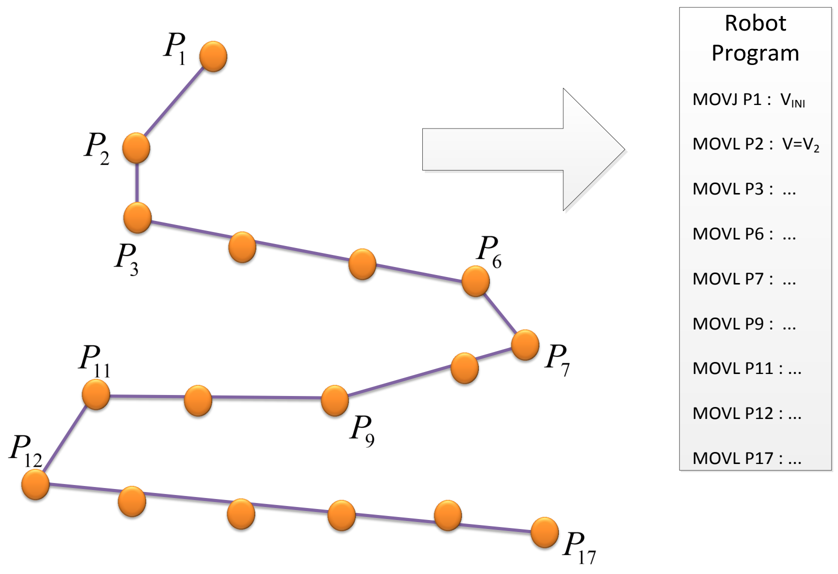


**Figure 4.27: Automatic Robot Program Generation -** From the 6D points and velocities is it easy to automatically generate a robot program: each point or line end is mapped into a move-line instruction. This approach is valid for any robot brand as this instructions work the same way in industrial manipulators.

# Chapter 5

# Tests and Results

This chapter presents the tests and results that show the performance of the human-robot skill transfer framework.

The first section introduces the industrial demonstrator with a description of the scenario and the hardware. The following sections present the results for each module of the motion tracking and robot interface. All tests were carried out at the demonstrator and the demonstrated tasks are from real spray painting applications.

## 5.1 Industrial Demonstrator — *Setup* and Hardware

As indicated in the introductory chapter, Flupol [3] served as an industrial demonstrator for the application and validation of the programming by demonstration framework.

Below is presented a list with the implementation details of the instalment in Flupol facilities of the motion imitation system. It includes the hardware specification and layout geometry/dimensions, both of which directly influence the performance of the motion mimic. As such, the remaining sections of this chapter detailing the tests and results are based on this implementation.

Description of Flupol's motion imitation system:

- The painting takes place in one of Flupol's coating cells; it has been adapted to accommodate the robot, its controller and the industrial cameras. The parts are painted on a turn-table so that the operator does not need to walk around the part and the over-spray can be sucked in by the air-suction system. Both the painter and the robot operate in the same space — Figure 5.1 shows the painter and the turn-table and Figure 5.2 shows the robot occupying the same place.

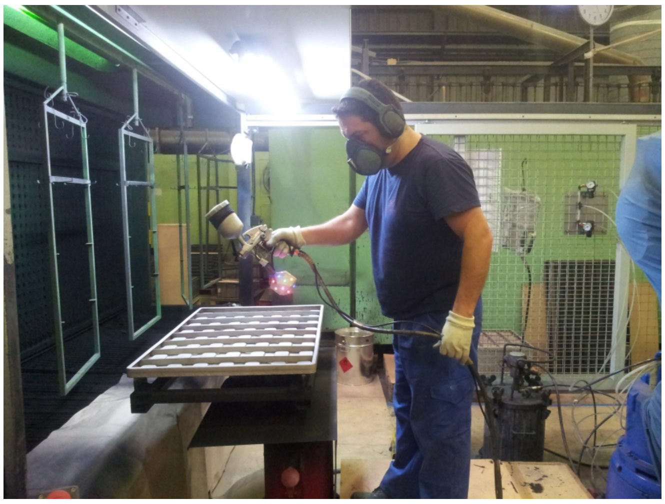


**Figure 5.1: Human Painter in Flupol Coating Cell -** This figure shows the work area with the painter operating over a baking tray held by a turn-table.

- The total workspace dimension is ($x \times y \times z$) 1000mm×1000mm×400mm. The horizontal axis on Figure 5.2, for the forward and backward robot movement is the $x$ axis. The vertical direction on the figure is the robot $z$ axis. The depth direction is the $y$ axis. In this workspace Flupol coats mainly baking trays and molds, of maximum size of 800×800mm.

- To cover up the entire workspace, a MOTOMAN EPX2050 robot was installed. It has approximately 2000mm of horizontal reach which enables it to easily move into any area of the workspace.

- The marker has 50mm radius and was attached on the bottom of the spraying gun — Figure 5.3. This way the marker does not get into the operator's line of sight, enabling him to maintain eye feedback on the process. Also, the LED lights cause much less eye disturbance while on the bottom side of the gun.

- To capture the marker a single pair of industrial cameras was used. Figure 5.4 shows the cabinet which holds the cameras. Even though a simple tripod would be sufficient, a larger and more robust cabinet was installed. It is able to protect the cameras from dust while they are not in use and, at the same time, it is very

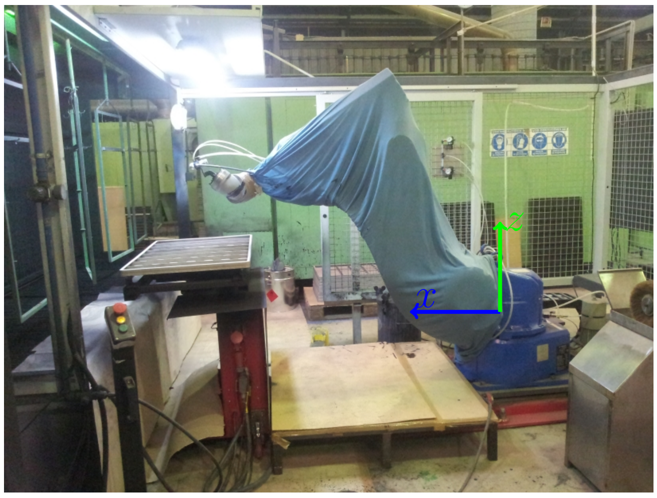


**Figure 5.2: Robot in Flupol Coating Cell** - This figure shows the work area with the robot operating over a baking tray held by the turn-table. The manipulator occupies the same place and the human did for demonstration (Figure 5.1).

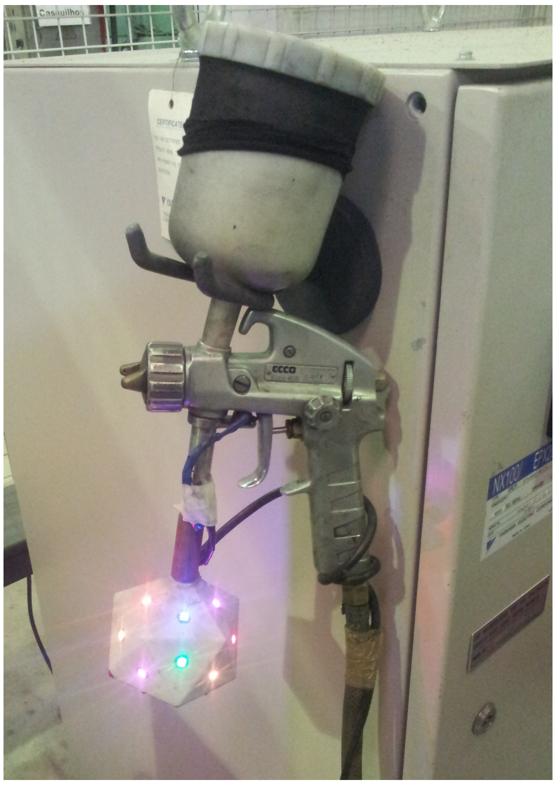


**Figure 5.3: Marker Attached to the Industrial Spray Painting Gun** - The luminous marker is attached to the bottom side of the spray gun so that it stays clearer from the spraying and the operator's line of sight.

robust so that the cameras' positioning remain unchanged. This way, the camera calibration process needs to be executed only once.

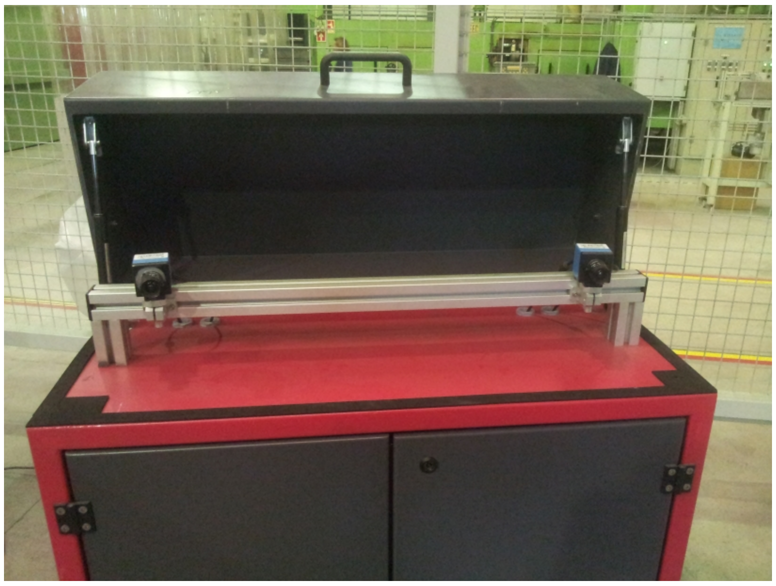


**Figure 5.4: Two Camera Arrangement for Stereoscopy** - The figure shows the cabinet that holds both cameras. These are separated by 70mm and slightly rotated towards each other. The cabinet has the function of maintaining the cameras' positioning and protect them from dust.

- The cameras are placed 700mm apart and are slightly rotated towards each other. Each device is a CCD 1024×768 RGB industrial camera with USB 2.0 connectivity.

- The use of a single pair is justified by both financial and logistic reasons. Two-cameras arrangement is the bare minimum to accomplish stereo and so it is the most cheap solution to start with. Additionally, there were strong spatial restrictions on the cameras' positioning; there is the problem of avoiding the over spraying and respecting the cell limits, while causing the minimum impact in the process and the cell layout; a second pair of cameras could not be installed because the opposite area is used as loading/unloading buffer.

- As a consequence of the reduced number of cameras, the painter needs to make caution during the demonstration stage not to step in-between the marker and the cameras. Such an action causes occlusion of the marker and jeopardizes the motion tracking.

- The synchronization device, which delivers the cameras and marker triggers, is made up from an *arduino* board. A second board is used to capture some I/Os such the spray gun trigger and the commands to operate the turn-table. Such I/Os are also connected to the robot controller making possible for the robot to operate the devices in the same way the human did.

In short, the coating cell was upgraded with an industrial robot, a pair of industrial cameras, a small box containing the micro-controller boards and a desktop PC. The extra apparatus specifically used to program the robot in an intuitive way, i.e, the hardware that makes it possible to program the manipulator using motion demonstration is limited to the pair of cameras, the marker and the synchronization devices. In terms of pricing (of the equipment, not taking in concern design, installation/assembly and configuration), this is comparable to an extra PC or two thus a very small fraction when compared to the price of an industrial manipulator.

## 5.2 Tests and Results

This section provides detailed description of the tests used to define the motion demonstration framework performance. Results are presented for each subsystem: from the precision of a single LED detection, to the final stage of trajectory smoothing; in-between these there are results for the colour classification, the success rate of the marker pose estimation and its accuracy in terms of translation and orientation. Finally, the timings of each step are analysed to prove the real-time capability of the framework as well as its ability to mimic motion right after the demonstration with no apparent post-processing overhead.

### 5.2.1 Camera Calibration

Camera and stereo calibration is achieved using the grid pattern to collect the set of pixel $\leftrightarrow$ $3D$ correspondences. For the $1000\times1000\times400$ $mm$ workspace, the grid cell size was set to $100mm$ which resolves to $10\times10\times3=300$ $3D$ points and pixel pairs. Figure

5.5 shows the CAD model of the developed tool which holds the calibration LED and is attached to the robot end-effector.

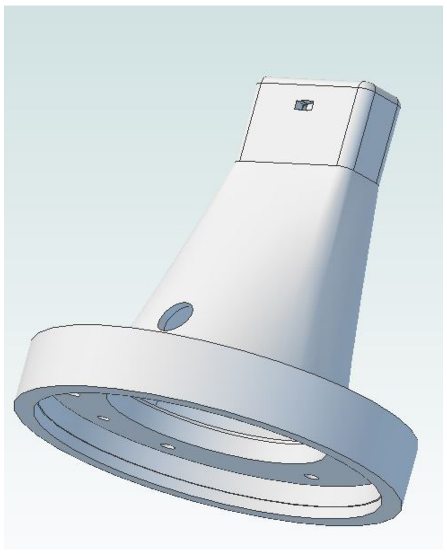


**Figure 5.5: CAD Model of the Calibration Tool -** This object is attached to the robot end-effector and holds a single LED near the tip. This LED is turned ON at each node of the calibration grid and allows capturing a pair of images corresponding to the node position in space.

Tables 5.1 and 5.2 shows de calibration error for the stereo arrangement. The analysis is done in two ways: the deviation in each axis alone, $x/y/z$, and the absolute deviation of the stereo 3D point to the real known coordinate. The error is defined by the $n$ calibration points $P_i = (x_i, y_i, zi)$ and the corresponding estimates from the stereo $\tilde{P_i} = (\tilde{x}_i, \tilde{y}_i, \tilde{z}_i)$. The $n\times 3$ error matrix $\mathbf{E}^{(xyz)}$ is defined as

$$\mathbf{P} = \begin{bmatrix} x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{bmatrix} = \begin{bmatrix} P_1 \\ \vdots \\ P_n \end{bmatrix}$$

$$\mathbf{E}^{(xyz)} = \mathbf{P} - \tilde{\mathbf{P}}$$

$\mathbf{E}_i^{(xyz)}$ is the cartesian deviation on point $i$. The total deviation error at each point is given by $\mathbf{E}_i = \left\|\mathbf{E}_i^{(xyz)}\right\|_2$, i.e, the distance from the real and the estimate position.

Figures 5.6,5.7 and 5.8 hold plots for the variation of the absolute error in the $x$,$y$ and $z$ axis.

**Table 5.1: Calibration error on $x$, $y$ and $z$ axis**: Four measures are presented for each axis: the mean error, the mean absolute error (MAE), the maximum absolute error and the root mean squared error (RMSE). Since the model is unbiased, $\bar{e} \approx 0$, the RMSE coincides with the standard deviation.

| Calibration Error (X, Y, Z) (all measures in mm) | x | y | z |
|---|---|---|---|
| Mean Error: $\bar{\mathbf{E}}^{(xyz)} = \frac{1}{n}\sum \mathbf{E}^{(xyz)}$ | 2E-05 | -5E-04 | 3E-05 |
| Mean Absolute Error: MAE$= \frac{1}{n}\sum \left\|\mathbf{E}^{(xyz)}\right\|$ | 0.36 | 1.23 | 0.34 |
| Max. Absolute Error: $\max e_i = \left\|\mathbf{E}^{(xyz)}\right\|_\infty$ | 2.20 | 5.29 | 1.22 |
| Standard Deviation: $\sigma = \sqrt{\frac{1}{n-1}\sum_i \left(\mathbf{E}_i^{(xyz)}\right)^2}$ | 0.47 | 1.58 | 0.42 |

**Table 5.2: Calibration error**: It is measured as the euclidean distance from the expected position $P$ to the estimated $\tilde{P}$.

| Calibration Error — $\left\|P - \tilde{P}\right\|_2$ (all measures in mm) | |
|---|---|
| Mean Error | 1.4 |
| Max Error | 5.7 |
| Std. Deviation | 9.8E-01 |

Table 5.1 shows that the stereo model is unbiased in every direction since the mean error is approximately zero in each axis. As expected, the smaller errors are present in $x$ and $z$ axis as they are fairly aligned with camera's $u$ and $v$ axis, respectively — the absolute error is almost the same for $x$ and $y$.

The measures along the $y$ axis show larger errors and are the major responsible for the total deviation error presented on Table 5.2. The average position error is 1.4mm and the average $y$ error is 1.3mm; the same happens for the maximum error. This is justified with the fact that the $y$ axis stands as the stereo depth axis. With a two camera arrangement it is expected that points further away present larger errors. This can be seen at Figure 5.7: the negative Y coordinates are further away from the cameras and show increasing error as distance increases (cameras' $y$ position coordinate is approximately 3000mm). On the other hand, the error remains almost constant among $x$ and $z$ directions — Figures 5.6 and 5.8, respectively. The $x$ axis has increased error in the plot's rightmost end since the positions get closer to the image border thus being affected with lenses distortion. Even though, the error show much more independence on $x$ and $z$ axis whilst it clearly increases in the $y$ direction (right to left means stepping away from the cameras). It can be concluded that increasing camera resolution will have limited benefit to the global stereo precision as it primarily affects $u$ and $v$ directions, $x$ and $z$ correspondingly; instead, having more cameras can significantly enhance the system precision. Particularly, having a second camera pair opposite to the first pair can be most advantageous. In such scenario, the error for negative $y$ coordinates would be a mirror of the positive coordinate tendency.

### 5.2.2 Marker Detection

#### 5.2.2.1 Colour Calibration

Colour calibration is achieved by moving the marker on the entire workspace with only one colour lit at a time. Figure 5.9 shows the HSV circle with the sampled hue values for each colour (each line represents a measure). It can be clearly seen that any two set of samples are disjoint except for yellow and red. Red is the most well defined colour (lowest hue variance among the samples) but yellow LEDs are made up from red and green; the red contribution imposes in some conditions and yellow clusters

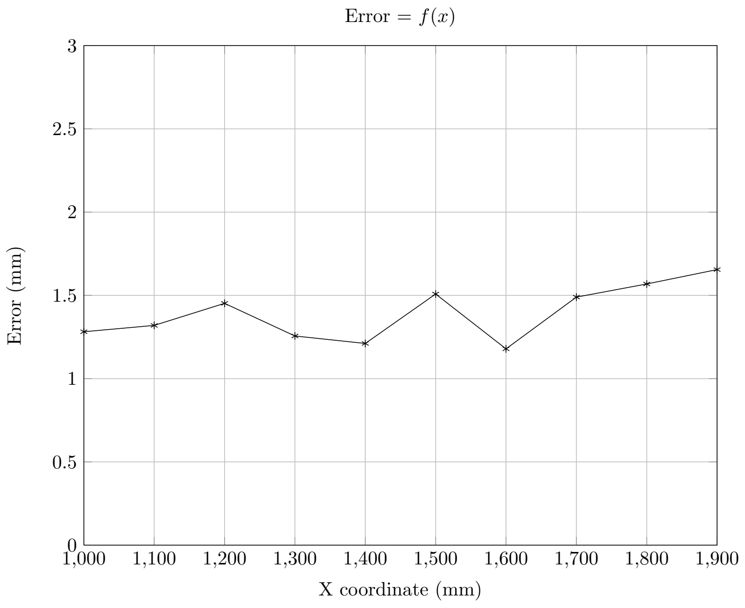


**Figure 5.6: Error variation in the $X$ axis -** The figure plots the calibration error as a function of the position's $x$ coordinate.

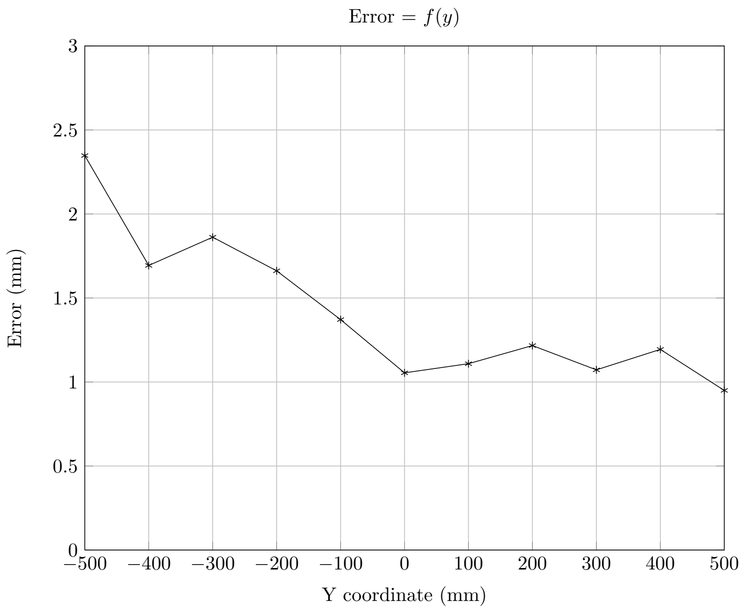


**Figure 5.7: Error variation in the $Y$ axis -** The figure plots the calibration error as a function of the position's $y$ coordinate.

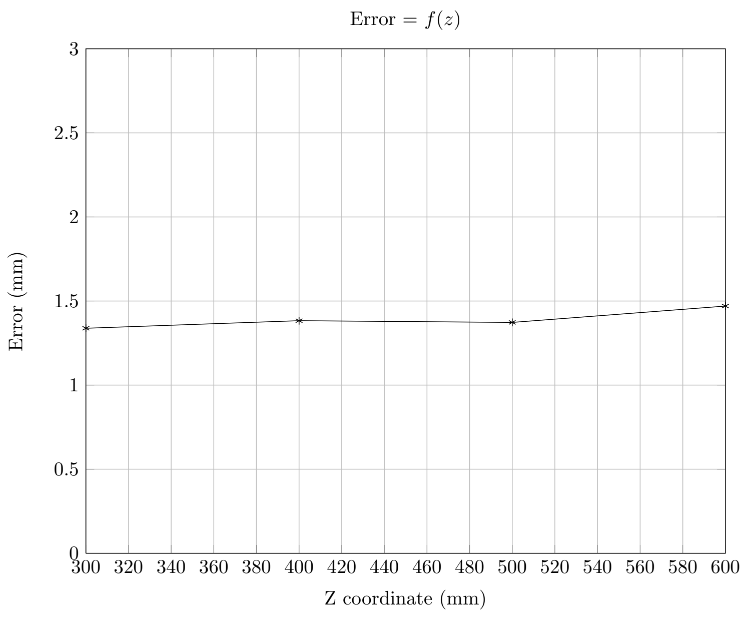


**Figure 5.8: Error variation in the *Z* axis -** The figure plots the calibration error as a function of the position's $z$ coordinate.

are evaluated as red. As it will be made clearer ahead, this class interception do not jeopardize the marker tracking.

#### 5.2.2.2 Detection and Pose Estimation

The *sincrovision* concept, allied to optimized camera exposition and LED driver electronics allow a very robust detection of each individual LED in the image.

Camera exposure was set to $3ms$ which is also the ON time for the LEDs. With this short duty cycle, $3ms$@25fps = 7.5%, LEDs were driven at 10 times their rated current.

The detection was evaluated according to the number of (un)successful marker detection: how many times individual LED colours were misinterpreted, how many times position and orientation estimation failed.

*Remark*: These results intend to state the quality and robustness of the implemented algorithms and not of any specific hardware. Therefore, the following results do not take in concern possible hardware failures; for instance, the USB cameras failing to deliver some image frames and/or the operating system drivers failing to put together

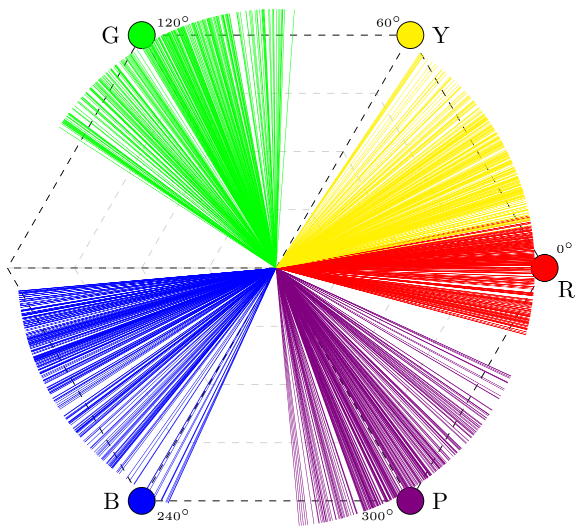


**Figure 5.9: HSV Colour Classes Calibration** - The figure shows the collected samples from each LED after moving it around the workspace. Almost all classes are separable with exception to yellow and red.

the received data in the USB bus thus bringing forth a misconceived image. Some of these problems were experienced in the industrial setup and worsen as more cameras were plugged into the same computer. Despite being rare occurrences, the results for this section were evaluated only on good image frames.

The following results were determined after the evaluation of 50000 paired image frames from roughly 30 different human demonstrations. Table 5.3 summarizes the accuracy for image cluster classification. That is, given an image with a number of visible LEDs, how many times each colour was misclassified. The evaluation was done over 50000 stereo frames and this analysis is image-wise, therefore the results are for a total of 100000 image frames. False positives (F/P) for red mean that the cluster is of another colour and is classified as red; false negatives (F/N) mean that the cluster is red and was classified as other; the same reasoning applies to all other colours.

For the 50000 stereo pairs, the successfulness of pose estimation is resumed in Table 5.4. It is divided in position and orientation estimation because different factors contribute for the output of each algorithm. Additionally, for the unsuccessful measures, it is given the number of failures (in brackets) derived from colour misclassification.

The colour classification results, from Table 5.3, are clearly in tune with the results

**Table 5.3: Number of misclassification of image clusters sorted by colour**: F/P are the false positives (another colour classified as the current) and F/N are false negatives (the current colour classified as some other).

| Image Analysis: 100000 frames | | |
|---|---|---|
| Cluster Colour Classification | | |
| | F/P | F/N |
| Red | 4544 (4.544%) | 0 |
| Green | 0 | 0 |
| Blue | 0 | 0 |
| Yellow | 0 | 4544 (4.544%) |
| Purple | 0 | 0 |

obtained in colour calibration (discussed above, in Figure 5.9). All the colours are perfectly classified except for yellow. Yellow LEDs evaluation on hue parameter is very close to red. As such, there are many detections where a yellow cluster is tagged as being red. Despite that, the pose estimation algorithm is robust enough to eliminate these misclassification; as depicted in Table 5.4, only 37 detections went wrong due to the bad classification. On one hand, the misclassified yellow do not have a matching cluster on the other image so it will never give birth to a 3D point that can used in the position estimation; the sphere fitting is unaffected by colour classification. On the other hand, if the yellow LED would complete a ”Y” in the image — the pattern used for orientation estimation — then the misclassification may lead to having no complete ”Y”; this is the scenario that contributes for 37 orientation estimation failures; the remaining are rare situations where no complete ”Y” is seen yet all colours are well classified.

Over the 30 demonstrations used for these tests, the pose could not be estimated for an average of 3 times per demonstration. It means that there are three $40ms$ ”holes” in the final path. This is completely negligible over a 1500-positions/one-minute-long trajectory. The smoothing algorithms will later fill in these holes.

**Table 5.4: Evaluation of the position and orientation estimation**: The success and failure rate for each algorithm is presented for a total of 50000 successfully paired image frames. Since orientation depends on colour classification, the number of failures due to colour misclassification is shown in brackets. The complete pose estimation performance is held at the last sub-table.

Stereo Frames: 50000 pairs

| Position Estimation | |
|---|---|
| Success | Failure |
| 50000 [100%] | 0 [0%] |

| Orientation Estimation | |
|---|---|
| Success | Failure (due to colour) |
| 49919 [> 99.8%] | 81 (37) [< 0.2%] |

| Complete Pose Estimation | |
|---|---|
| Success | Failure |
| > 99.8% | < 0.2% |
| average/per demonstration (1500 frames) | |
| < 3 fails | |

### 5.2.3 Marker Accuracy in the Workspace

The pose precision was accessed using the same principle of camera calibration: the marker was attached to the robot's end-effector and the same routines were used to sweep the entire workspace. Therefore, the robot moved in a grid like pattern and samples were collected at each node.

To estimate position accuracy the manipulator was set to move in a $100mm$ grid. This renders a total of 305 positions; at each position 10 samples were collected. The (robot) coordinates from the 305 test positions are well known as they are read from the robot controller; also, the manipulator was moved to the target positions and allowed to remain there for 4 seconds in order to minimize vibrations.

To estimate orientation accuracy, the robot was stopped at 5 different positions. At each one, the orientation was incrementally changed by 30° for a total of 60 different orientations: for example, $R_x = [0, 30, 60] \times R_y = [0, 30, ..., 90] \times R_z = [-60, -30, ..., 60]$. In the end there are 300 orientations to evaluate.

Tables 5.5 and 5.6 resumes the marker pose accuracy; the analysis is done along each direction and also for the euclidean error/shortest angle between the position/orientation estimate and the ground truth.

**Table 5.5: Marker position error on $x$, $y$ and $z$ axis**: Four measures are presented for each axis: the mean error, the mean absolute error (MAE), the maximum absolute error and the standard deviation.

Marker Pose Error (X, Y, Z) — $(R_x, R_y, R_z)$
(all distances in mm, angles in degrees – °)

| | x | y | z | $R_x$ | $R_y$ | $R_z$ |
|---|---|---|---|---|---|---|
| Mean Error | 0.52 | -2.1 | 0.9 | 0.5 | -0.2 | -0.5 |
| Mean Absolute Error | 1.1 | 3.2 | 1.9 | 1.0 | 0.8 | 1.1 |
| Max. Absolute Error | 3.6 | 8.2 | 5.4 | 4.1 | 3.9 | 3.6 |
| Standard Deviation | 1.5 | 2.9 | 1.7 | 2.2 | 2.1 | 1.8 |

**Table 5.6: Marker pose error**: Marker position error measured as the euclidean distance from the expected position $M$ to the estimated $\tilde{M}$. For orientation, the absolute error is the shortest path (angle) between the two orientations which is computed from the angle between the two corresponding quaternions, $q_M$ and $q_{\tilde{M}}$.

Marker Position Error: $E = \left\| M - \tilde{M} \right\|_2$, and
Orientation Error: $\theta = 2\cos^{-1} q_M^{-1} q_{\tilde{M}}^{*}$
( all distances in mm, angles in degrees $^\circ$ )

| | $E$ | $\theta$ |
|---|---|---|
| Mean Error | 3.8 | 1.7 |
| Max Error | 8.9 | 6.2 |
| Std. Deviation | 2.7 | 2.1 |

* shortest angle between 2 quaternions

Comparing these results (Tables 5.5 and 5.6) with those for a single LED in the calibration procedure (recall Tables 5.1 and 5.2) it can be seen that the error has increased globally. With the two-camera arrangement, going from one LED (which tracks translation alone) to the whole marker (which tracks six degrees of freedom) penalizes accuracy in a few millimetres. First, the error pattern is no longer unbiased; it is specially noticeable along the $y$ direction, i.e, the depth axis. This happens due to the biased placement of the cameras, that is, the pair of devices being installed only on one side of the workspace; the sphere fitting algorithm receives samples from just a small area around the sphere. Since the cameras are aligned horizontally, i.e, along the robot $x$ axis, the position error tends to be lower along that direction. Concerning orientation, each direction seems equally affected by the estimation noise.

### 5.2.4 Filtering and Smoothing

Evaluating the performance for the B-Spline smoothing is not easy since there is no measure of the real path of the human hand; that is, there is no ground truth for the free hand movement. Therefore, a set of path segments were analysed and are presented for visual inspection. The comparison is always done against the piecewise linear path, i.e, when the tracking points are connected through a line.

The following results were recorded when the human operator painted a baking tray as the one depicted in Figure 5.10. The tray was held at the turntable and the painting is executed in four moments: the painter sprays the whole surface, then rotates the table and sprays again the tray, and this is repeated twice more. For a visual feedback of the path and the smoothing output the results are shown just for a small illustrative segment of the trajectory so that the plots are more readable.

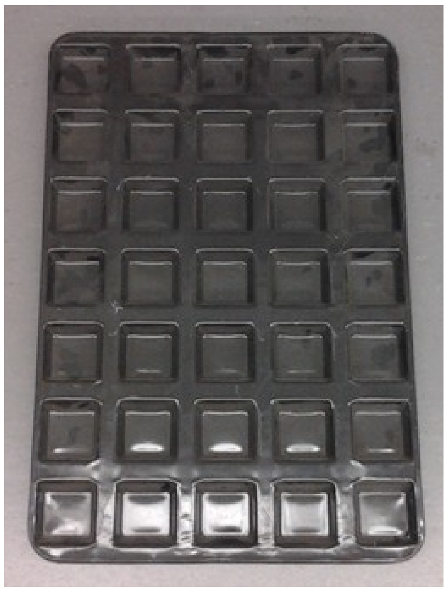
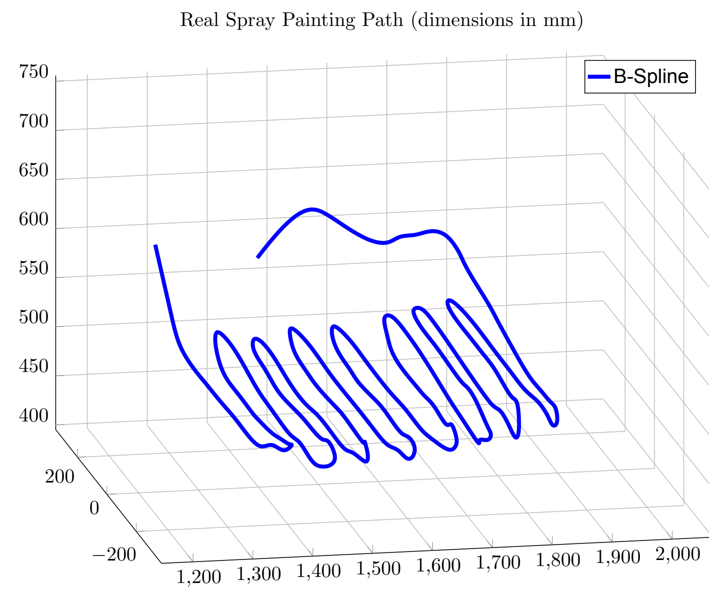


**Figure 5.10: A Baking Tray -** One of the work pieces coated at Flupol. The discussion on the trajectory features for the remaining of this section is based on the painting path for this shape.

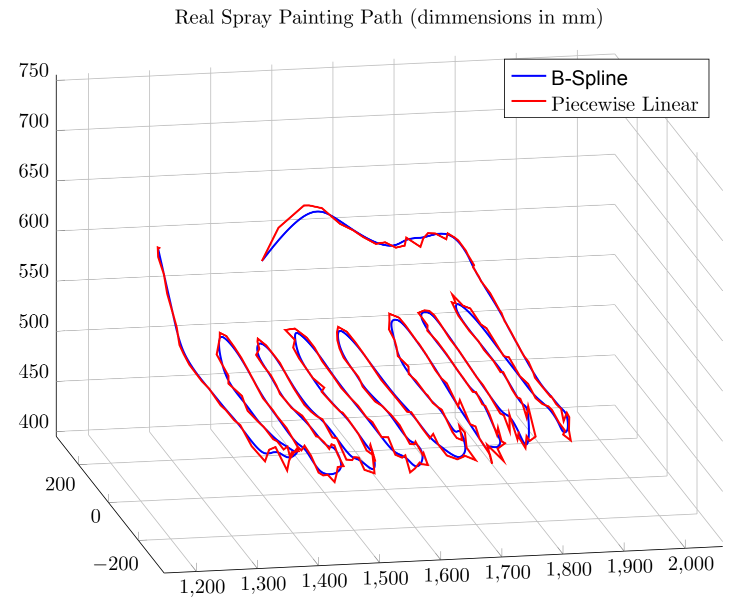
Figure 5.11 shows the resulting B-Spline for the spraying of the entire tray surface; on top, the (a) plot shows th B-Spline alone; on bottom, the (b) plot shows the B-Spline and the piecewise linear approximation.

A closer (zoomed) look at the B-Spline vs Piecewise linear approximation shows how the high frequency noise is eliminated particularly on U-turns. Figures 5.12,5.13 and 5.14 hold three different views to the spline and original curves.

As it was made reference in section 4.4.1.1, the curve is parametric thus not a $x, y, z$



**(a)**



**(b)**

**Figure 5.11:** (a) **Smooth B-Spline**: It goes smooth through the tracked positions and do not have sharp edges or discontinuities. The path is a coast-to-coast swing along the surface of the tray. (b) **B-Spline vs Piecewise Linear Approximation**: The same path as in (a) against the piecewise linear approximation. The smoothness facet is obvious as it is the noise reduction over the entire path.

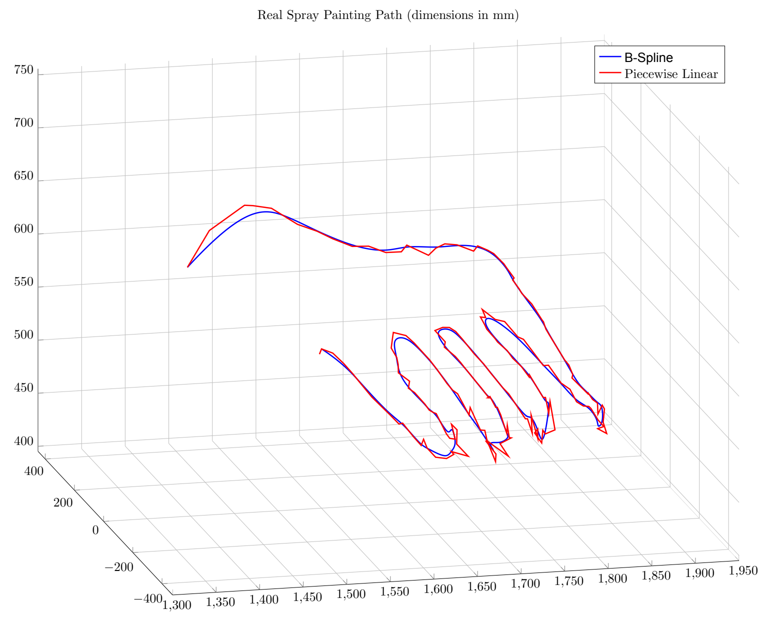


**Figure 5.12: B-Spline vs Piecewise Linear Side View -** Comparison between the B-Spline and the Piecewise Linear Approximation on a portion of the painting path

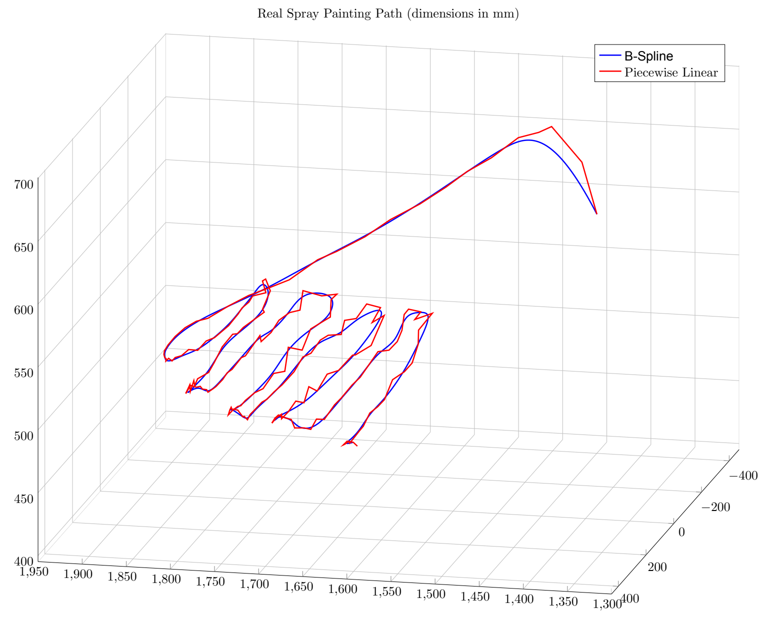


**Figure 5.13: B-Spline vs Piecewise Linear Opposite View -** Comparison between the B-Spline and the Piecewise Linear Approximation on a portion of the painting path — opposite view of Figure 5.12

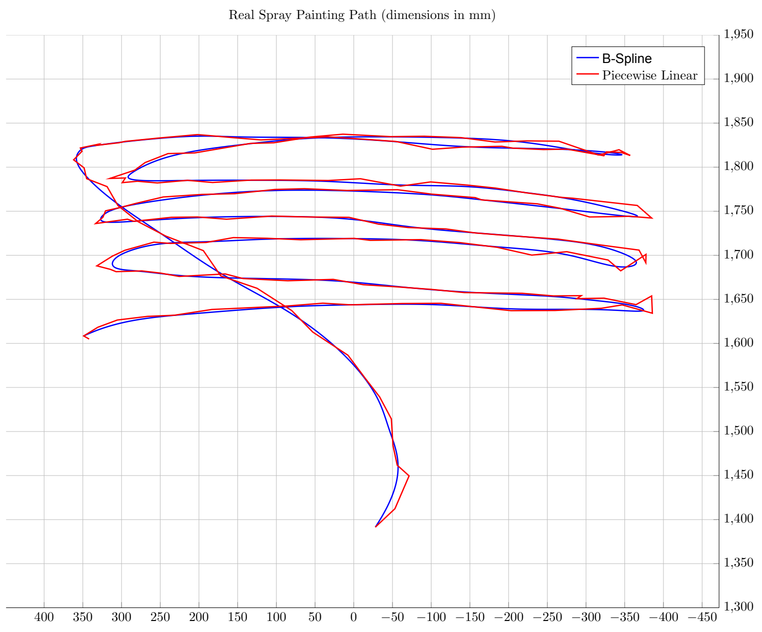


**Figure 5.14: B-Spline vs Piecewise Linear X-Y view -** Comparison between the B-Spline and the Piecewise Linear Approximation on a portion of the painting path — X-Y plane view

collection of coordinates. Using the chord length method — recall equation 4.16

$$d = \sum_{k=1}^{n} \|P_k - P_{k-1}\|_2$$
$$u_j = \frac{\|P_j - P_{j-1}\|_2}{d}$$

it is possible to know which B-Spline position is equivalent to the original points. Figure 5.15 shows a section of the path where the tracked positions are marked in red and the equivalent value of the B-Spline, $S(u_i)$ is marked in green. It is easy to conclude that the filtered points not always are the ones that are close to the tracked positions; this shows how the B-Spline looses track of space and how hard it is to numerically evaluate the result of the spline smoothing since there is no real point correspondence.

The tracking points are collected throw the cameras at constant rate: the camera's frame rate, hereafter designated as $F_t$. After the B-Spline reconstruction of the path it is necessary to sample the curve in order to have again a set of 3D points with which it is fed the robot controller. The main variables that influence the sampling rate of the spline are the final number of points — a high sampling rate implies large

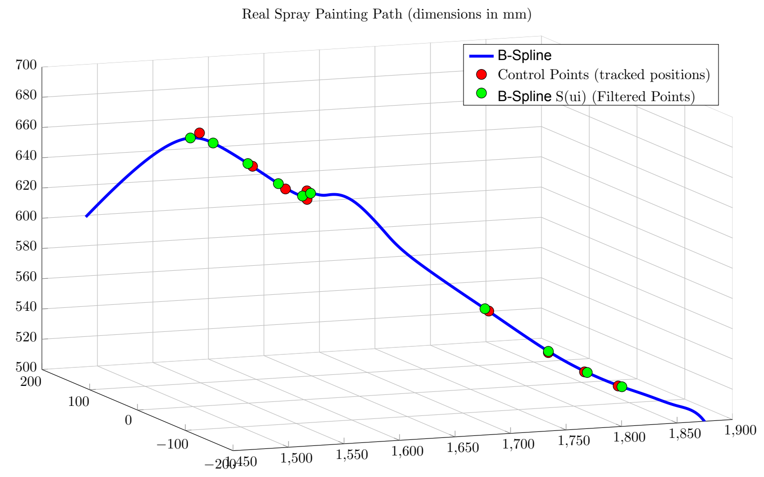


**Figure 5.15: B-Spline Evaluation -** Marked in red are the original tracked positions. The evaluation of the blue B-Spline is done over the parameter $u$ and not in cartesian space. As such, the correspondence between $S$ values of the spline and 3D coordinates is not known. Using the same parameter definition method as used in the B-Spline construction it is possible to have an equivalence. The spline points, i.e, the filtered positions corresponding to the tracked ones are marked in green.

robot programs, and the distance between points — as the sampling rate changes so does the distance between each point; these will be used in move-linear instructions so the fewer points the less roundness of the final path. Figures 5.16,5.17, 5.18 and 5.19 show the final shape of the path using a sampling frequency of $1{\times}F_t$, $2{\times}F_t$, $4{\times}F_t$ and $8{\times}F_t$ respectively. After $4{\times}F_t$ the smooth effect is already undistinguishable; there are enough points to perform a smooth path. Table 5.7 summarizes the data from these sampling rates: it shows the number of points of each sampling rate, the computational time required to evaluate the B-Spline, the total curve length and the average distance from one point to the next. The curve length is a measure of how long the path is, and can be compared to the piecewise linear approximation of the tracked points. It shows how much the original tracked path was shortened. To estimate the arc length one must find the integral of the B-Spline:

$$L = \int_{u=0}^{u=1} S\,(u)du$$

The solutions to these integrals are extremely difficult to find as it involves complex elliptic integrals. Alternatively, a chord length method can be used; that is, it is possible to divide the curve into small segments and compute its length as a linear path. The approximation becomes better as more divisions are considered. If the spline $S$ is sampled into $n$ points $D_i, i = 1, ..., n$ than the arc length is given by:

$$\int\limits_{u=0}^{u=1} S\left(u\right) du \simeq \sum_{i=2}^{n} \left\| D_i - D_{i-1} \right\|$$

Actually, since the robot is going to playback the path with linear interpolation, this measure is the best length estimation of the curve executed by the manipulator (the tolerance feature adds uncertainty). As can be seen from Table 5.7, increasing the sampling frequency more than four times has little to no effect of the final path length. This sampling rate already provides points $9mm$ close to each other to the linear approximation of the robot controller appears smooth.

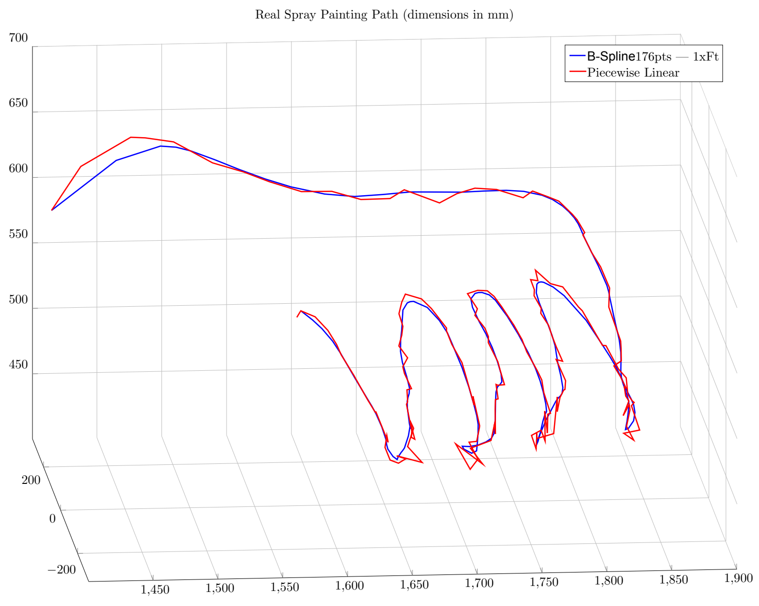


**Figure 5.16: Sampling of the B-Spline at the Same Frequency of the Tracking** - The plot shows the piecewise linear approximation and the B-Spline curve. The spline is drawn using as much points as were originally captured by the tracking system. The linear tracks are clearly visible

Finally, these last plots evaluate the path over time. Figure 5.20 plots the evolution of each path component with time, $x\left(t\right)$, $y\left(t\right)$ and $z\left(t\right)$. These plots alone give some feedback on the precision of human hand work. As depicted on the picture, the human

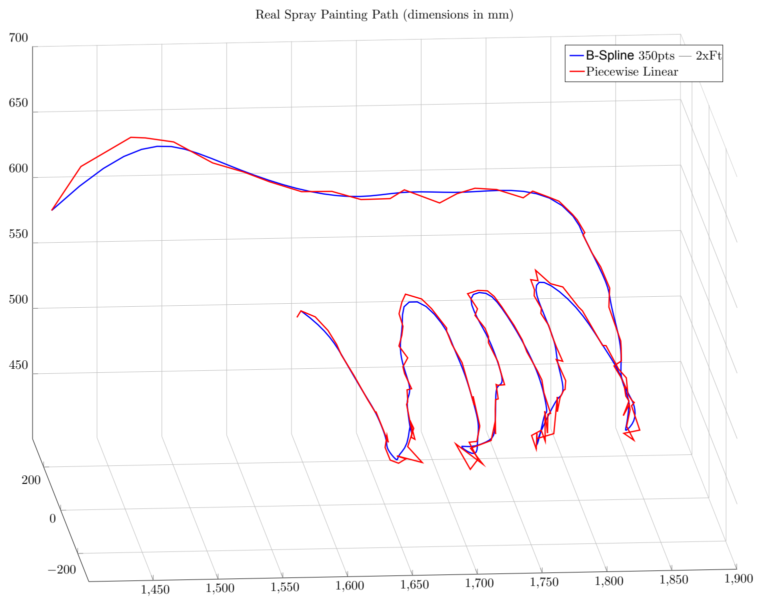


**Figure 5.17: Sampling of the B-Spline at 2 times the Tracking Frequency -** The plot shows the piecewise linear approximation and the B-Spline curve. The spline is drawn using twice as much points as were originally captured by the tracking system. The linear tracks are still visible.

**Table 5.7: Sampling of the B-Spline Curve**: This table shows the resulting number of points, their average spacing, the computational time and the final curve length for various sampling rates. The first entry is the original path, without spline smoothing. The next entries are for spline sampling at rates proportional to the tracking frequency.

| Sampling Rate | No Points | Chord Length (mm) | Avg. Point Spacing (mm) | Comp. Time (ms) |
|---|---|---|---|---|
| Piecewise Linear @ Ft | 1750 | 64580 | 37.0 | 0 |
| 1xFt | 1750 | 62590 | 35.8 | 40 |
| 2xFt | 3500 | 62830 | 18.0 | 64 |
| 4xFt | 7000 | 62870 | 9.0 | 115 |
| 8xFt | 14000 | 62890 | 4.5 | 215 |
| 16xFt | 28000 | 62890 | 2.2 | 426 |
| 32xFt | 56000 | 62890 | 1.1 | 839 |

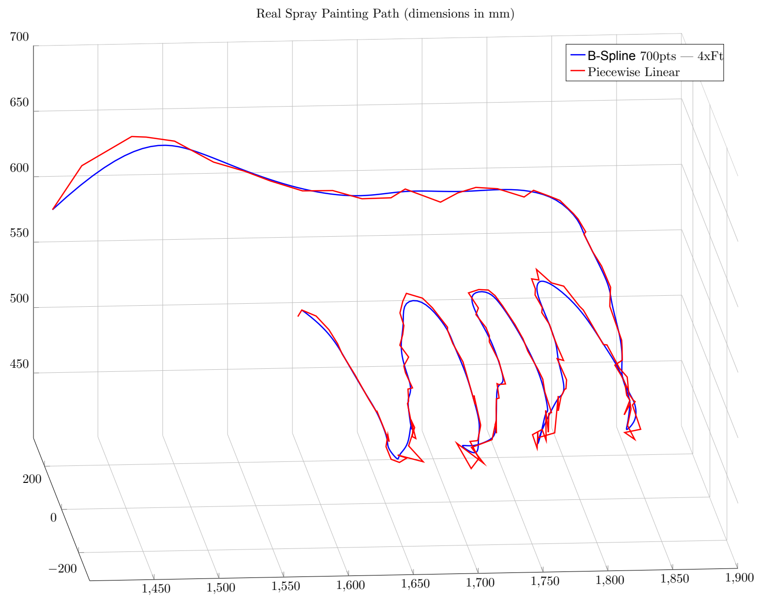


**Figure 5.18: Sampling of the B-Spline at 4 times the Tracking Frequency -** The plot shows the piecewise linear approximation and the B-Spline curve. The spline is drawn using 4 times as much points as were originally captured by the tracking system. The path appears smooth and the linear tracks are not visible.

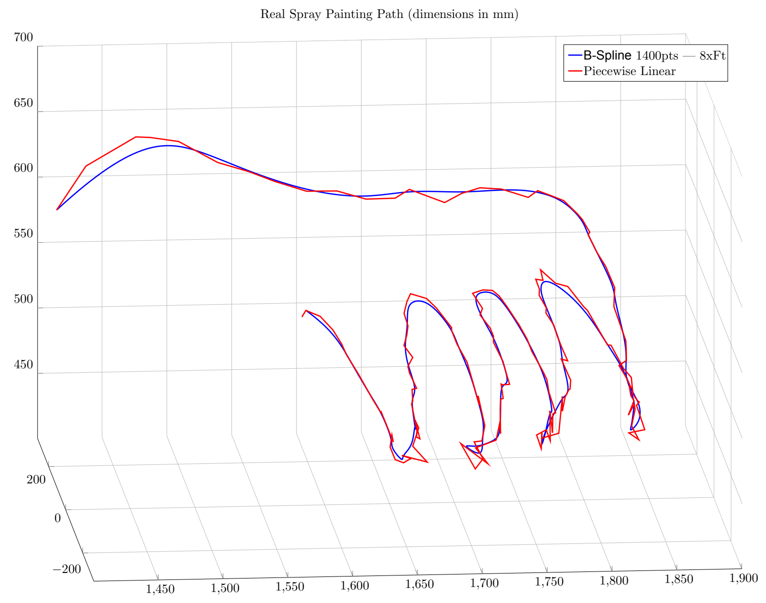


**Figure 5.19: Sampling of the B-Spline at 8 times the Tracking Frequency -** The plot shows the piecewise linear approximation and the B-Spline curve. The spline is drawn using 8 times as much points as were originally captured by the tracking system. Visually there is no improvement over the 4 times the tracking frequency, presented in Figure 5.18. Yet, it costs double its computational time and twice more points.

hand movement is quite precise: $x\,(t)$ shows how linearly the movement is going from the beginning until the end of the tray at constant velocity; $z\,(t)$ is the distance to the work piece, and the human operator can keep it almost constant over the entire path, i.e, the spray gun is kept at constant distance from the object ensuring the constant spraying over the surface; $y\,(t)$ is an almost perfect periodic movement of the side to side swinging over the surface. As a whole, this path clearly shows the expertise of the operator.

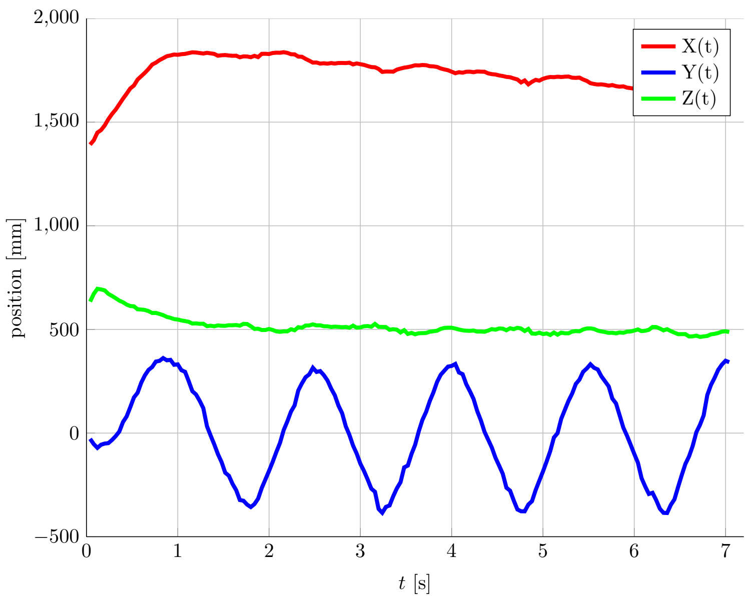


**Figure 5.20: Path decomposed into $X\,(t)$,$Y\,(t)$ and $Z\,(t)$** - The $x$,$y$ and $z$ components of the painting path plotted against time. It gives some insight on the precision of the human operator movements.

Figure 5.21 shows the same data adding the B-Spline to each curve. The spline and the original positions are almost indistinguishable; despite that, it has already been shown that the B-Spline is actually smoothing the path. This shows how in spite of smoothing through the points and not passing over them, the B-Spline still holds the essence of the movement, i.e, it does not change the pattern nor the speed of the original path; the real operator's know-how is kept after the path smoothing. Looking closer, as in Figure 5.22, the B-Spline is eliminating the noisy data while preserving the frequency and the geometry/shape of the demonstrated trajectory.

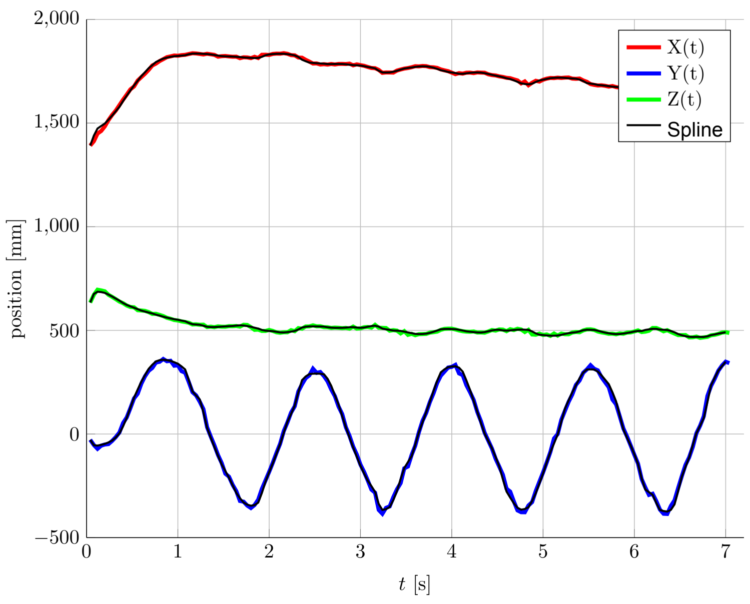


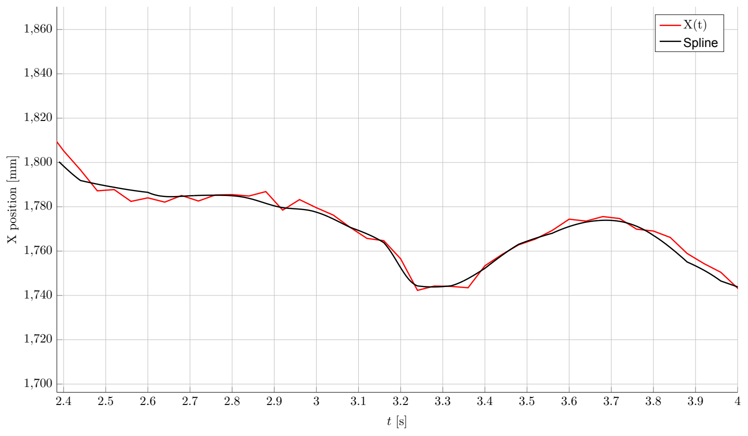
**Figure 5.21: B-Spline smoothing of $X(t)$,$Y(t)$ and $Z(t)$** - The B-Spline curve does not change the main shape of the trajectory; instead it cleans up noise and hand jitter. The resulting path is very similar to the original one.

### 5.2.5 Timings

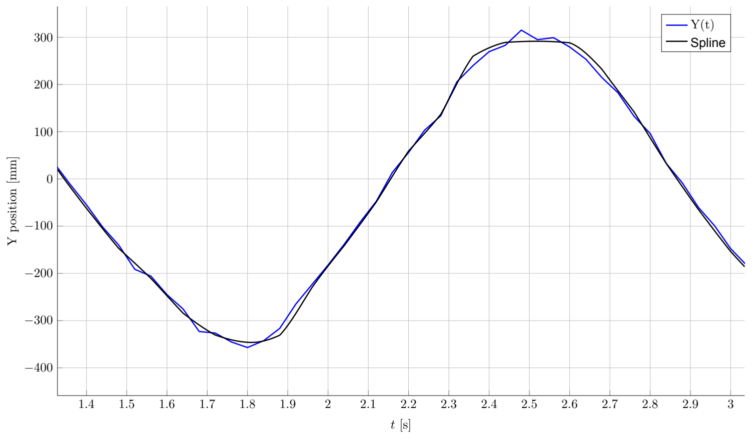
One of the most advantageous features of this motion mimic framework is to deliver an instant playback of the demonstrated movement. Even if the high level programming is achieved, it is important to reduce setup times. Also, a given demonstration may not be successful on the first try; if the movement mimic consumes too much post-processing time (i.e, the operator needs to wait an extended time to see the result) then the high level interface is still carrying expensive downtime and becomes unfeasable.

Table 5.8 presents the processing times for both the online and offline processes that make up the motion demonstration system. The online stages are those being executed as image frames arrive; it includes the image debayer (conversion of the bayer pattern into the RGB in some colour cameras), cluster detection and classification, matching clusters from different cameras and 3D retrieval, and finally pose estimation. The offline steps include the trajectory smoothing and the generation of the robot program.

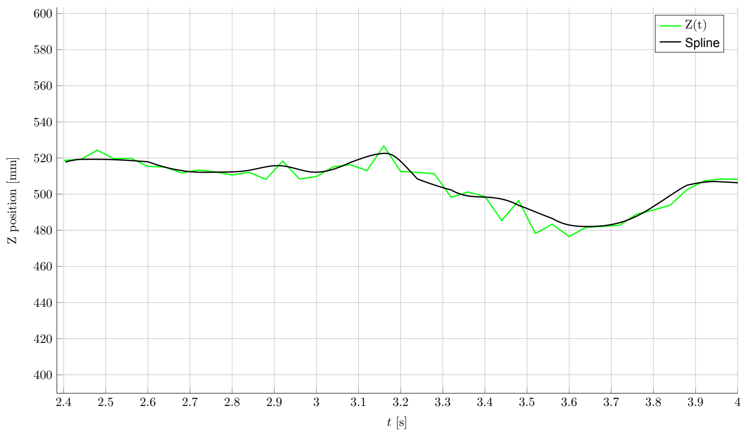
The tracking (all processes up to the marker pose estimation) is done in real-time which adds another powerful feature to the framework: the robot can be moved in real-



**(a)**



**(b)**



**(c)**

**Figure 5.22:** (a,b,c) **Zoom view of B-Spline with $X(t)$, $Y(t)$ and $Z(t)$:** the spline eliminates the high frequency noise while preserving the shape of the path.

time, for instance in an attempt of accessing reachability and reproducibility before the definitive demonstration, or for teleoperating the robot in some applications. The B-Spline smoothing and code generation take place after the demonstration is concluded but the processing time involved is far below 1 second which makes it perfectly unnoticeable to the user. Even the communication with the robot controller and uploading the program do not consume more than a couple of seconds. Therefore, the robot is ready to perform right after the demonstration has ended.

Moreover, looking at the online processing times it can be concluded that less than 15% of the available time is being used — 6ms at 25 frames per second means $6ms/40ms = 0.15$. This makes it possible to increase camera resolution or even the frame rate whilst maintaining the real-time capability.

**Table 5.8: Processing times for every stage of the motion demonstration**: The results are divided in online and offline: the former set is done in real-time as the images come in from the cameras; the later include the routines that need the whole trajectory and, as such, can only be executed when the demonstration is complete. The offline processes clearly dominate processing time yet they barely take more than one second. The tests were run on a core i7 @2.8GHz , under a Fedora 16 installation.

| | Routine | Time (ms) | |
|---|---|---|---|
| Online | Image Debayer | 2.2 | |
| Online | Cluster Detection & Classification | 2.6 | |
| Online | 2-Camera Cluster Matching and 3D Retrieval | 0.5 | |
| Online | Pose Estimation | 0.6 | |
| Total | | | < 6 |
| Offline | Filtering and Smoothing | 74.7 | |
| Offline | Robot Program Generation | 220 | |
| Offline | Communication and Upload | 1000 | |
| Total | | | < 1300 |

# Chapter 6

# Conclusions

## 6.1 Global Assessment and Conclusions

In the domain of industrial applications, this research begun with the two available workforces: on one side, the skill-full human operator; on the other, the productive robotic manipulator. In-between there are set of companies, SMEs, that mostly rely on the former to deliver quality, and eager for the other to have the edge in an increasingly competitive global economy. Technology-unaware operators and complexly-programmable machines fail to interface each other.

This thesis described a high level programming framework that ultimately delivers the possibility of transferring human know-how to the industrial manipulator. It proposes a new mean for human motion tracking based on an innovative marker together with a collection of routines that allows a real-time interface with the robot. And it does so while caring for the most crucial elements for a successful industrial implementation in low budget companies: the low cost, the simplicity (minimal changes required to working cells, minimal intrusion in the process, minimal human interface), readiness (instant output, real-time capability) and flexibility (possible different marker sizes, number of cameras, workspace volume, adjustable precision).

The goal of allowing an human to program an industrial manipulator with complete abstraction of programming concepts has been achieved. But in addiction, it happens while the operator is kept in his area of comfort: doing the everyday tasks, putting his skills on work. Instead of an human-robot interaction scheme where the operator needs to learn a new tool to intuitively command the robot, this framework also allows complete abstraction from the machine. That is, the operator does not need to perform any new action in order to have the robot programmed, he does not even need to be aware that the robot exists. His know-how is captured from the daily routine and transformed into a robot program that mimics the actions. The human is able to instruct the robot without knowing he is doing so.

From the start, this framework is optimized for industrial feasibility. The marker, besides being a cheap plastic-printed shape with basic electronics, performs at industrial grade with high detection rates, high immunity to environment conditions (particularly lighting and dust) and high flexibility (easily plugged to industrial tools, adaptable in size). Using industrial cameras is also another robustness and flexibility indicator: they are fit to harsh conditions by design; the wide range of commercial options makes a solution available to a wide range of applications — whether an expanded work volume or a fine accuracy is needed, one can easily choose proper lenses and camera resolution. Additionally, the light-based marker and the cameras form an advanced sensor-set that performs without contact and only requiring a clear line of sight. This makes installation easier and minimally intrusive to the process.

The proposed framework was installed in a Portuguese company, Flupol, which served as an industrial demonstrator and validator. The former human-operated coating cell has been robotized and serves now as a hybrid workstation where either painters or robot can operate. The manipulator painted several shapes, mostly baking trays and molds, starting with an human demonstration for one sample, and then having the robot resume production with the motion mimic. The validation and evaluation of performance is primarily done by human inspection of the finishing quality, but also by weighting the amount of dispensed taint and measuring the coating thickness. The results have been approved and, by the time of writing of this dissertation, Flupol has a fully functional robotized cell with a high level human-robot interface through motion imitation. The intricate human wrist movements have been successfully captured in its full six degrees of freedom, and through a series of filtering and smoothing techniques the robot is able to mimic the human actions. The painting know-how is instantaneously transferred from the painter (with over 25 years experience on the field) to the robot. This is accomplished without operators writing a single line of code, but also with the bare minimum software interface to start and stop the demonstration process.

## 6.2 Future Work

The proposed motion imitation framework is presently installed at Flupol and serves as a high level interface for a spray painting industrial robot. Nonetheless much work can be developed in order to improve the concept and the implementation.

The performance validation at Flupol asserts the industrial feasibility but spray painting is still a contact-less application. One of the most interesting improvements is to prepare the system for a full contact task. Eventually, this will lead to the integration of a sensor force in the robot control loop and, as such, integration of the sensor as an optional component in the motion demonstration framework. The research on this topic can significantly expand the areas of applications of the developed system.

On the performance level, it is important to study the actual impact of varying the number of cameras, changing the camera resolution and varying the marker size. The conclusions reported in the results chapter of this thesis provide means to predict how the system output changes with some of those variables. Increasing the resolution and adding another pair of cameras opposite to the first may effectively boost the playback accuracy. Yet, it is important to run tests and put these conclusions in numbers. It can help dimensioning new setups for different applications in matters of both performance and cost.

A natural evolution of the human-machine interface for this framework would be the minimization of software interfaces, for instance, by using voice commands to initialize and finish the demonstration process. There are already much work developed in this area such that the integration with these kind of solutions would be fairly simple and to a great profit.

This framework was conceived for an accurate mimic of human moves at the lowest cost. Comparison with off-the-shelf solutions should be done, in terms of price, accuracy, and industrial applicability.

Finally, the proposed architecture is supposed to lay the basis for a new method of robot programming through motion demonstration. From here, every kind of application specific optimizations can be built on top of it. For instance, spray painting applications can benefit with the offline analysis of the trajectory in order to reduce over-spray whenever possible, or to correct path segments to achieve an uniform taint thickness along the object surface.

# References

[1] Statistical Department International Federation of Robotics (IFR). **World Robotics 2012 (Executive Summary)**. http://www.worldrobotics.org/, 2012. 2, 3

[2] T. Beck, A. Demirguc-Kunt, and R. Levine. **SMEs, Growth, and Poverty**. Working Paper 11224, National Bureau of Economic Research, March 2005. 3

[3] Flupol. **Flupol - Surface Engineering**. 7, 101

[4] M. Ferreira, A. Paulo Moreira, and P. Neto. **A low-cost laser scanning solution for flexible robotic cells: spray coating**. *The International Journal of Advanced Manufacturing Technology*, **58**(9-12):1031–1041, 2012. 9

[5] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. **A survey of robot learning from demonstration**. *Robotics and Autonomous Systems*, **57**(5):469 – 483, 2009. 13

[6] G. Biggs and B. Macdonald. **A Survey of Robot Programming Systems**. In *in Proceedings of the Australasian Conference on Robotics and Automation, CSIRO*, page 27, 2003. 13

[7] G. Hirzinger, J. Bals, M. Otter, and J. Stelter. **The DLR-KUKA success story: robotics research improves industrial robots**. *Robotics Automation Magazine, IEEE*, **12**(3):16–23, 2005. 14

[8] R. Calcagno, F. Rusinà, F Deregibus, AS Vincentelli, and A Bonivento. **Application of wireless technologies in automotive production systems**. *VDI BERICHTE*, **1956**:57, 2006. 14

[9] L. Wang, Y. Tian, and T. Sawaragi. **Case-based automatic programming in robotic assembly production**. *Industrial Robot: An International Journal*, **38**(1):86–96, 2011. 16

[10] R. Bischoff, A. Kazi, and M. Seyfarth. **The MORPHA style guide for icon-based programming**. In *Robot and Human Interactive Communication, 2002. Proceedings. 11th IEEE International Workshop on*, pages 482–487. 16

[11] E. Freund and B. Luedemann-Ravit. **A system to automate the generation of program variants for industrial robot applications**. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, **2**, pages 1856–1861 vol.2. 16

[12] X. Li, O. A. Landsnes, H. Chen, Sudarshan M-V, T. A. Fuhlbrigge, and M.A. Rege. **Automatic Trajectory Generation for Robotic Painting Application**. In *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, pages 1–6, June. 16

[13] L. Qi, X. Yin, H. Wang, and L. Tao. **Virtual engineering: challenges and solutions for intuitive offline programming for industrial robot**. In *Robotics, Automation and Mechatronics, 2008 IEEE Conference on*, pages 12–17, Sept. 16

[14] X. Li and B. Zhang. **Toward general industrial robot cell calibration**. In *Robotics, Automation and Mechatronics (RAM), 2011 IEEE Conference on*, pages 137–142, Sept. 16

[15] H. Yu, J. Shan, and X. Zhu. **Off-lineprogramming and remote control for a palletizing robot**. In *Computer Science and Automation Engineering (CSAE), 2011 IEEE International Conference on*, **2**, pages 586–589, June. 17

[16] G. C. I. Lin, T.-F. Lu, and D. Zhang. **CAD-based intelligent robot vision system**. pages 156–167, 1995. 17

[17] G. C. I. Lin and T. Lu. **CAD-Based Intelligent Robot Workcell**, 1995. 17

[18] Heping C., Weihua S., N. Xi, M. Song, and Y. Chen. **Automated robot trajectory planning for spray painting of free-form surfaces in automotive manufacturing**. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, **1**, pages 450–455 vol.1, 2002. 17

[19] Heping C., Weihua S., N. Xi, M. Song, and Y. Chen. **Automated robot trajectory planning for spray painting of free-form surfaces in automotive manufacturing**. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, **1**, pages 450–455 vol.1, 2002. 17

[20] Heping C., Weihua S., N. Xi, Y. Chen, A. Roche, and J. Dahl. **A general framework for automatic CAD-guided tool planning for surface manufacturing**. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, **3**, pages 3504–3509 vol.3, 2003. 18

[21] Heping Chen and N. Xi. **Automated tool trajectory planning of industrial robots for painting composite surfaces**. *The International Journal of Advanced Manufacturing Technology*, **35**(7-8):680–696, 2008. 18

[22] Heping Chen, T. Fuhlbrigge, and Xiongzi Li. **Automated industrial robot path planning for spray painting process: A review**. In *Automation Science and Engineering, 2008. CASE 2008. IEEE International Conference on*, pages 522–527, 2008. 18

[23] T. Pulkkinen, T. Heikkila, M. Sallinen, S. Kivikunnas, and T. Salmi. **2D CAD based robot programming for processing metal profiles in short series manufacturing**. In *Control, Automation and Systems, 2008. ICCAS 2008. International Conference on*, pages 156–162, 2008. 18

[24] M. Soron and I. Kalaykov. **Generation of continuous tool paths based on CAD models for Friction Stir Welding in 3D**. In *Control Automation, 2007. MED '07. Mediterranean Conference on*, pages 1–5, 2007. 18

[25] J Norberto Pires, T. Godinho, and P. Ferreira. **CAD interface for automatic robot welding programming**. *Industrial Robot: An International Journal*, **31**(1):71–76, 2004. 18

[26] F. Nagata, T. Hase, Z. Haga, M. Omoto, and K. Watanabe. **CAD/CAM-based position/force controller for a mold polishing robot**. *Mechatronics*, **17**(4?5):207 – 216, 2007. 18

[27] P. Neto, N. Mendes, R. Arajo, J. N. Pires, and A. P. Moreira. **High-level robot programming based on CAD: Dealing with unpredictable environments**. *Industrial Robot*, **39**(3):294–303, 2012. Cited By (since 1996):1. 18

[28] P. Neto, J. Norberto Mendes, N.and Pires, and A.P. Moreira. **CAD-based robot programming: The role of Fuzzy-PI force control in unstructured environments**. In *Automation Science and Engineering (CASE), 2010 IEEE Conference on*, pages 362–367, 2010. 18

[29] S. Waldherr, R.Romero, and S. Thrun. **A Gesture Based Interface for Human-Robot Interaction**. *Autonomous Robots*, **9**:151–173, 2000. 19

[30] M. Strobel, J. Illmann, B. Kluge, and F. Marrone. **Using Spatial Context Knowledge in Gesture Recognition for Commanding a Domestic Service Robot**. In *In Proc. 11th IEEE Workshop on Robot and Human Interactive Communication*, pages 468–473, 2002. 19

[31] P. Kumar, J. Verma, and S. Prasad. **Hand Data Glove: A Wearable Real-Time Device for Human-Computer Interaction**. *Hand*, **43**, 2012. 19

[32] J. Norberto Pires. **Robot-by-voice: Experiments on commanding an industrial robot using the human voice**. *Industrial Robot, an International Journal*, **32**, 2005. 20

[33] R. Dillmann, O. Rogalla, M. Ehrenmann, R Zollner, and M. Bordegoni. **Learning robot behaviour and skills based on human demonstration and advice: the machine learning paradigm**. In *ROBOTICS RESEARCH-INTERNATIONAL SYMPOSIUM-*, **9**, pages 229–238, 2000. 20, 21

[34] R. Zollner, O. Rogalla, and R. Dillmann. **Integration of tactile sensors in a programming by demonstration system**. In *Robotics and Automation,*

*2001. Proceedings 2001 ICRA. IEEE International Conference on*, **3**, pages 2578–2583 vol.3, 2001. 20

[35] R. Zollner, O. Rogalla, R. Dillmann, and M. Zollner. **Understanding users intention: programming fine manipulation tasks by demonstration**. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, **2**, pages 1114–1119 vol.2, 2002. 20

[36] O. Rogalla, M. Ehrenmann, and R. Dillmann. **A sensor fusion approach for PbD**. In *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, **2**, pages 1040–1045 vol.2, 1998. 21

[37] A. Skoglund, B. Iliev, B. Kadmiry, and R. Palm. **Programming by Demonstration of Pick-and-Place Tasks for Industrial Manipulators using Task Primitives**. In *Computational Intelligence in Robotics and Automation, 2007. CIRA 2007. International Symposium on*, pages 368–373, 2007. 21

[38] Y. Maeda and Y. Moriyama. **View-based teaching/playback for industrial manipulators**. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4306–4311, 2011. 21

[39] J Norberto Pires, G. Veiga, and R. Araújo. **Programming-by-demonstration in the coworker scenario for SMEs**. *Industrial Robot: An International Journal*, **36**(1):73–83, 2009. 21

[40] M. Stoica, F. Sisak, and A. D. Morosan. **Reinforcement learning algorithm for industrial robot programming by demonstration**. In *Optimization of Electrical and Electronic Equipment (OPTIM), 2012 13th International Conference on*, pages 1517–1524, 2012. 22

[41] B. Hein and Heinz Worn. **Intuitive and model-based on-line programming of industrial robots: New input devices**. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 3064–3069, 2009. 22, 23, 52

[42] B. Hein, M. Hensel, and H. Wo?rn. **Intuitive and model-based on-line programming of industrial robots: A modular on-line programming environment**. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 3952–3957, 2008. 22

[43] M. Field, Z. Pan, D. Stirling, and F. Naghdy. **Human motion capture sensors and analysis in robotics**. *Industrial Robot*, **38**(2):163–171, 2011. 22

[44] A. Elgammal and Chan-Su Lee. **Tracking People on a Torus**. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **31**(3):520–538, 2009. 22, 52

[45] A.P. Shon, K. Grochow, and R. P N Rao. **Robotic imitation from human motion capture using Gaussian processes**. In *Humanoid Robots, 2005 5th IEEE-RAS International Conference on*, pages 129–134, 2005. 22

[46] N. Naksuk, C. S G Lee, and S. Rietdyk. **Whole-body human-to-humanoid motion transfer**. In *Humanoid Robots, 2005 5th IEEE-RAS International Conference on*, pages 104–109, 2005. 23, 52

[47] L. Sigal, A. Balan, and M.J. Black. **HumanEva: Synchronized Video and Motion Capture Dataset and Baseline Algorithm for Evaluation of Articulated Human Motion**. *International Journal of Computer Vision*, **87**(1-2):4–27, 2010. 23, 52

[48] P. Azad, T. Asfour, and R. Dillmann. **Robust real-time stereo-based markerless human motion capture**. In *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*, pages 700–707, 2008. 23

[49] P. Azad, A. Ude, T. Asfour, and R. Dillmann. **Stereo-based Markerless Human Motion Capture for Humanoid Robot Systems**. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3951–3956, 2007. 23

[50] A.G. Cutti, A. Giovanardi, L. Rocchi, and A. Davalli. **A simple test to assess the static and dynamic accuracy of an inertial sensors system for human movement analysis**. In *Engineering in Medicine and Biology Society, 2006. EMBS '06. 28th Annual International Conference of the IEEE*, pages 5912–5915, 2006. 23

[51] T. Yamamoto and T. Fujinami. **Hierarchical organization of the coordinative structure of the skill of clay kneading.** *Hum Mov Sci*, **27**(5):812–22, 2008. 23

[52] J. Moldenhauer, I. Boesnach, T. Beth, V. Wank, and K. Bos. **Analysis of Human Motion for Humanoid Robots**. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 311–316, 2005. 23

[53] A. Y. Benbasat and J. A. Paradiso. **An Inertial Measurement Framework for Gesture Recognition and Applications**. In *Revised Papers from the International Gesture Workshop on Gesture and Sign Languages in Human-Computer Interaction*, GW '01, pages 9–20, London, UK, UK, 2002. Springer-Verlag. 23

[54] Chien-Chou Lin. **A hierarchical path planning of manipulators using memetic algorithm**. In *Information and Automation, 2009. ICIA '09. International Conference on*, pages 746–750, 2009. 23

[55] J. Aleotti, S. Caselli, and G. Maccherozzi. **Trajectory reconstruction with NURBS curves for robot programming by demonstration**. In *Computational Intelligence in Robotics and Automation, 2005. CIRA 2005. Proceedings. 2005 IEEE International Symposium on*, pages 73–78, 2005. 24

[56] Joon-Young Kim, Dong-Hyeok Kim, and Sung-Rak Kim. **On-line minimum-time trajectory planning for industrial manipulators**. In *Control, Automation and Systems, 2007. ICCAS '07. International Conference on*, pages 36–40, 2007. 24

[57] W. Li, G. Liu, Y. Wang, J. Zhi, S. Ma, and T. Chen. **Optimized tracjectory planning algorithm for industrial robot**. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*, pages 2397–2400, 2012. 24

[58] G. Xu and Zhengyou Zhang. *Epipolar Geometry in Stereo, Motion, and Object Recognition: A Unified Approach.* Kluwer Academic Publishers, Norwell, MA, USA, 1996. 35, 38, 39

[59] R.I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, ISBN: 0521540518, second edition, 2004. 38, 39, 40

[60] P. Malheiros, P. Costa, and A. Paulo Moreira. **Robust 3D motion capture and object positioning system using light emitting markers synchronized with stereoscopic camera system**. *UPIN NPat.77/ Pat. 41, Int. Patent PCT/IB2009/007186.* 40

[61] R. Y. Tsai. **Radiometry**. chapter A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses, pages 221–244. Jones and Bartlett Publishers, Inc., USA, 1992. 48

[62] Z. Zhang. **A Flexible New Technique for Camera Calibration**. *IEEE Trans. Pattern Anal. Mach. Intell.*, **22**(11):1330–1334, November 2000. 48

[63] O. Faugeras. *Three-dimensional computer vision: a geometric viewpoint.* MIT Press, Cambridge, MA, USA, 1993. 48

[64] *Industrial Robotics: Programming, Simulation and Applications.* Free Open Access Book — InTechOpen. 50

[65] Peter Fixell. **Absolute Accuracy Marketing Presentation**. *ABB Automation Technologies AB.* 51

[66] G. Lukács, A. D. Marshall, and R. R. Martin. **Geometric least-squares fitting of spheres, cylinders, cones and tori**. Technical report, 1997. 67

[67] V. Pratt and P. Point. **Direct Least-Squares Fitting of Algebraic Surfaces**, 1987. 67

[68] S. Joon Ahn, W. Rauh, and H. Warnecke. **Least-squares orthogonal distances fitting of circle, sphere, ellipse, hyperbola, and parabola**. *Pattern Recognition*, **34**(12):2283 – 2303, 2001. 70

[69] W. Kabsch. **A solution for the best rotation to relate two sets of vectors**. *Acta Crystallographica Section A*, **32**(5):922–923, September 1976. 76

[70] J. Pan, L. Zhang, and D. Manocha. **Collision-free and smooth trajectory computation in cluttered environments**. *The International Journal of Robotics Research*, **31**(10):1155–1175, 2012. 80

[71] H. I. Krebs, N. Hogan, M. L. Aisen, and B. T. Volpe. **Robot-aided neurorehabilitation.** *IEEE Trans Rehabil Eng*, **6**(1):75–87, March 1998. 82

[72] L. Piegl and W. Tiller. *The NURBS book (2nd ed.).* Springer-Verlag New York, Inc., New York, NY, USA, 1997. 82

[73] G. Farin. *Curves and surfaces for computer aided geometric design (3rd ed.): a practical guide.* Academic Press Professional, Inc., San Diego, CA, USA, 1993. 82

[74] K. Shoemake. **Animating rotation with quaternion curves**. *SIGGRAPH Comput. Graph.*, **19**(3):245–254, July 1985. 87

[75] E. B. Dam, M. Koch, and M. Lillholm. **Quaternions, interpolation and animation**. Technical report, 1998. 87, 89, 90

[76] W. Boehm. **On cubics: A survey**. *Computer Graphics and Image Processing*, **19**(3):201 – 226, 1982. 89

[77] A. H. Barr, B. Currin, S. Gabriel, J. F. Hughes, and S. Design. **Smooth Interpolation of Orientations with Angular Velocity Constraints using Quaternions**, 1992. 89

[78] S. Chatterjee and A.S. Hadi. *Regression Analysis by Example.* Wiley Series in Probability and Statistics. Wiley, 2006. 93