On Sensor-Controlled Robotized One-off Manufacturing

Cederberg, Per

2004

On Sensor-Controlled Robotized One-off Manufacturing

Per Cederberg

Division of Robotics
Department of Mechanical Engineering
Lund Institute of Technology
Lund University , 2004

Organization Lund University Department of Mechanical Engineering P.O. Box 118, SE-221 00 Lund, Sweden Phone: +46-46-222 45 92 Fax: +46-46-222 45 29	Document Name PhD Thesis Date of Issue 2004-08-27 CODEN: LUTMDN/(TMMV-1058)/1-78/(2004)
Author(s) Per Cederberg	Sponsoring Organization(s) Vinnova (Komplexa Tekniska System)
Title and subtitle On Sensor-Controlled Robotized One-off Manufacturing	

A semi-automatic task oriented system structure has been developed and tested on an arc welding application. In normal industrial robot programming, the path is created and the process is based upon the decided path. Here a process-oriented method is proposed instead. It is natural to focus on the process, since the path is in reality a result of process needs. Another benefit of choosing process focus, is that it automatically leads us into task oriented thoughts, which in turn can be split in sub-tasks, one for each part of the process with similar process-characteristics. By carefully choosing and encapsulating the information needed to execute a sub-task, this component can be re-used whenever the actual subtask occurs.
By using virtual sensors and generic interfaces to robots and sensors, applications built upon the system design do not change between simulation and actual shop floor runs. The system allows a mix of real- and simulated components during simulation and run-time.

Keywords
robotics, sensor, control, simulation, task-oriented programming, real-time, world model, arc welding, one-off manufacturing

Classification system and/or Index term (if any)		
Supplementary bibliographical information		Language English
ISSN and key title		ISBN 91-628-6289-8
Recipient's notes	Number of pages 123	Price
	Security classification	

On Sensor-Controlled Robotized One-off Manufacturing

Per Cederberg

LUND UNIVERSITY

Akademisk avhandling som för avläggande av teknologie doktorsexamen vid tekniska fakulteten vid Universitetet i Lund kommer att offentligen försvaras i sal M:E, Tekniska Högskolan i Lund, onsdagen den 8 dec 2004, kl 10.15.

On Sensor-Controlled Robotized One-off Manufacturing

Per Cederberg

Division of Robotics
Department of Mechanical Engineering
Lund Institute of Technology
Lund University, P.O. Box 118, SE-221 00 Lund, Sweden

PhD Thesis
CODEN: LUTMDN/(TMMV-1058)/1-78/(2004)
ISBN 91-628-6289-8

Printed in Sweden
KFS i Lund AB, Lund

Summary

While robots today are cheaper, faster, more reliable and accurate than ever before, they are still mainly used to reiterate preprogrammed trajectories. These static robot programs are fairly efficient in handling high volume production processes, but fail to address problems facing small batch and one-off manufacturing-dependent systems.

Off-line programming has reduced the transition time between products in a manufacturing system, but requires accurate information about the physical work-cell. In practice, the precision needed is seldom possible to achieve, and costly adjustments have to be taken care of on the shop floor. Other ways to address the precision deficiencies, besides touch-up on the shop floor, are expensive clamping and design modification.

It is tempting to think that sensors solely should be able to bridge the gap between a manufacturing potential and the result accomplished. But advanced application processes imply more complex relationships; observable variables are not necessarily controllable and controllable variables are not necessarily those that define the task. Hence, in complex industrial operations, there are mapping issues in both directions between not only variables that are detected by sensors and controllable variables, but also between the task specification described in terms of how to reach productivity and quality measures and how to control the process to obtain such goals.

From the above, a comprehensive view yields the best understanding of the problem. Knowledge of the environment is important in robotics automation. The use of advanced sensors may yield the competitive edge in robotized small batch and one-off production systems. However, sensors increase the system complexity. To avoid, for instance, singularities and collisions, decisions have to be taken at a system level, during run-time or even before a process is started. This collection of system knowledge has been given a name – the *world model*.

The world model may quite easily be appreciated as a general concept but is much harder to implement in reality. It depends on process-specific conditions and equipment limitations. Computer modeling of sensor behavior is difficult. Manufacturers of robots, sensors, robot simulation systems, etc., are as eager to sell the equipment as they are reluctant to reveal the “business secrets” inside. In the perfect world, a manufacturer of sensors should be able to provide a black box, a piece of software that when used in simulations rendered similar results as the real sensor. Robot simulations are for instance more accurate today than ten years ago much because of the introduction of a corresponding technique, RRS – Realistic Robot Simulation.

Despite static off-line programming problems, *only one* robot program exists after simulation and can be down-loaded to the robot, and we feel quite assured that the robot will carry out an identical sequence of instructions each time the robot program executes. On the contrary, if simulating and executing dynamic programs that includes sensors, it would also be preferable to be sure that certain situations do *not* occur, and when we feel assured, we do not wish to be uncertain about what to expect later on the shop floor. This situation is quite different compared to the former, and it seems obvious that a simulation and run-time environment for sensor-driven robots must, besides high quality simulated sensors, also incorporate a realistic work-cell simulation tool with, for instance, collision detection.

Even if we succeed to copy robots and sensors well, we still need to decide the context our world model should grasp and at what abstraction level people will interact with the system. Robot motion is usually seen as series of continuous moves, but from a human point of view, the process is rather understood as a sequence of discrete steps, tasks and sub-tasks. Thus, on a human level, it is favorable to be able to work with components on process levels that encapsulate the logic needed to drive the robot and its sensors in the work-cell.

The thesis includes a task-oriented structure that describes how to organize simulated and real components in development and deployment of sensor-controlled applications. Realistic sensor simulation includes management of sensor deficiencies. Likewise, it involves sensor APIs that do not reveal whether the sensor utilized is modeled or real. The proposed structure also includes means to organize an application for allowing users to extend the world model with respect to process knowledge by applying tasks and sub-tasks as reusable components in an object-oriented way.

As a test and implementation case, the thesis describes an arc welding application process where sensors have been utilized: a laser seam tracker, a distance-sensor and stereo cameras. The technique is by no means limited to these sensors, but

they serve to illustrate difficulties that can be handled by the proposed structure. Process parameters studied include collision tests and singularity detection and avoidance. Particular arc welding process issues have not been under study.

By the development and organization of simulated and real components in the proposed structure, the potential of applications that includes sensor-controlled industrial robots have been studied. Most components are developed in-house by the author and could be used on any operating system with minor effort. The Robot Simulation Application is used as a server for graphical feedback and collision tests, and as a tool to create the nominal work-cell in. In this context, the term nominal refers to known knowledge of the work-cell's frame dependencies before any input from sensors ("how we think the world looks like"). The application and the developed library for high-level robot motion, RLib, manages kinematical relationships, trajectory creation, error handling, etc.

Sensor-driven applications need to be able to create trajectories in run-time. The included "feeder", which allows the developed experimental platform to continuously create trajectories and to send joint values to the unmodified industrial robot used, has provided the means to show the industrial potential of the platform (given the significant limitations imposed by the robot manufacturer). Still, an Open Control System[1] would yield a more appealing solution.

The organization of objects and the level chosen for the sub-tasks they encapsulate are essential. Their initialization parameters define the reusability in different contexts. The level suggested was chosen considered that the example application was created as a raw C program. The objects define process-specific subtasks and handle a certain set of sensors. By use of an object-initialization GUI, a higher granularity of collaborating objects, or as an alternative, more customizable objects would probably yield greater reusability without adding extra burden on the user. It is also important that these objects have clear responsibilities in their relation to the underlying motion control and to other objects.

[1]Many robot control systems support some type of user IOs connected to local networks or buses. An Open Control System would for instance allow an application to read and write the robot pose with short and predictable latency, something that would make the application more robust.

Preface

Acknowledgements

First of all I would like to thank Professor Gunnar Bolmsjö for his contributions to the research and the guidance throughout the work. I would also like to express my gratitude to other colleagues at Division of Robotics, especially Dr. Magnus Olsson who I have had the pleasure to share ideas and a lot of research hours with, Dr. Mikael Fridenfalk who has been a great supporter and discussion partner, Dr. Stefan Adolfsson who has contributed to the experimental model and Dr. Giorgos Nikoleris with whom I have had interesting discussions in regard to seam tracker calibration methods.

I have also appreciated the cooperation with the Department of Automatic Control and Department of Computer Science at Lund University. I would like to thank Professor Rolf Johansson, Dr. Klas Nilsson, Dr. Anders Robertsson, Techn. Lic. Mathias Haage and Techn. Lic. Tomas Olsson for their generous help and interest in my work. I am also grateful to Professor Roland Pusch for his valuable comments respecting the content and the language, Dr. Lars Christer Böiers, Department of Mathematics, for guidance on issues related to calibration and, finally, to Dr. Jacek Malec, Department of Computer Science, for advice in mobile robotics. Financial support from Vinnova "Complex Technological Systems" is gratefully acknowledged.

Research work is often stimulating and energizing but sometimes implies isolation and frustration. I therefore wish to express a special gratitude to my dear friends, Richard Weston and Arne Ingemansson, for their concern and attention through good and hard times.

Finally, my ♡ belongs to my family, especially my beloved wife Meta and I thank her for just being here for me.

Per Cederberg, Lund, 2004

Papers

The thesis is based on the following papers:

A. Cederberg, P., Olsson, M. and Bolmsjö, G. (1999), A Generic Sensor Interface in Robot Simulation and Control, *in* 'Proceedings of Scandinavian Symposium on Robotics 99', Oulu, Finland, pp. 221–230.

B. Cederberg, P., Olsson, M. and Bolmsjö, G. (2002*a*), Remote control of a standard ABB robot system in real time using the Robot Application Protocol (RAP), *in* 'Proceedings of the International Symposium on Robotics, ISR2002', IFR, Stockholm.

C. Cederberg, P., Olsson, M. and Bolmsjö, G. (2002*b*), 'Virtual triangulation sensor development, behavior simulation and CAR integration applied to robotic arc-welding', *Journal of Intelligent and Robotic Systems* **35**(4), 365–379.

D. Cederberg, P., Olsson, M. and Bolmsjö, G. (2004), 'A semiautomatic task oriented programming system for sensor-controlled robotised small batch and one-off manufacturing'. Submitted to Robotica at the time of printing of this thesis.

E. Olsson, M., Cederberg, P. and Bolmsjö, G. (1999*a*), Integrated system for simulation and real-time execution of industrial robot tasks, *in* 'Proceedings of Scandinavian Symposium on Robotics 99', Oulu, Finland, pp. 201–210.

F. Olsson, M., Cederberg, P. and Bolmsjö, G. (1999*b*), Tele-Robotics for Sensor Driven Industrial Robot Tasks, *in* 'Proceedings of Deneb User Conference 99', Troy, MI, USA.

G. Olsson, M., Cederberg, P. and Bolmsjö, G. (2002), Integration of Simulation and Execution in Industrial Robot Systems, *in* 'Proceedings of the International Symposium on Robotics, ISR2002', IFR, Stockholm. paper No. 112.

H. Bolmsjö, G., Olsson, M. and Cederberg, P. (2002), 'Robotic Arc Welding - Trends and Developments for Higher Autonomy', *Industrial Robot* **29**(2), 98–104.

I. Johansson, R., Robertsson, A., Nilsson, K., Brogardh, T., Cederberg, P., Olsson, M., Olsson, T. and Bolmsjö, G. (2004), 'Sensor Integration in Task-Level Programming and Industrial Robotic Task Execution Control', *Industrial Robot* **31**(3), 95–102.

J. Blomdell, A., Bolmsjö, G., Brogårdh, T., Cederberg, P., Isaksson, M., Johansson, R., Haage, M., Nilsson, K., Olsson, M., Olsson, T., Robertsson, A. and Wang, J. J. (2004), 'Extending an industrial robot controller with a fast open sensor interface – implementation and applications'. Accepted for publication in Robots and Automation.

Contents

Chapter 1

Introduction

Industrial robot utilization of today is optimized for large batch manufacturing. Large batches are needed since high costs caused by, for instance, robot programming, touch-up on the shop floor and clamping, largely prevent industrial robots from being used in small batch or one-off manufacturing.

1.1 Research Problem

As mass production of large-lot items moves to less labor-expensive countries, it is believed that if one is able to successfully introduce industrial robots in small batch manufacturing and in one-off manufacturing, the increased automation will yield the edge to stay competitive in a changing market.

By introducing advanced sensors, these costs are assumed to be reduced. Today's programming technique, using off-line systems that create static programs, which are downloaded to the robots, assumes that advanced sensors such as cameras and seam trackers are not used, since no sensor feedback is provided except in local loops.

1.2 Research Objective

The presented objectives below have the overall objective to develop a concept for utilization of sensor-controlled industrial robots for one-off manufacturing. Scope and limitations of the thesis are specified in Chapter 7.

Research objective 1 *Sensor modeling, simulation and integration*

Develop methods for sensor simulation and integration in simulation and execution environments.

The concept of (modeling of) virtual sensors and generic sensor interfaces is presented and is applied to arc welding. A virtual arc welding sensor and its generic interface to graphical environments and control units have been developed.

Research objective 2 *Simulation and execution of sensor-guided robots*

It is apparent that realistic sensor simulation and run-time execution of sensor-guided robots cannot be performed using today's mechanisms, where no information is fed back to the model in which the robot program was created. To avoid the problem of differences between simulation and execution of applications that include sensor-guided robots, find a structure that allows a single source code and a transparent transfer between simulation and execution.

A structure parameterizing on reusable objects representing sub-tasks, and the underlying run-time system is presented. It permits execution of high-level control on a single nominal and predictable model. Generic robot and sensor interfaces developed, which allow an application to transparently simulate and execute experimental welding operations, are utilized.

1.3 Outline of thesis

This Chapter has given some introductory perspective to the research area and formulated the problem and the objective of the thesis. Chapters 3-5 give the necessary background to the main part of the thesis, Chapters 6 and 7. Besides this Chapter the thesis outline is as follows.

Chapter 2 starts with the methodology behind the thesis and follows up with some words about the scientific method used and a summary of the modeling tools.

Chapter 3 gives an introduction to welding, the chosen experimental case area of the thesis. Welding is complicated and when robotized and performed with the use of sensors, it is even more complex and the Chapter creates an understanding to the problems.

In Chapter 4, robot programming is discussed, both from an industrial robotics view and from the viewpoint of research. Examples of non-commercial robot programming libraries and environments are also given.

The high-level control Chapter, No. 5, comprises reasons for and against Artificial Intelligence, general sensor concepts and their relation to industrial robotics. High-level control is still an important research area, and examples of developed applications are given.

The main part of the thesis is described in Chapters 6 and 7. Chapter 6 is important as it gives a background to the choice of research area and identifies small batch manufacturing and one-off manufacturing as an important industrial focus for the years to come. Chapter 6 also presents the conceptual ideas of the work presented in Chapter 7.

The thesis contribution is described in Chapter 7. Scope and limitations of the thesis are specified. Then, it continues with the philosophy behind the research work and describes a model for sensor-controlled robots applied to arc welding. Some parts of the implementation are discussed, but the main focus is on the methodology.

In the experimental system structure Chapter, i.e. No 8, the proposed system of real and computerized components built to enable a task-oriented application to operate is described.

In Chapter 9, the experiments are depicted. They show how the system handles process-related events during run-time in a system where real and simulated objects (robots, sensors, workpieces) are transparently interchangeable.

A wider discussion of the implications by the thesis is taken in Chapter 10 where after conclusions follow in Chapter 11.

Chapter 2

Materials and Methods

A seaman meets a pirate in a bar, and they take turns to tell their adventures on the seas.
The seaman notes that the pirate has a peg leg, hook, and an eye patch.
Curious, the seaman asks "So, how did you end up with the peg-leg?"
The pirate replies "I was swept overboard into a school of sharks.
Just as my men were pulling me out, a shark bit my leg off".
"Wow!" said the seaman. "What about the hook"?
"Well...", replied the pirate, "We were boarding an enemy ship
and were battling the other sailors with swords.
One of the enemy cut my hand clean off."
"Incredible!" remarked the seaman. "How did you get the eye patch"?
"A seagull dropping fell right in my eye", replied the pirate.
"You lost your eye to a seagull dropping?" the sailor asked.
"Well..." said the pirate, "That was my first day with the hook."

2.1 Introduction

Laboratory experiments seldom surprise the researcher by functioning immediately; it may not be easy to receive data from some equipments for different reasons: the manual is not up-to-date and does not explain the preferred method of how to communicate, or the often cryptic communication interface may have been invented at a time with low bandwidth. Sometimes, the equipment does not work at all which may only be discovered after tedious test work. This was the also the case during the experiments in this thesis. The sensor used did not seem to accept any data requests sent to it at all. It was noticed rather early but for different reasons it took a while before it was decided to ship the 50 kg sensor control unit overseas for service.

Time constraints made it necessary to test the performance of the various components although the actual hardware was missing. This resulted in the development of a tracker control simulator with a GUI and two seam tracker interfaces, one simple and dedicated to the current task, and one underlying and more complete. The interfaces were generic in the sense that the sensor and other system components did not have to change for using the hardware after simulation. This matter plays a key role in the thesis.

2.2 Methodology

The methodology is based on earlier research and reflections on how to create a structure that benefits most from advantages of sensors used in small batch and one-off manufacturing cases while minimizing possible drawbacks. A number of systems exists for off-line programming purposes, but have been found unsuitable for on-line execution of sensor-driven robots for different reasons. An important reason is the lack of an environment that allows both development and deployment of sensor-driven processes. A run-time environment needs, in practice, robots with Open Control Systems (OCS). OCS is important, because decisions based on (global) sensor information can be taken and trajectories can be produced in real-time and executed by the low-level robot controller.

An experimental platform has been developed, which is interfaced to a commercial robot system. Because of the limited openness of commercial robot systems, the platform developed also has had to cope with significant latencies. By the choice of a slow[1] process – arc welding – as the example application, it has been possible to conduct experiments. It includes a library, RLib, for high-level robot motion control in soft real-time. Furthermore, two virtual sensors, which both are soft replicas of real industrial sensors with similar characteristics, and an application built on object-oriented principles, have been developed. The platform also includes a "feeder" that permits joint values to be sent to a commercial robot in real-time and allows high-level control during execution.

The application defines a task as a number of instances of sub-tasks that can be re-used in related contexts. A commercial robot simulation system is used to graphically perform any robot motion induced by the application and produced by the library. Any component, robot or sensor in the system, can either be simulated or real. Moreover, a mix of these components may coexist during execution. This is manageable by the use of a world model that keeps virtual and real components synchronized. The platform is not dependent on a particular simulation system and any system that is capable of providing means to respond to requests via the experimental platform's API will do well.

[1]Slow considering the travel speed of the robot.

The experimental system would, with minor modifications, be usable in an industrial test case. With support from OCS, the precision would increase drastically, thereby widening the industrial opportunities.

2.3 Scientific Method

Scientific work methods are the processes by which scientists create a reliable and consistent representation of the world. The use of procedures[2] aims to minimize influences of cultural and personal beliefs in our perceptions and interpretations. These methods can assist in getting sufficient as well as correct and structured information compiled and expressed in an understandable arrangement (Patel and Tebelius, 1987).

2.4 System Modeling Hardware

Most system development and evaluation has been done on SGI[3] workstations running the IRIX operating system. The M-Spot laser scanner, the physical tracker and two Unibrain Fire-i-400 cameras each equipped with a 3.5 mm lens are equipped with setup and GUI software that only run on DOS/Windows-based PCs. All other software is developed in-house, and the chosen language, C, has made it possible to compile and run it at home on an Apple Workstation equipped with the MacOSX operating system.

The robot used is a standard ABB IRB2400/16 with the S4C+ robot controller.

2.5 System Modeling Software

During experiments, IGRIP from DELMIA[4] has been used. IGRIP is a commercial 3D Robot Simulation Application (RSA[5]) that includes tools for design,

[2]However, good procedures mainly create a foundation for the research work when the idea is already there. If we know what we do not know, a good research procedure may be of great help. Insights in the subject of research that is gained through years of experience increases research efficiency but do also limit the creativity. What is of more philosophical interest is how new ideas originate. How do we find out the things we do not know that we do not know? It is the author's belief that chance and pure co-incidents are probably the main factors of real invention.

[3]Formerly Silicon Graphics, Inc. http://www.sgi.com

[4]Digital Enterprise Lean Manufacturing Interactive Applications. http://www.delmia.com

[5]RSA is not an established abbreviation but will be used in this thesis.

evaluation and off-line programming of robotics work-cells. It has been extended with custom logic in C using IGRIP's shared library technology.

It is only a minor part of the system that is developed as an extension to IGRIP. Most of the system code, underlying routines to support the application development, and the example application itself have been programmed in separate C programs. Some implementations are in JAVA. Chapter 7 describes the supporting code in more detail.

The running program on the robot controller is developed in the ABB RAPID language. Communicating software between the system code and the robot has also been developed in C and has followed the ABB RAP API and the Sun Microsystems[6] RPC API.

[6]http://www.sun.com

Chapter 3

Welding

3.1 Introduction

Welding can, according to Merriam-Webster, be defined as “to unite (metallic parts) by heating and allowing the metals to flow together” (*Merriam-Webster’s Collegiate Dictionary, 11th Edition*, 2003). Welding can be performed by one of the two major welding processes – spot welding and arc welding. Here, only arc welding will be discussed. In welding, the base materials are joined by having their respective abutting faces melted. In some welding processes, mainly during laser welding and TIG welding, only the base material melts to form the weld. In other processes a filler material is added to the joint.

3.2 Joint types

There are basically two major joint types – fillet weld and butt weld, see Figure 3.1. Fillet welds are triangular in cross section and form between two surfaces that are not in the same plane. The weld metal is placed alongside the two surfaces. Butt welds are applied between two components, normally in the same plane. The weld metal creates continuity between the components. Other weld types, for instance corner weld, lap weld and edge weld can be considered as special variations of these two types (Chester, 2004; AWS, 1976).

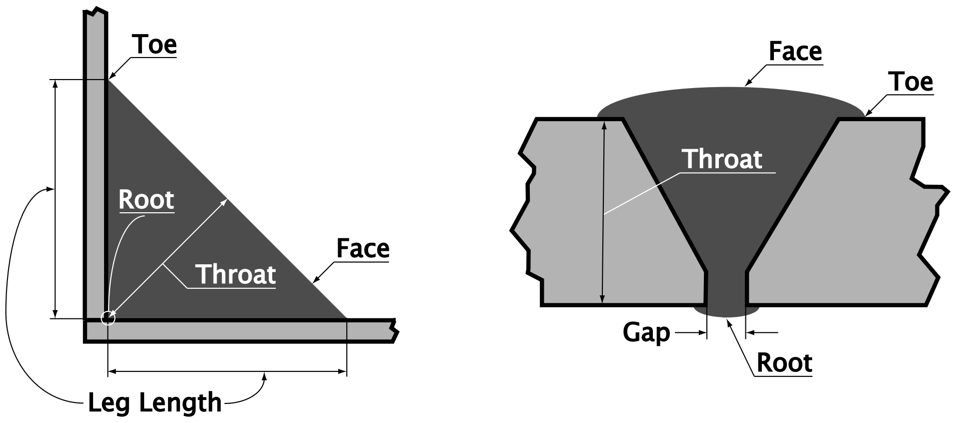


Figure 3.1: *The two major joint types – fillet weld (left) and butt weld.*

3.3 Weld process variables

Welding may look uncomplicated but is really a sophisticated process. The two weld types mentioned must be treated differently dependent on shape, size and a number of other process variables, for instance: gaps between the plates, travel speeds, electrode selection, electrode angles, and thermally induced strain. Furthermore, different techniques must be used if the weld is produced horizontally, vertically or overhead (AWS, 1976).

3.4 Weld techniques

Two different welding techniques are applied based on how wide weld bead one want to achieve: straight welding and weaving. In straight welding, the size of the weld pool can be adjusted by selecting a proper weld speed and size of the electrode. If a wider bead is needed, several passes of straight weld beads can be used. A wider bead can also be achieved with a smaller electrode by laterally moving the tip across the weld puddle. The weaving is normally limited to a movement twice the diameter of the electrode. Weaving techniques, which differ in the shape, exist as well (AWS, 1976).

3.5 Weld errors

The goal is of course to achieve a “perfect” weld, but there are many pitfalls. *Cracks* can occur in both the base metal and the weld metal as a result of welding. Therefore, the composition of base and weld metals is important. Too high current produces excessive concavity. A too large electrode, too slow travel speed or too low current will create slag entrapment on the root of the weld. Both concavity and slag entrapment will reduce the throat thickness and may lead to cracking of a shrinking weld and a restrained joint.

Besides cracks, *undercut* is considered a severe defect. Undercut is a term for a sharp narrow groove along the toe of the weld due to the scouring action of the arc removing the metal and not replacing it with weld metal, as seen in Figure 3.2. It reduces the cross sectional area and provides a notch into the heat-affected area of the joint, raising the stress and acting as a nucleus of crack initiation and possibly causing fatigue failure. It is often caused by incorrect electrode angles, incorrect weaving technique, excessive current and too fast travel speed. If, on the other hand, too much molten metal is flowing within the joint area without sufficient direct arc action on the base metal beneath, another defect, *lack of fusion*, occurs, see Figure 3.2.

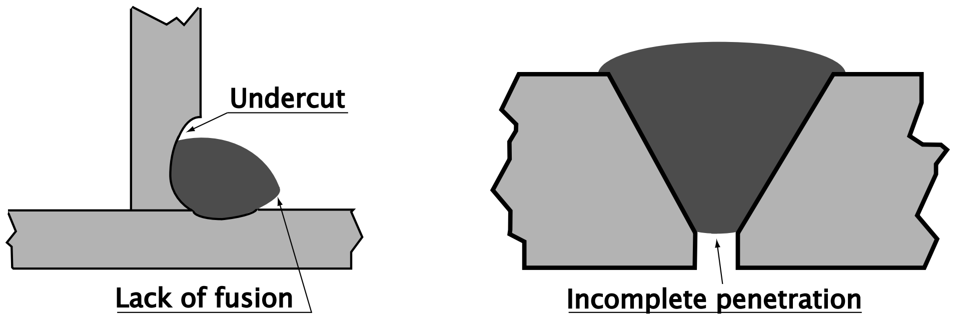


Figure 3.2: *Common weld defects: undercut, lack of fusion and incomplete penetration.*

Slag inclusions may occur if penetration of the weld is incomplete and there is lack of fusion. Insufficient cleaning of slag along an undercut toe of a multi-pass weld and incorrect electrode manipulation can also cause slag inclusions. Other reasons are excessive weaving and the use of a too large electrode diameter in a narrow groove, or too low amperage. Slag inclusions reduce the cross sectional area strength and may serve as an initiation point for serious cracking, particularly in harder steels (Chester, 2004; AWS, 1976).

If incorrect current and size or type of electrode is chosen or if the electrode is poorly manipulated, an *incorrect profile* may be the result. Besides appearance deficiencies, the overall strength of the joint is also affected by excessive concavity or convexity.

Incomplete penetration occurs when the weld does not penetrate to the root, resulting in insufficient throat thickness and reduced joint strength, see Figure 3.2. Insufficient root gap, too large electrode, too low current or incorrect electrode angle all contribute to such imperfection.

Finally, *porosity* may arise as a result of coating breakdown due to excessive current or moisture pickup by the electrode and absorption of impurities from the base metal.

Besides these weld errors, the expansion caused by the heat during welding and the subsequent contraction as the metal cools down may result in distortion from the original or expected shape. It is therefore important in welding to carefully plan the placements and order of welds, preheats, joint preparations, use of jigs, etc (Chester, 2004; AWS, 1976).

3.6 Weld management

Besides avoiding weld errors it is even more important to live up to specifications. These are normally defined in Weld Procedure Specifications, WPSs, and in quality control records, NDEs. WPS is an instruction sheet that gives details of how the weld is to be performed. Its purpose is to aid planning and quality control of the welding operation. A WPS may cover a range of thicknesses, diameters and materials and is required for demonstration of the ability to produce welds with required mechanical and metallurgical properties. Examples of Non Destructive Examination methods are: Eddy current testing; Ultrasonic leak detection; X-ray diffraction and Visual inspection. Finally, it is also necessary to demonstrate that the welders have the required knowledge and skill to do a sound work. Certain weld tests exist to assure this, see (Dyson, 2004) and (Boving, 1989) for more information.

3.7 Manual welding

Manual welding is still an important production process. Many areas still exist where welding robots cannot provide the flexibility of human workers. This applies especially to workpieces that are too large to be handled by standard welding robots. Another problem is geometry changes due to heat distortion during the welding process especially in case of multilayer welding, something that a manual welder (but not an automated robotized system) may be able to compensate for. Because of these reasons many advantages of welding robots cannot be realized (Helms, Schraft and Hagele, 2002).

A human worker is able to start welding with no further instruction or teaching, while the teaching of a welding robot can be very time-consuming especially for small batch sizes and multiple curved welding trajectories. Another advantage of manual welding is offered by the possibility to change important process variables like current, welding angle and feed rate instantly. This can improve the welding quality (Helms et al., 2002).

3.8 Robotic welding

There are two popular types of industrial welding robots: articulating robots and rectilinear robots[1]. Rectilinear robots move in line along any of three axes (X, Y, Z). In addition to linear movement, there is a wrist that allows rotational movement. Articulating robots employ arms and rotating joints and are the type of robots focused on in this thesis. Robotic welding needs to be engineered differently from manual welding and there are a number of factors that must be considered, for instance (McCabe, 2003)

- the number of axes needed,
- reliability and maintenance of the equipment used,
- accuracy and repeatability demands,
- fixture planning,
- calibration and programming of robots and sensors,
- seam tracking and other sensor system demands,
- control issues of robots and sensors,
- arc welding or laser welding equipment, and
- positioner and part transfer issues.

[1]A common configuration combines the two with a XYZ gantry and an articulating robot hanging upside down from the gantry.

To reach an arbitrary pose in 3D space, six-axes robots are needed but if seam tracking is not used, welding can sometimes be a five-axis process. The sixth axis is then free for rotation about the weld-wire which is a proposed method to avoid singularities (Olsson, 2002). Besides the six axes, external ones may as well be necessary to position the work-piece or to be able to move around large work-pieces.

Even if modern industrial robots are considered robust and reliable, they need scheduled maintenance on a regular basis. Preventive maintenance could keep a robot from experiencing major breakdown. Most robot manufacturers have courses on robotics maintenance. Robot maintenance training includes: training on safety operations, basic operations, periodic maintenance, countermeasures for troubles and exchange of worn units for recovery (Fuller, 2004). The robot's robustness may not be the weakest link in the chain. Tools, tool-exchangers and other equipment will also have to be checked and possibly repaired or exchanged.

Mechanical movements cause physical wear, and electronic power supplies, sensors, and analog-to-digital converters change in value with age. Preventive maintenance should include checking for possible accuracy and repeatability problems. Recalibration of the robot requires adjustment or replacement of mechanical or electronic parts. Sensor calibration is as equally important as robot calibration and orientation is often harder than positioning for correct calibration.

3.9 Weld methods

Several weld methods exist, each with different advantages and drawbacks associated with the welding engineering problem to be solved. The methods pertinent to this thesis are briefly discussed below. For outline of the theoretical and practical basics of welding and metallurgy, special literature is recommended (Cary, 1997).

3.9.1 Arc welding

Welding is one of the major applications in the use of industrial robots. Arc welding is a predominant application area for industrial robotics and the process normally includes control of positioning of the welding-torch and control of weld process variables in real-time. Different materials and quality demands need diverse approaches and many methods and variants exist. The most common techniques will be discussed here.

Gas Tungsten Arc Welding (*TIG*[2]) is a commonly used high quality and high precision welding process. In TIG welding, an arc is formed between a non-consumable tungsten electrode and the metal being welded. Gas is fed through the torch to shield the electrode and molten weld pool. If filler wire is used, it is added to the weld pool separately. TIG welding may require greater dexterity than other welding methods and has normally lower deposition rates and may be more costly (AWS, 1976).

Plasma Arc Welding (PAW) is achieved by establishing the arc and generating the plasma inside the torch head. The plasma reaches temperatures as high as $30000°$C near the tungsten electrode and exits the torch through a small-diameter aerodynamically designed constricting orifice that collimates the plasma and dramatically concentrates its energy into a beam-like, high-velocity stream. The small-diameter constrained plasma column provides directional control and produces narrower welds than the TIG process as well as deeper penetration for the same energy level. The higher energy density of the concentrated plasma heats the weld joint more rapidly and significantly decreases the size of the heat-affected zone adjacent to the weld (Eckart and Francoeur, 2002).

Gas Metal Arc Welding (GMAW) is frequently referred to as *MIG*[3] and *MAG*[4] welding. In this context, an inert gas is one that does not react with the molten material. MIG and MAG welding are high deposition rate processes where the wire is continuously fed from a spool. The shielding gas forms the arc plasma, stabilizes the arc on the metal being welded, shields the arc and molten weld pool, and allows smooth transfer of metal from the weld wire to the molten weld pool. The primary shielding gas in MIG welding is argon mixed with helium. In MAG welding, argon containing a small proportion of carbon dioxide or oxygen is an example of an active gas (McCabe, 2003; Sedlenieks, 2004).

The travel speeds achieved with arc welding is in the range of 0.25 to 2 meters per minute.

3.9.2 Laser welding

Laser beams have been applied for welding since the invention of laser techniques. In the early 1970s, laser beams where used for thick steel plate welding using experimental carbon dioxide lasers with output power exceeding 10kW. Applications for laser welding have increased steadily. Today, laser welding is versatile

[2]Tungsten Inert Gas
[3]Metal Inert Gas
[4]Metal Active Gas

with power ranging from a few hundreds to 60kW and can be applied to as different areas as joining miniature electronic components and welding steel structures thicker than 25mm (Farson and Duhamel, 2001).

The laser welding process speed and penetration contribute to its high productivity compared to arc welding. Speeds range from 1 to 10 meters per minute and are even higher for sheet metal applications with thin material thickness. In addition to single pass welding at relatively high speed, the laser process commonly does not utilize filler material that is necessary for multiple-pass arc welding. Furthermore, the deep penetration characteristic of laser welds usually allows single-pass welding, and finally, since laser welding is a process with minimum thermal distortion, fixturing may be simplified, resulting in shorter loading and unloading time and lower fixture cost (Farson and Duhamel, 2001).

3.10 Sensor technology normally applied to robotic welding

Tight tolerances and other requirements for automated welding may not always be possible to achieve. This, however, does not automatically imply that robotized welding is out of the question; sensory technology can sometimes be used for seam location, seam tracking, torch distortion compensation, metrology and to localize form errors. The most important techniques utilized today can be divided into

- wire touch sensing,
- through-arc sensing,
- vision-guided line scan systems, and
- vision-guided circular scan systems.

There exist also variations on the methods described here.

3.10.1 Wire touch sensing

After applying a sensing voltage to the weld wire, the robot is programmed to move to a series of positions relative to the weld joints. The tool point position is recorded when the wire touches the part and the voltage drops to zero. After a series of touches, the position of the joint found relative to the original program is calculated and the original program is adjusted. The technique is not applicable to all joints, materials and shapes (McHaney, 2001).

3.10.2 Through-arc sensing

The robot is programmed to weave the arc across the weld joint. The weaving results in a current change in the weld power supply and the robot controls offsets in the programmed trajectory to bring the weld current back to a specified level. As with wire touch sensing, this technique cannot be applied to all joints, materials and shapes (McHaney, 2001).

3.10.3 Vision-guided line scan systems

A laser camera is mounted a distance ahead of the weld torch. An accurate position of the weld, down to 0.1 mm, and process variables such as gap and joint angles, are measured. Based on an accurate calibration of the camera's pose compared to the weld torch, the weld pose is calculated and time stamped with regard to the weld speed and saved in a circular array of calculated values. The array length is proportional to the sensing speed and/or accuracy needed. But vision-based systems have drawbacks as well. Robotized welding is normally a five-axis process and a sixth axis can be utilized to avoid singularities by rotation along the sixth axis (Olsson, 2002). However, the vision system, prevents rotation by its need to be nearly perfectly perpendicular to the weld direction. Another problem is that despite the small size of modern laser cameras, they do make the robot system more sensitive to collisions.

3.10.4 Vision-guided circular scan systems

Using a circular scan rather than a line scan, three-dimensional data can be obtained from a single measurement. If the circle crosses the seam twice, the trajectory can also be calculated within one scan. A circular scan system is usually slower than a line scan system.

3.10.5 Calibration of line scan systems

When relying on a sensor for guidance, any initial calibration inaccuracy or subsequent misalignment during production will result in a robot system that consistently produces defective parts, unless this can quickly and automatically be detected and corrected. Laser welding is a technique that demands extreme spatial accuracy with which the robot must position the focal point of the laser with respect to the joint to be welded.

Determining the transformation that defines the sensor frame with respect to the wrist frame is referred to as the “sensor mount registration problem” and was first addressed by Shiu and Ahmad (Shiu and Ahmad, 1989). Several other approaches for obtaining a solution have been presented, see (Chou and Kamel, 1988; Tsai and Lenz, 1989; Zhuang and Shiu, 1993; Park and Martin, 1994).

In (Huissoon, 2002), a calibration system consisting of a reference object and a laser focal-point sensor is described. The reference object is designed so that a structured-light seam-tracking sensor can be used to determine uniquely the pose of a frame associated with the reference object with respect to a frame associated with the seam-tracking sensor solving a set of linear equations using the singular value decomposition (SVD) technique.

According to Huissoon, the technique works well with simulated data but is prone to measurement noise and provides only a good estimate when actual measurement data are used. However, by solving the forward problem: given an assumed sensor frame pose with respect to the reference object, the location of the edges in the sensor image can be predicted. By using the solution provided by the SVD algorithm as an initial estimate of the sensor pose, a multidimensional optimization technique such as the Simplex algorithm can be used to quickly converge to the sensor pose that provides a general least squares best fit to the measured edge locations (Huissoon, 2002).

3.11 Designing for automated systems

A simple but efficient type of tooling uses fixed locators and hand clamps to secure the part. With larger volumes more sophisticated approaches such as pneumatic or hydraulic clamping, part-presence sensing and auto-pneumatic clamping can be used. Products designed with arc welding in mind may minimize the costs and problems associated with jigs. By applying laser cutting, a jig-less approach can be utilized. However, the method requires relatively large volumes to cover its increased product development costs.

3.11.1 Tooling and fixtures

Air-driven actuators reduce load/unload time and improve ergonomics for fixtures that require a large number of clamps if clamping points are not easily reached and if high clamping forces are needed. By adding part-presence sensors, the robot system can confirm that parts have been properly loaded before part-processing starts. Auto-pneumatic clamping combines pneumatic clamping and part-presence sensing. This tooling is controlled by a PLC[5] to allow part-presence confirmation, clamp position sensing, clamp sequencing and extension or retraction during welding (AWS, 1976).

It is important to determine if the tooling and fixturing equipment should hold a pre-tacked subassembly or to hold individual pieces in their exact locations while they are being welded together. Holding individual pieces during a welding sequence is much more complex since they must be held securely in their relative position as the positioner moves and rotates at high speeds and accelerations (Crowe, 2001).

3.11.2 Part tolerances

To increase productivity in an automated system, the parts to be welded and the system itself must by designed properly. Careful re-design of existing parts may be necessary, and the design of new parts will probably change to accommodate to an automatic system. The tolerance for each component in the robotic arc welding system affects the overall tolerance of the system and when tolerances from robot, positioner, fixture and the part itself is summed up, the location of the weld may very well end up to be out of place. Size and capacity of components in a robot system can also affect tolerances; maintaining low tolerances is critical when the volume increases (Crowe, 2001).

3.11.3 Joint design decisions

After choosing the correct weld method, several ways exist that can be used to reduce welding costs and increase finished part quality (Crowe, 2001):

- Reduce the distortion potential and eliminate possible secondary processes such as bending or straightening by welding close to the centerline of the part.
- Increase the travel speed of the welding robot and reduce the amount of filler material by using the minimum root opening and the smallest included angle needed.
- Improve weld quality and reduce heat input to the part by designing the weld joint for torch access and maintaining the proper torch angle throughout the weld.

[5]Programmable Logic Controller.

However, the methods above increase in general the demands on the control of the weld process as the allowable process operational window with respect to various variables becomes smaller.

Joint design decisions also include methods of stress calculation, structural design philosophies and their influence on the choice of welding processes and procedures and their effect on fatigue and fracture behavior. Also practical constraints on joint design and configuration in the context of joint preparation and access must be considered. For a deeper understanding, see (Hicks, 2000).

3.11.4 Parameterization of robot programs

To be able to handle different variants of parts without writing a new robot program for each part, *parameterization* of the robot program is a commonly used method. Normally the differing shape parameters are saved in files residing on a hard drive on the robot controller, and the appropriate set of parameters is chosen by user-selection on the teach pendant.

This is a technique that works well if parts have similarities in shape, for instance different lengths within a product line of forks to forklifts, but it does not work if parts have different forms. One-off manufacturing or small-batch series of different products cannot benefit from using this method. For that kind of production, the only option is to write different robot programs or to write a single and complex program. Therefore, robots are rarely used for one-off products unless quality demands make manual welding unfeasible.

3.12 Conclusions

The welding process is one of the most complex manufacturing processes and frequently the least understood. Choosing a weld method and weld variables is difficult both in manual and robotized welding because of all the different aspects mentioned. Robotized welding not only includes optimization of the robot process but also introduces new difficult decisions in terms of feeding, sensing and robot and sensor programming.

There is a limited use of advanced sensors in the industry and they are mostly used for closing local loops. The reluctance to use sensors is based on

- the assumed increase in system complexity that normally follows,

- the fact that only parts of the process information needed are normally given by the sensor,
- the mapping problem between what is observable and controllable, and
- that the sensors normally are only used in a local loop; decisions are taken at a too low level.

A fully automated system is not a realistic goal today. However, tools that make it possible to use advanced sensors in robotized welding will give the welding engineer the opportunity of making his specialist input to the manufacturing process for one-off products and a tool to handle the increase in complexity.

The most flexible manufacturing system takes advantage of both the robot and the welding engineer. The knowledge and the skillfulness of the welding engineer are combined with the advantages of the robot (e.g. strength, endurance, speed, accuracy) to an enhanced system (Helms et al., 2002).

Chapter 4

Programming of robots

4.1 Introduction

Robot programming is the means by which a robot is instructed to perform its task. The art of robot programming is almost as old as the development of robot manipulators. The first was a language, called MHI, designed by Ernst at MIT in 1961 (Ernst, 1961). Two languages developed at Stanford University, WAVE (Paul, 1977) and AL (Finkel, Taylor, Bolles, Paul and Feldman, 1974; Mujtaba and Goldman, 1979) were particularly influential in the field. The earliest efforts were on hardware level via point-to-point and simple motion-level languages to motion-oriented structured robot programming languages (Blume and Jakob, 1986; Ránky and Ho, 1985). But as with programming languages and environments in general, the robot programming efficiency has, despite the tremendous efforts undertaken, by no means followed the manipulator hardware improvements.

A common denominator of modern robot programming is that it has to deal with both the real, physical world and a virtual world representing – and possibly augmenting – the physical world. The two worlds need to be kept synchronized at all times. Robot programs are basically computer programs which deal with a richer variety of I/O devices. However, manipulating objects in the physical world may lead to errors, and therefore the programmer must try to anticipate and test erroneous conditions (Korein and Ish-Shalom, 1987).

The position and orientation of a rigid object in space with respect to a reference frame may be represented as a 4×4 homogeneous transformation matrix with, when the last row is ignored, three unit vectors describing rotation $(\mathbf{n}, \mathbf{o}, \mathbf{a})$ and a translation vector $\mathbf{p}$ locating the origin of the object coordinate system. The conventions for the names[1] where taken from Paul (Paul, 1981). Although not computationally efficient (Taylor, 1979), the homogeneous matrix representation is particularly convenient for manipulation. In the context of robotics, the homogeneous matrix representation has come to be called a *frame*. It is important in robot programming to be able to specify the pose of an object with respect to a reference frame and then obtain that pose with respect to another reference frame and this conversion is made simple by the use of homogeneous transformations (Korein and Ish-Shalom, 1987).

The testing and debugging of robot programs is characterized by much greater degree of trial and error than many other kinds of programming. It is common to debug robot programs at low speed, and then increase the speed as much as possible without producing intolerable losses in accuracy (Korein and Ish-Shalom, 1987).

There are many different ways to specify robot-programming systems. In this thesis, we will divide the field of robot programming into industrial robotics programming and approaches used in research projects.

4.2 Industrial robotics programming

There are basically two programming modes in developing programs for industrial robots: on-line and off-line programming (Nitzan, 1990; Lozano-Perez, 1987*a*). Most programming languages are controller-specific and robot manufacturers often provide a simulation system as well. By using a translator between robot languages a single simulation system can be used. A translator framework worked out by various researchers has been used in commercial environments (Freund, Ludemann-Ravit, Stern and Koch, 2001). However, a few expensive off-line systems provide the means to generate robot programs for different target robots.

4.2.1 Teach programming

A safe but time consuming method to program the robot is to manually move the robot using a teach pendant. This time demanding procedure also halts the production and each variation of a workpiece demands a new or modified robot program, reasons that make teach programming suitable only for large series manufacturing of identical workpieces (Jacobsen, 2004). While a substantial research

[1]Because the object was often a gripper, $\mathbf{o}$ indicates the axis along which the gripper *opens*, $\mathbf{a}$ indicates the *approach* direction in which the gripper points, and $\mathbf{n}$ indicates their common *normal* in the direction $\mathbf{o} \times \mathbf{a}$ (Korein and Ish-Shalom, 1987).

effort is directed toward task-level programming, most industrial robots are still programmed using a simple pendant. However, teach pendants are ill suited for tasks involving complex manipulator trajectories, or when there is increased reliance on outside sensing (Burdea, 1999).

4.2.2 Off-line simulation systems

Off-line programming is a term that is usually applied to a collection of techniques for robot programming without actually using a (physical) robot. A Computer-Aided Design (CAD) system is typically used to model the robot workstation, parts, and auxiliary equipment. Then the simulated robot is programmed and its task executed in the simulated environment (Meyer, 1981). However, if a model does not include uncertainties in part position, part dimensions, and robot position, the simulation will succeed in situations where a real application would fail (Korein and Ish-Shalom, 1987).

Off-line simulation systems were introduced in the mid 1980's and today make up a mature technology that permits performance and functional simulation of work-cells. These systems are usually programmed in virtual systems such as IGRIP[2], eMPower[3] and CimStation[4]. Some features of off-line systems are

- cycle time evaluations,
- reach determination,
- trajectory optimisation,
- collision detection,
- jigs and fixture design,
- calibration, and
- evaluation of robots.

Besides being able to create the robot program without interfering with the physical robot system except for touch-up, the major benefit of off-line systems is the possibility of doing advanced trajectory optimizations to achieve shorter cycle times. Some of these optimizations, especially in cases where two or more robots

[2]http://www.delmia.com/
[3]http://www.tecnomatix.com/
[4]http://www.acel.co.uk/

move in the same working-area, would not be possible by manual-processing. There are many systems currently on the market, both from robot vendors targeting their own product range, for instance RobotStudio[5] and from other suppliers that are independent such as IGRIP, eMPower and CimStation.

Off-line programming environments typically offer a simulation language that hides the underlying robot language, which means that the robot programmer “only” has to learn one language, even if robots from multiple vendors are to be programmed. However, they are normally not feasible for advanced sensor simulation.

4.2.3 “Automatic robot programming”

In (Jacobsen, 2004), a third method to program a robot which Jacobsen have named “Automatic robot programming”[6] (ARP) is discussed. In ARP, robot trajectories are automatically calculated based on CAD information, a task description, sensor data and expert process knowledge from a robot operator. A disadvantage with ARP is that the system must be configured specifically for each type of production. While CAD data has been standardized to a few formats, task descriptions have not and it is not trivial to transform task descriptions into ARP system readable forms. Process knowledge is important and even processes that look simple actually contain extensive human expertise and successful robot integration depends on to what extent process knowledge is paired with the task description (Jacobsen, 2004).

At the Odense Steel Shipyard in Denmark, an ARP installation has been running for nearly ten years. Steel plates are joined in large block assemblies which are similar in construction but vary in details. Multiple gantry-mounted robots are off-line programmed to weld fillet joints on block assemblies. Sensors are utilized to find the exact start point and to correct deviations of the groove compared to CAD data during welding. The system has a theoretical capacity of welding 2000-2500 meters per day, but about 5000 meters are weld per week due to restrictions in the assembly line. When the wire mesh input data is replaced by modern solid 3D information and with the introduction of a more advanced ARP tool, the system is expected to be able to weld about 90% of all seams (Jacobsen, 2004).

Jacobsen identifies five important issues to take into account when introducing

[5]http://www.abb.com

[6]The term Automatic Robot Programming is in (Wahl and Thomas, 2002) referred to as a task-level programming systems where the application (*what* to do) is specified on a high abstraction level and is automatically converted into a sequence of actions/motions (*how* it should be done) by a task planner.

robots in a manufacturing system (Jacobsen, 2004):

1. The operator's knowledge and understanding of the system.
2. A good working position, enough workspace and degrees of freedoms for the robot.
3. The use of sensors to allow for imprecisions of the workpiece.
4. Good planning of the capacity of the robot work cell.
5. CAD data *must* fit the real world.

4.3 Approaches used in research projects

Besides controller-specific languages a few different approaches exist. In research environments it is not uncommon to extend a high-level language such as C, C++ and Java to provide robot-specific functionality. Some implementations are specific to a certain robot, but others use abstract classes to circumvent hardware differences. One example is Pyro[7] that provides abstractions for robots and algorithms. Pyro is written in Python, which is an interpreted language. This means that it is easy to experiment interactively with the robot program on the expense of the execution speed.

Virtual systems should be suited for sensor-intensive tasks, but robotic languages are dependent on the particular manipulator used, and the debugging stage is quite time consuming (Blume and Jakob, 1986; Rehg, 1997; Burdea, 1999). The DLR project, a man-machine interface based on a high performance VR[8]-environment (Landzettel, Brunner, Hirzinger, Lampariello, Schreiber and Steinmetz, 2000), and the project at Fraunhofer IPA in Stuttgart, Germany (Strommer, Neugebauer and Flaig, 1993; Flaig, Grefen and Neuber, 1996), are examples of attempts to advance the state-of-the-art in robot programming. In the Fraunhofer project, a GUI is used, which allows the user to specify the dynamic behavior of components. Once the program is completed, it is downloaded to a real robot connected to the same VR engine and the same task is executed (Burdea, 1999).

Behavior-based languages provide a different approach, specifying how a robot should react to different conditions and defining functionality that the end-user would use to perform tasks. An alternative to text-based methods is graphical

[7]http://emergent.brynmawr.edu/pyro/?page=Pyro
[8]Virtual Reality.

programming systems. They typically use flow-charts and are considered easy to use, at least for simple application programming. Lego Mindstorms robotics kit LEGO[9] let the user create advanced performances by combining stacks of low-level actions to form high-level tasks.

A graphical system for off-line programming of welding robots has been developed with the main objective to increase sensor use in welding robot programs by providing a user-friendly interface (Dai and Kampker, 2000). An icon-oriented interface provides the main programming method. To make it easy to incorporate sensor operations, macros are defined in a sensor editor.

Another method, *Programming by demonstration*, is to use teach pendants for demonstration. Current research in this area extracts information besides move-instructions in order to, for example, optimize paths (Chen and McCarragher, 1998; Chen and McCarragher, 2000; Chen and Zelinsky, 2001*b*; Chen and Zelinsky, 2001*a*). There are mainly three different ways to collect information: voice, touch and vision. While these systems certainly are improving, they are not easily used in cluttered industrial environments. Programming that includes virtual environments will probably prove to be more successful. For advances in this particular area, see (Friedrich, Holle and Dillmann, 1998; Zollner, Rogalla, Dillmann and Zollner, 2002; Onda, Suehiro and Kitagakiand, 2002).

Voice- and gesture recognition can be used to command robots to carry out tasks for which they are already programmed. These *Instructive Systems* use for example gestures to find and direct the attention of the robot to particular objects, avoiding and overcoming problems caused by cluttered environments (Strobel, Illmann, Kluge and Marrone, 2002; Steil, Heidemann, Jockusch, Rae, Jungclaus and Ritter, 2001).

4.4 Programming languages

4.4.1 Motion-oriented programming languages

Modern motion-oriented programming languages were first developed in the mid seventies. Some of the proprietary languages used today still lack the sophisticated data structures that can be found in the early languages such as VAL (Shimano, 1979) and AML (Taylor, Summers and Meyer, 1982), although elaborated languages exist, for instance ABB Rapid (*ABB Rapid Reference Version 3.2, RAPID Summary*, n.d.). Today, well over a hundred robot languages exist.

[9]http://mindstorms.lego.com/eng/products/ris/rissoft.asp

Attempts to introduce general languages (Fahim and Choi, 1998; Lapham, 1999) have so far had none or little impact on industrial robotics.

Modern robot languages use frames, often expressed by homogeneous transformation matrices that somehow conform to human intuition, and hide the robot hardware behind software interfaces. The use of internal and external sensors has led to the monitor concept (M. A. Lavin, 1982). Monitors are small user-defined or built-in programs heavily communicating with both user applications and the robot control system's motion pipeline. Monitors read a set of sensors in specified time intervals, or depend on changed sensor values and react by modifying the path *'guided motion'* or trigging some action *'guarded motion'* (Wahl and Thomas, 2002). While this monitor concept seems to be useful, it is hardly applied today in commercially available robot programming languages.

4.4.2 Task-oriented robot programming languages

The idea behind task-oriented robot programming is to hide machine details and let the user specify an application at a higher level of abstraction. Instead of lengthy programs that describe *how* to do something, the emphasis is on *what* to do. As task-level specifications do not describe the details of how the robot is to perform an operation, the translation of such a specification into a robust robot program is a central research problem in robot programming. Some pioneer efforts in this area can be studied in (Feldman, 1971; L.Lieberman and Wesley, 1977; Popplestone, Ambler and Bellos, 1980) and (Lozano-Pérez and Winston, 1987).

Still, the complexity of a task remains of course even if we are successful in hiding it under a task-oriented interface. Human interfaces, e.g. GUIs, and commands, have to be developed. These interfaces will, when invoked by the user, transfer the high-level commands to a sequence of actions and motions.

If this transformation is made automatically, we normally talk about a *task planning system* such as the one described in (Mosemann and Wahl, 2001; Thomas and Wahl, 2001). The most difficult part in automatic robot programming seems to be execution and control of the generated action/motion sequences, including the generation of collision-free paths and force-controlled mating operations (Wahl and Thomas, 2002). A task planning system also includes a *motion planner*.

Grasp motion planning handles object grasping. Early work made by R. Paul on grasp planning made the assumption that the gripper should be positioned so that, at grasp time, simply closing the jaws produced a stable grasp on the workpiece but was, in the presence of uncertainty, not stable (Paul, 1972). Existing methods for automatic grasp motion planning handles objects with certain geometric restrictions; arbitrarily shaped objects would magnify the time and space complexity. In (Miller, Knopp, Christensen and Allen, 2003), an object is modeled as a set of shape primitives (spheres cones, boxes etc.) and uses a set of rules to generate a set of grasp starting positions and pre-grasp shapes that can be tested on the object model.

Gross motion planning computes the intermediate paths and includes obstacle avoidance which is a problem that has received considerable attention. Early work includes (Udupa, 1977) and (Lozano-Pérez, 1987*b*). In (Hwang and Ahuja, 1992), research directions and their performances are briefly described. Several methods exist, among them approaches based on a numerical potential field method (Barraquand and Latombe, 1991; Barraquand, Langlois and Latombe, 1992). Other attempts are based on an idea of attraction and repulsion forces and, more recently, probabilistic road map planners (PRMs) (Bohlin and Kavraki, 2000). As with grasp motion planning, all methods have to trade between computation time and path optimization.

While task-oriented programming in theory provides mechanisms to convert abstract high-level actions into low-level code, there are some difficulties to overcome according to (DaCosta, Hwang, Khosla and Lumina, 1992):

- no generalized mechanism to translate the task specification low-level code exists,
- sensor integration is difficult, for example error detection and part variation handling, and
- it is often difficult to represent complex real-world scenarios.

4.5 Task-level programming systems

According to (Meynard, 2000), a *task-level programming system* architecture must be able to cope with three main issues: task specification, representation of objects and robot program synthesis.

The task specification specifies a task to the planner as a sequence of actions on the objects in the world model. Different concepts exist on with respect to specification of these actions. The spatial relationship between objects is described by their relative position in (Nnaji, 1993). This method is interesting because it allows a high-level of automatic reasoning leading to advanced automated planning. Other methods rely on manually created sequences of actions that let the user describe the requested task without creating an object model at a specified position.

The *world model* contains the spatial, kinematical and dynamical features of the objects related to the task.

The *robot program synthesis* corresponds to three steps: sequence planning, motion planning and plan checking. Sequence planning translates a task into a low-level code. Then, motion planning finds a collision-free path that is kinematically suitable for the manipulator. Finally, plan checking guarantees that the planned task does not violate any rules and is allowed in the current system state.

In (Meynard, 2000), an experimental research platform, XPROB, is presented that features “a task-level programming environment, a dynamic representation of the work-cell’s equipment, and sensor data integration at run-time allowing on-line program monitoring and adaptation”. By introducing an interpreted task-level programming language that hides hardware-specific references, the XPROB system has, according to Meynard, been made application-independent and hardware-independent.

DLR[10] has focused its work in space robotics on the design and implementation of MARCO[11], a domain-specific high-level task-oriented robot programming and control system. The goal is to develop a unified concept for a flexible and highly interactive on-line programmable teleoperation ground station as well as an off-line programming system, which includes all sensor-based control features partly tested in the ROTEX[12] scenario (Brunner et al., 1993; Landzettel et al., 2000; Landzettel, Brunner, Schreiber, Steinmetz and Dupuis, 2001).

In ROTEX, tasks such as assembly, tracking and capture of an object floating in zero gravity were successfully executed despite almost seven seconds of round trip time delays. The key to the impressive demonstration was the use of virtual models of the robot geometry as well as of its sensory data (Burdea, 1999).

DLR’s system provides a flexible architecture that can be adapted to application specific requirements. Programming and control methodology is based on the use of sensors such as cameras, laser range finders and force-torque sensors. Applications that include sensor feedback loops in a well-known environment can be preprogrammed and verified on-ground (Landzettel et al., 2000; Landzettel et al., 2001).

Tasks can be composed in a virtual world by a user without expertise in robot-

[10]Das Deutsche Zentrum für Luft- und Raumfahrt (Eng: The German Aerospace Center).

[11]Modular Automation and Robotics COntroller.

[12]ROboter Technology EXperiment

ics. A man-machine interface based on a high performance VR-environment. The power of the task-oriented sensor-based programming approach was shown 1999, demonstrating the vision and force control scheme by executing a prototypic peg-in-hole task fully autonomously on-board. All the operations were preprogrammed in the simulation environment on-ground, including image processing and vision controller parametrization (Landzettel et al., 2000).

The task-level architecture in (Landzettel et al., 2000) was based on two layers, the *expert layer* and the *user layer*. The expert layer controls *how* the system should act to successfully execute the task and make heavily use of sensor data processing algorithms. In the virtual environment, nominal sensor patterns were stored and appropriate robot motion reactions were generated as elementary operations.

An operation in the user layer was characterized by a sequence of elementary operations. These operations were, however, not visible to the user. The operator only selected the object/place that should be handled. This was reflected by the virtual reality environment which showed the work cell without the robot. Via a 3D interface, an object could be grasped and moved to an appropriate place thereafter a specific VR-hand gesture started the task execution (Landzettel et al., 2000).

MARCO is a task-oriented programming and controller framework to program sensor-based robots like the DLR 7-axes light weight robot (Hirzinger, Albu-Schäffer, Hähnle, Schäfer and Sporer, 2001), which enables an operator to pre-simulate the robot tasks at a graphical simulation, including all sensor processing (force-torque, laser range finder, vision[13]). The controller code for the simulation as well as the real system is the same, however the interfaces to robots and sensors are not identical for simulation and real operation. PowerPC and VxWorks are used for the real-time environment, while Linux or IRIX are used for the simulation. Almost all the code (written in C) is platform-independent[14].

Donald L. Pieper, who derived the first practically relevant result when he in his Ph.D. thesis (Pieper, 1968) showed that the inverse kinematics of serial manipulators with six revolute joints, and with three consecutive joints intersecting, could be solved in closed-form, i.e., analytically, 1989 posted a proposal to NASA[15]'s

[13]The sensors are general models of force-, camera-, and laser scanner type, and not models of existing sensors.

[14]Thanks to Bernhard Brunner at DLR for sharing information about MARCO not disclosed in papers.

[15]National Aeronautics and Space Administration

SBIR[16] program. The abstract of his[17] proposal Macro- and Task-Level Programming of Arc Welding Robots for Aerospace Applications" contains ideas related to those presented in this thesis and is quoted below:

Abstract:

The goal of the overall project is an innovative programming environment for the next generation of advanced welding robot controllers. This environment will incorporate macro-level programming, icons for user interfaces, graphical simulation, and, possibly, elements of task-level programming, e.g. automatic path planning and weld sequence optimization. Macro-level programming, aspects of which will be considered in Phase I, refers to the generation of complex part programs from primitives which encapsulate all the required motions and operations needed to weld generic classes of parts, components, or joints. Such developments could significantly improve productivity and consistency of teaching welding robot programs and impact the cost and reliability of robotic welding in aerospace application. Phase I tasks include: a brief review of related work, analysis of welding requirements for aerospace fabrication, preliminary design of the advanced programming environment, and evaluation of macro-level programming approaches. The feasibility of the proposed schemes will be examined, needs for future research will be identified, and the Phase II effort will be planned.

Potential Commercial Applications:

Advanced welding robot controllers may benefit commercial low-volume/small-batch robotic welding applications.

Unfortunately, no documentation except the above abstract has been made publicly available[18] but the concept is nonetheless relevant and close to some ideas presented in this thesis.

Task-level programming systems have been identified as a goal in the field of robot programming, since they should be able to produce a robot program solely from a description of a desired final state without specifying the actions needed to achieve it (Ude and Dillmann, 1995). The problem of going from the final state description to the underlying actions at a low level has been a problem considered in planning

[16]Small Business Innovation Research (http://www.sba.gov/sbir/). This particular project was sponsored by NASA's Marshall Space Flight Center.

[17]The proposal (89-1-04.10-7900) was actually sent from Automatix, Inc, Billerica, MA 01821, USA.

[18]This has been confirmed by a representative from NASA SBIR Program Support and National Technology Transfer Center (NTTC). Technology Transfer products and services that are sponsored by NASA are not available to individuals or organizations outside the United States or to resident aliens living in the United States.

research in the field of artificial intelligence (Nilsson, 1980; Sacerdoti, 1977).

An obvious approach to augment workspace knowledge is to use sensors. But each sensor has its own issue. Vision-based systems, for instance, suffer from inaccuracy even in a clean laboratory with close to optimal conditions. In disordered shop floors, these drawbacks become even more pronounced. If multiple sensors are used, sensor fusion problems may also occur.

4.6 Robot programming libraries and environments

Besides the commercial robot programming environments that normally turn to end-users and in-house developers, a number of "open source" software libraries exist that are released under different licenses. These libraries would normally interest application programmers or serve as a base for research projects. Most open source software projects have originally been developed as a side effect of research in some particular area.

4.6.1 Orocos

The OROCOS[19] (Open Robot Control Software) organization (*OROCOS*, 2004) ambitious goals are to develop robot control software under Free Software[20] and/or OpenSource[21] licenses for all sorts of robotic devices, including manipulators, mobile robots and humanoids offering an infrastructure that is independent of any particular architecture for both hard real-time and non real-time applications.

The organization also attempts to contribute to the development of programming interfaces that can be accepted as standards by robotics sub-communities and to the development of free educational material. Orocos is written in C++ and runs under Linux[22] and RTAI[23], a Real-Time Linux Application Interface.

The Orocos project follows a component-based strategy, where components alter their state when called. This modular robot control framework provides the infrastructure and the functionalities to build applications. The visionary software engineering requirements include (Bruyninckx, Soetens, Issaris and Leuven, 2002):

- Object-oriented design.

[19]http://www.orocos.org
[20]http://www.fsf.org/philosophy/license-list.html
[21]http://www.opensource.org/licenses/
[22]http://www.linux.org
[23]http://www.aero.polimi.it/ rtai/

- Decoupling and modularity. Class-wise interfaces and complete encapsulation of internal data to promote independent evolution and re-implementation of classes.
- Small and shallow interfaces. Separated interfaces give more flexibility in changing implementations.
- Distributable. The robot control software benefits from being scalable and network-based. Abstraction layers are needed for both operating system and interfacing hardware.
- Minimalism. Superfluous APIs may lead to different implementation with similar functionality.
- Hardware independent. Linux has been the operative system of choice for both development and deployment.
- Thorough large-scale design. Safe design should always consider a complex distributed and hard real-time implementation.

Orocos started in September 2001, is still developing but is far from complete. The current status is available on the Orocos website.

4.6.2 Pyro

Pyro provides abstractions for robots, especially mobile robots, and algorithms. Pyro is written in Python, which is an interpreted language. This means that it is easy to experiment interactively with the robot program on the expense of the execution speed. Since Pyro abstracts underlying hardware details, it can be used for experimenting with several different types of robots and robot simulators. Currently, robots supported include the Pioneer and the Khepera family. These are all small, mobile robots. Industrial robots do not seem to be a primary target.

4.6.3 Robotics Toolbox for Matlab

The Robotics Toolbox[24] (Corke, 1996) provides many functions that are useful in robotics, such as kinematics, dynamics and trajectory generation. The toolbox provides functions for manipulating data types such as vectors, homogeneous

[24] http://www.cat.csiro.au/cmst/staff/pic/robot/

transformations and quaternions (Chou, 1992). It is also able to graphically display any robot that can be represented in the Denavit and Hartenberg parameters (Denavit and Hartenberg, 1955). It also includes a Simulink[25] block library. Robotic Toolbox is small and simple and is equipped with source code.

4.6.4 SPACELIB

SPACELIB[26] (Legnani, Casolo, Righettini and Zappa, 1996; Legnani, Casolo, Zappa and Righettini, 1996) is a library for 3D kinematics and dynamics of systems of rigid bodies and is available in C and in Matlab source code. It is based on an extension of the homogeneous transformation matrix approach by Denavit and Hartenberg and adds 4×4 matrices for velocity, angular- and linear accelerations, and 4×4 matrices for forces, torques, angular and linear momentum and inertia. Its functionality includes basic operations on matrices, points, lines and planes, vectors and transformation matrices and is freely available for non-profit activities.

4.6.5 ROBOOP

ROBOOP[27] (Gourdeau, 1997) is an object-oriented programming toolbox written in C++ for robot kinematics, dynamics and linearized dynamics of serial robotic manipulators. It depends on the NEWMAT10[28] matrix library and if graphical features are to be used, also the gnuplot[29] software. It is released under the terms of the GNU General Public License[30]. It includes operations on homogeneous matrices such as rotation and translation, operations on quaternions and a specific robot class, which for each link handles kinematics as defined by Denavit and Hartenberg, link- and motor inertias and motor gear ratio and friction.

[25]Simulink (http://www.mathworks.com/products/simulink/) is an interactive tool for modeling, simulating and analyzing dynamic, multi-domain systems.

[26]http://bsing.ing.unibs.it/%7elegnani/

[27]http://www.cours.polymtl.ca/roboop/

[28]http://www.robertnz.net/

[29]http://www.gnuplot.info

[30]http://www.gnu.org/copyleft/gpl.html

4.6.6 Open Dynamics Engine

The Open Dynamics Engine[31] (ODE) is a free library for simulating articulated rigid body dynamics. It has built-in collision detection and according to the author, Russel Smith, ODE is “reasonably mature and stable”. ODE is designed to be used in interactive or real-time simulation. The user has the freedom to change the structure of the system while the simulation is running. ODE is written in C++ but uses a native C interface. ODE is a computational engine and completely independent of any graphics library. However the examples that come with ODE use OpenGL. It is released under either the GNU Lesser General Public License (LGPL)[32] or the BSD-style license[33].

4.6.7 Simderella

Simderella[34] (van der Smagt, 1994) is a robot simulator. It was originally developed to aid research in neural networks for arm control. It consists of a simple controller, a forward kinematics part (simulator), and an X-Windows oriented graphics back-end. The controller, simulator and graphical back-end are separate processes synchronized by message parsing. The flexible communication setup lets the controller handle multiple simulators and/or real robots simultaneously if required. The simulator handles kinematics for any robot with rotational and/or translational joints as defined by Denavit and Hartenberg (Denavit and Hartenberg, 1955).

During operation, the simulator expects joint values, velocities and accelerations from the robot socket connected to the controller. At each time step, the simulator translates the current pose of the robot into a set of homogeneous matrices describing the position and orientation of the links in Cartesian space. These matrices are then sent to the graphical back-end, followed by a matrix describing the pose of the target object, which place the objects on the canvas. In addition, it has a user interface for object rotation, translation, scaling and picture dumping. It can also render z-projection of all shown objects – i.e., a shadow on the floor. Simderella 3.0 is written in C++ and was released 1999 under the GNU Public License.

[31] http://q12.org/ode/
[32] http://www.opensource.org/licenses/lgpl-license.html
[33] http://opende.sourceforge.net/ode-license.html
[34] http://www.robotic.dlr.de/Smagt/software/simderella/

4.6.8 Game Engines

Game engines offer frameworks for games or game-like applications. Their power lies normally in appearance features such as lighting, shading and reflections. Rendering and visualization is important to game engines and how models and worlds are stored is often apart of the function of the renderer. In game engines, the 3D pipeline (the equivalent to the motion pipeline in off-line systems) handles everything from scene/geometry database traversal via geometry, different transforms, triangle set up[35] and rendering/rasterization to the final display on screen. As a contrast to most robotic off-line programming applications, which are proprietary and expensive, many sophisticated game engines are released under GNU and other licenses and can be downloaded from the Internet.

One example is Crystal Space[36] which is a free (LGPL) and portable 3D game development kit written in C++ with an extensive list of features. Crystal Space currently runs on all major platforms.

4.7 Conclusions

Off-line programming systems have evolved to efficiently handle design and optimization of robot programs. Their greatest strength is the predefined robot libraries and also their ability to visualize the designed work-cell. However, as design tools and run-time environments for sensor-driven robot applications, the generic, complex, monolithic and integrated approach yields an unpredictable performance. Instead, specialized software based on the problem to be solved has the potential to create efficient solutions to problems that include sensor-guided robots.

There obviously exists a somewhat higher threshold in programming a system from “scratch”, but the lesson learned from the author’s own experience is that even if one usually gets a quick start with some pre-engineered expensive tool, the thrill of development speed is likely to end well before the project objectives are fulfilled.

Open source libraries and frameworks are available for free on the Internet. Almost all of them have developed as offsprings from research projects and have therefore particular strengths and weaknesses. Nevertheless, they usually represent good ideas, and provide inspiration and sometimes source codes that can be used as a basis for specialized software development. The use of a game engine

[35]E.g. back-face culling, slope/delta calculations and scan-line conversion
[36]http://crystal.sourceforge.net

involves commitment to its large infrastructure and will only be beneficial if most of the features of the engine will be used.

Task orientation will mean as much for robot technology as object-oriented programming has improved software engineering. Research shows that there are tremendous obstacles that must be overcome to find general solutions (DaCosta et al., 1992). These difficulties clearly show that it is not realistic to find a generic task-oriented approach valid for the vast robotic area. But still, even if general mechanisms cannot be found, particular application areas can and will benefit from task orientation. Arc welding is one such area.

Chapter 5

High-level control

5.1 Introduction

For low-level control, a close relation between what a sensor measures and what is to be controlled is not uncommon, but for a sensor measuring more complex data patterns or data from several sensors, the matter is not so trivial. A seam tracker for arc welding can of course be used for simple positioning control, but if it provides some geometrical features of the weld joint profile, that information can be used for high-level control of the process or the sub-task the robot is performing[1].

High-level control in industrial robotics is a vast research area and a selection of interesting topics related to this thesis is discussed. Starting with a short introduction to artificial intelligence from an industrial robotics point of view, a discussion of the concept of sensors in general and virtual sensors in particular and their relation to high-level control follows, and finally, a few examples of research projects in the wide robotics research field related to high-level control are presented.

[1]Conceptually, such sensor data is considered as process-related feedback information. If, for example, a gap in the weld joint is varied, but still within acceptable limits defined in a WPS, different controlling actions may be needed. Such actions can be a combination of controlling the welding power source and the motion of the robot. A wider gap may, for instance, need a lower travel speed and/or weaving. Likewise, the orientation of the weld relative to the horizontal plane may need to adjust based on the size of the weld pool.

5.2 Artificial intelligence

Artificial Intelligence (AI) is an interesting area that has always had a close relation to robotics in general. It has always been a delicate problem to define what AI is. Problem areas that earlier were thought of as AI areas are now well known and reinterpreted (Stork, 2001).

One of the central lessons learned in the last 40 years of AI research is that problems that were thought to be difficult turned out to be less troublesome and vice versa (Stork, 2001). AI papers and textbooks often discuss significant questions, such as "how to reason with uncertainty" or "how to reason efficiently". It is hard to find descriptions of ambitious, interesting and concrete AI problems (Selman, Brooks, Dean, Horvitz, Mitchell and Nilsson, 1996).

The success of Deep Blue[2](*DeepBlue*, 2004; Schaeffer and Plaat, 1997) is a good example of something that at the time was a concrete AI challenge problem: develop a program that is able to defeat the world chess champion. However, the Deep Blue was not superior to Kasparov in strategy and tactics because of any sophisticated, heuristically guided search AI-method. Deep Blue's strength was an efficient brute-force search. In fact, it was so successful that it led Kasparov to exclaim "I could feel – I could smell – a new kind of intelligence across the table" (Kasparov 1996).

While many AI researchers may not like the lesson learned from the Deep Blue experience, the value of brute-force over more "intelligent" search-forms is, nevertheless, very substantial (Selman et al., 1996). However, chess is far easier than many other tasks like those involved in the evolution of an infant, such as understanding a simple story, recognizing objects and their relationships, understanding speech, and so forth. For these and nearly all realistic AI problems, the brute force methods in Deep Blue are hopelessly inadequate (Stork, 2001).

5.2.1 Artificial intelligence applied to industrial robotics

The AI community seems to have a limited interest in concrete applications in general and industrial applications in particular. The important goal of having a completely automatic and still reliable task-level approach in a concrete industrial robotic case does not yet appear to have been fully examined by the academic world.

[2]Deep Blue is an IBM supercomputer that 1997 had a special-purpose hardware that enabled it to calculate nearly a quarter of a billion chess positions per second

Trajectory planning, which often requires exploring large search spaces, raises critical complexity issues and there is strong evidence that any complete algorithm will require exponential time in the number of degrees of freedom (Reif, 1979; Canny, 1989). The AI community has made some progress, mainly in assembly, but academia has yet to find industrial robotics an interesting platform for applying its technology. A reason for the reluctance to apply AI in industrial robotics may be that the high process accuracy demands, both in absolute accuracy and in repeatability, normally prevent the uncertainty-based AI algorithms.

Another problem is the problem of time allocation for planning versus control and sensing (Halperin, Kavraki and Latombe, 1999). This problem remains poorly understood, though promising ideas have been proposed (Boddy and Dean, 1989; Nourbakhsh, 1996). Moreover, real robot tool trajectories deviate from planned paths due to errors in control and position sensing and such errors raise recognizability issues which make planning more complex (Halperin et al., 1999).

5.3 Sensors

Simple switch-type sensors are often used in industrial robotics, but more advanced sensors such as cameras and seam trackers are still uncommon. The goal is to reach a higher level of autonomy, but the introduction of sensors is associated with high-level control issues. A higher level of autonomy implies complex relationships; there is normally no straight coupling between observable and controllable variables and between controllable variables and variables that define the task.

A comprehensive survey of sensors that describes general aspects, technical and physical fundamentals, construction, function, applications and developments of the various types of sensors can be found in (Gopel, Hesse and Zemel, 1997).

5.3.1 Sensor simulation

Computer-based sensors known as *simulated sensors* or *virtual sensors*[3] should yield a significant leap forward in industrial robotics. In (Cederberg, Olsson and Bolmsjö, 1999), the author defines a virtual sensor as “a software model of a physical sensor with similar characteristics, using geometrical data from a ’real’ world model”[4]. Off-line programming systems normally provide means to sim-

[3]Virtual sensors is also a concept in Automatic Control, but has the different meaning of calculating values of (unsensed) process variables from real sensor readings.

[4]In this context, sensors for use in robotics are considered, like vision and laser seam trackers.

ulate work-cells with several robots working in parallel, but simple switch-type sensors typically control their interaction.

Advanced sensors are not common in industrial robotics and therefore it is difficult to find sensor models of real industrial sensors. Most sensor simulation models are strictly of research character and are not soft replicas of any known sensor. An off-line system is a reasonably useful design, prototyping and visualizing tool for shorter delivery time by automatic code generation (Adolfsson, Ng, Olofsgård, Moore, Pu and Wong, 2002). The normal process in off-line robotic programming using an off-line system is to build the work-cell, create trajectories and finally generate a program for the particular target robot. If the work-cell is created with sufficient accuracy and fixtures prevent changes in geometry during the manufacturing process, the program will run correctly on the target system.

In reality, however, such a procedure is not adequate for robots using sensors, especially if they are used for (simulating) trajectory guidance in real-time. Feedback of advanced sensor data must then be reflected in the world model. Virtual sensors provided as black boxes by manufacturers and sold as an option along with the physical sensors, would of course yield the most natural solution. However, there is no indication that this will happen in the near future. The idea of simulating sensors in robotic systems is not new.

In (Chen and Trivedi, 1994), Chen and Trivedi describes a system which main goal is to create a visualization environment to aid the automatic robot programming and off-line programming capabilities of sensor-driven robots. The software system should help the users visualize the motion and reaction of sensor-driven robots. Their main research objective included 1) integration of sensing and motion simulation; 2) integration of planning and simulation; 3) simulation of multiple sensor modalities; 4) compatibility of operation in real and simulation modes, i.e., switching between real and virtual modes occurs at lower level of control and is transparent to the operator; 5) exchangeability of software tools and hardware devices between real and virtual world operations, e.g., a real robot controller should be able to control the virtual robot, and the perceptual and motor commands and routines generated in virtual environment should be able to control the real robot as well; and 6) functionality as a stand alone and also as a system interfaced with a real robotic system.

In the system, sensory information is generated (simulated) by the system upon receiving sensing commands. The sensing simulation is based on the sensor's type, position and orientation, process parameters, and the status of the workpiece. A 3-D object data structure is defined to specify structural relationships and kinematics of the components. The ability to switch from virtual world operation to real world operation and vice versa ensures 1) *compatibility* of the system operation in real and simulation modes, and 2) *exchangeability* of software tools and hardware devices (Chen and Trivedi, 1994).

The system had three modes. In the *Operator Interface/Monitor Mode*, the system received the control signals from the real controller of the robot, and displayed real-time graphics simulation of the robot's motion and sensing. The *Real Controller/Virtual Robot Mode* allowed the operator to control the virtual robot using the real controller without running the real robot. In the *Off-Line Visualization Mode*, the system acted as a stand alone simulation and visualization environment. This mode allowed the operator to create and examine new robots and their work environments before real production and construction, and also allowed for development and testing of new algorithms (Chen and Trivedi, 1994).

Virtual sensors are at least as important as virtual robots. A non-tactile sensor, for example a camera or laser (Cederberg, Olsson and Bolmsjö, 2002*b*), is normally more easily simulated than a tactile sensor (Li, Wang and Ho, 1998; Li and Wang, 1999), especially when the tactile sensor measures forces in a process that includes removal of materials.

5.3.2 Sensor fusion

Sensor fusion is defined by (Hall and Linas, 2001) as: "A process dealing with the association, correlation, and combination of data and information from single and multiple sources to achieve refined position and identity, estimates, and complete and timely assessments of situations and threats, and their significance". This definition is appropriate for the kind of sensor fusion technology that was originally developed in military applications research. But there are wider and more recent definitions than the original one from U.S DoD. Hence, in (Steinberg, Bowman and White, 1999), sensor fusion is "the process of combining data or information to estimate or predict entity states".

Independent of definition, sensing and gathering of environmental information make up the first step and one of the most fundamental tasks in building intelligent Human-Computer-Interaction (HCI) systems. The purpose of sensor fusion is to sense the environmental information of its users or the users' own activity information (Wu, 2003). According to Wu, this purpose ranges from deploying suitable sensors to detect the interesting phenomena or variables, extracting necessary features and combining these information pieces together.

Sensor fusion is important in several research areas. In *context-aware computing*, the ultimate goal is to have computers understand the real world. The result would be that human beings and computers could interact at a higher level of abstraction, for instance in hospitals (Munoz, Rodriguez, Favela, Martinez-Garcia and Gonzalez, 2003) or when using mobile phones (Myers and Beigl, 2003).

In (Brooks and Iyengar, 1997), sensor fusion has been divided into three classes: *complementary sensors*, *competitive sensors* and *cooperative sensors*. The authors give definitions and examples:

- *Complementary sensors* are not dependent on each other, and can be merged to form a more complete picture of the environment. An example is a set of radar stations covering non-overlapping geographic regions. Since these sensors do not conflict with each other, they are easily implemented.
- *Competitive sensors* provide redundant information about the environment. An example is that three identical radar units can tolerate the failure of one unit. It is a challenging general problem that involves interpreting conflicting readings.
- *Cooperative sensors* work together to mobilize information that none of the individual sensors can provide. This is exemplified by the case with two cameras in stereo for 3D vision. It is a type of fusion that depends on the affected devices' details and cannot be approached as a general problem.

For those specially interested in sensor data fusion, (Brady, 1989) and (Hall and Linas, 2001) may be of interest. Even if an application only handles one sensor at a time, packaging of components and information may still be one of decisive importance for the result of production improvement.

5.4 High-level control of industrial robots with examples

By default, commercial robot environments are proprietary and closed systems. Normally, they are customized by the vendor of the system to meet certain application demands. A more flexible production, however, demands a comprehensive view of a work-cell or even a shop floor, where optimization of controllable variables must be taken on system level, outside any robot, sensor or other specific part of the system.

High-level industrial robot control deals with similar problems as telerobotics (Hirzinger, Brunner, Koeppe and Vogel, 1997). Learnings from DLR's ROTEX, the first remotely controlled robot in space, has been applied to task-level programming (Brunner et al., 1995; Landzettel et al., 2000; Landzettel et al., 2001) and to teleconsultation by telepresence in medicine (Wei, Arbter and Hirzinger, 1997).

Experiences from space robotics have had a tremendous impact on robotics in general. Telerobotics depend on virtual environments for simulation and training, automatic control, dynamics, haptics and robotics. At the Stanford Telerobotics Lab, the core belief is that robots can be great tools, but human intelligence “remains unsurpassed in guiding robots”, and this is also the author’s opinion. Constructing robots that are fully autonomous is, at least in unstructured environments, not possible today.

Sensors is a way to cope with accuracy problems and moving from large to low volumes with less try outs will generally increase the use of sensors (Bolmsjö, Olsson and Cederberg, 2002). But today’s robot simulation systems are not well suited to handle advanced sensor-driven applications. Sensors are basically not used by the industry because of lack of simulation and run-time tools and the industrial robotics vendors unwillingness to open up their control systems. A proposed OCS architecture can be studied in (Nilsson, 1996; Johansson, Robertsson, Nilsson, Brogardh, Cederberg, Olsson, Olsson and Bolmsjö, 2004).

5.4.1 Example: A surgical robot system

The surgical robot system RobaCKa used for maxillofacial[5] and craniofacial[6] treatment by “automation” in (Raczkowsky, Däuber, Engel, Hoppe, Korb, Schorr, Hassfeld and Wörn, 2003) is custom-made, since it was impossible to use a standard robot from any of the major vendors. RobaCKa enables the surgeon to carry out precise bone cuts for bone repositioning. The problem of finding the correct robot position relative to the patient is similar to the problem of finding a suitable path in robot arc welding:

1. Feasibility of the robot path
2. No proximity to singularities
3. No collisions of robot and patient
4. No collisions of robot arm segments
5. Consideration of surgical access path

[5]Relating to the jaws and the face.
[6]Relating to both the cranium and the face.

The RobaCKa system consists of a modified Stäubli RX90 robot with a robot control system running V+. The system relies on input from a force-/torque sensor, an infrared navigation system Polaris and an inspection camera and sensor data processed on a Sensor-PC running RT/Linux. Furthermore, a graphical user interface with a pointer-based human-machine interface is used.

To support the surgeon in planning and intraoperative realization an augmented reality system PROBARIS (Hoppe, Kuebler, Raczkowsky, Woern and Hassfeld, 2002) has been developed. The system allows overlaying the operating field with planning data and other information.

5.4.2 Example: A meat-processing robot system

Food Science Australia (FSA) has developed several applications for the meat industry that require the robot to have quick responses during both on-line operation and off-line communication (Li, Ring, MacRae and Hinsch, 2003). However, the authors establish the fact that "commercial robots have not been designed with the operational response common among other types of 'real-time control' equipment". The dynamic response of industrial robots is recognized as a critical factor when determining their use in meat industry.

The particular system studied in this case was a robotic beef carcass splitter. The components integrated to the application were an ABB IRB 6600 robot with a standard S4C+ controller, an ultrasound image-processing unit and a remote primary control unit responsible for the majority of the system-processing. A simple ABB RAPID (controller resident program) routine was loaded on to the S4C+ controller that performed specific move operations. The RAPID program executed on the robot received update position data from a remote computer in real-time[7] according to the schedule proposed in (Cederberg, Olsson and Bolmsjö, 2002*a*). In this application the position of the backbone was detected via analysis of images from an off-the-shelf medical ultrasound unit.

The motivation to use a commercial industrial robot rather than purpose-built manipulator was that the former possess proven positioning performance, has sound safety records, has operational reliability and have replacement parts and service technicians readily available. Furthermore, FSA has found that the development of an application that integrates an industrial robot reduces development cycle time for an application significantly, and is more economical, especially since a single robot can be used in the development of multiple applications.

[7]Real-time means that the RAPID program running on the robot controller accepts new joint values from a external computer with a frequency of 5-10Hz.

However, the robot response to position data sent varied with the velocity and actual acceleration performance of the robot and with the distance to a singularity configuration. Li, Ring, MacRae and Hinsch conclude that adapting an industrial robot directly to an application is “extremely challenging” if real-time or quick response operation is required (Li et al., 2003).

5.4.3 Example: Experiments using an open control architecture

A platform for fast external sensor integration to an industrial robot control system (ABB S4C+) has been used to interface an ABB IRB6400 robot equipped with a force sensor and a grinding tool (Johansson et al., 2004; Blomdell, Bolmsjö, Brogårdh, Cederberg, Isaksson, Johansson, Haage, Nilsson, Olsson, Olsson, Robertsson and Wang, 2004). The experiments have been performed at Lund University and at Kranendonk, an enterprise in the Netherlands. The special grinding tool has been developed at KU in Leuven, Belgium. The sensor interface main features are

- a shared memory interface to the built-in motion control, enabling fast interaction with external sensors in both hard and soft real-time,
- compensation at low-level propagates to higher level of execution and control,
- system and safety supervision and other standard controller features (IO, RAPID, etc.) are still preserved, and
- add-ons to the original controller can be engineered by both standard and state of the art engineering tools.

The sensor platform is an open experimental platform for robotics research and can handle problems with different need of bandwidth. Experience from the platform confirms that the design is appropriate and that software and control need to be tightly integrated.

5.5 Conclusions

High-level control is an area of research that definitely will grow in the near future. Manufacturing of one-off products creates a spectrum of questions that will need creative solutions which traditional local control fails to deliver. The decision to possibly use AI for path planning, process planning, etc., should be based on the particular application control needs in terms of precision, repeatability and speed. In some cases, exact algorithms with calculated worst-case exponential time growth might behave quite well[8].

Sensors are vital to high-level control, but the potential in using them may be hampered by the introduced uncertainty and increased system complexity. Simulations of manufacturing systems that include sensors is therefore needed to find robust and safe solutions. Virtual sensors with a behavior that conforms with their real counterparts must be developed. Sensor fusion is an important research area of its own. In this thesis, complementary sensors have been used.

The three examples clearly show the need for open control systems to be able to automate typical small batch production systems. Industrial robots of today do not generally support control outside the standard system, which normally demands an off-line program. Industrial robot vendors have not been especially responsive to provide solutions for applications for which it is necessary to let the robot's end tool position reflect a real-time output from one or more sensors.

Naturally, safety and technical issues (Hissam and Klein, 2004) are good reasons to reject end user influence tight control loops, but it is probably not the only reason. Today, most vendors try to control the whole market chain, not only by selling the manipulator, but also by exerting control over installation and service.

[8]See for instance (Andersson, 2003), where the exact algorithms presented not only produce better solutions than the traditional heuristic methods, but also, indeed, seem to perform surprisingly fast according to the measurements on the presented implementations.

Chapter 6

Motivation

6.1 Introduction

The business potential for robotized production has been high in recent time, with lower robots prices, higher reliability and accuracy both respecting repeatable positional accuracy and in absolute terms. Industrial robots are increasingly becoming an integral part of manufacturing strategies. Industrial robots are used for a wide spectrum of applications: material handling, assembly, spray painting, welding and product inspection[1]. There are several reasons to use robots, for instance

- fast implementation times,
- increased manufacturing performance and output rate,
- enhanced quality,
- eliminated dangerous and undesirable jobs, and
- reduced labor cost.

According to the UN, robot orders in first half of 2003 were up by 26% to the highest level ever recorded (United Nations, 2003). 80 000 robots were sold between January and June. According to the same source, a robot sold in 2002 would have cost less than a fifth of what it would have cost in 1990. The UN report also states that the usual pay-back period is as short as 1-2 years. Moreover, the

[1]A new potential market is underground construction where robots are expected to be used for a number of applications like drilling in rock and concrete, as well as shotcreting.

report states that the price of robots in Germany relative to labor costs has dropped from 100 in 1990 to 17 in 2002 taking into account the radically improved performance of robots.

Regarding its profitability, it may be asked why robotics investment not even greater? In the UN report, UNECE claims that robot systems still are so complicated that potential buyers need “sufficient in-house technological know-how as well as a thorough comprehension of their production processes” to benefit from the investments.

Another reason is that there are a few dominating application areas, i.e. arc welding and material handling serving a single type of end user, like the automotive industry. A problem with this dominance is that integration of robots into other areas may be hampered since these applications have other demands measured in speed, accuracy or flexibility such as fast changeover to other products and easy operation. Traditionally, most other application areas have not been considered important business segments for the robotics manufacturers, which therefore have not provided solutions. There is a challenge to robot manufacturers to evaluate the potential of robots in other areas of manufacturing.

Finally, the shift in production batch sizes is an important factor. As mass production of large-batch items moves to less labor-expensive countries, many high-developed countries will find that the key to future manufacturing success is to move to production of many smaller batches of different items. However, fast product changes along with customization and optimized design using new materials and manufacturing processes put greater demand on manufacturing operations with respect to control performance and productivity as well as quality. This inevitable change to small batch volumes and one-off manufacturing of products is also the main focus in this thesis.

6.2 Manufacturing of one-off products

Small batch and one-off production systems are rarely able to use robots efficiently. The investment in an industrial robot system and the time taken to program the robots are usually too long and costly compared to the financial benefits and actual throughput in material. Another reason is that the cost of necessary fixtures and clamping becomes high compared to the number of products to be made. Sometimes, for instance in shipyards, it may even be impossible to use fixtures because of the size of the parts to be welded. Finally, some products are unique, expensive and may have such a high material cost compared to the manufacturing cost that production mistakes cannot be afforded.

The use of sensors is a promising way of meeting the demands of this situation. The use of robots generally requires an integrated approach where product data defined within a CAD[2] environment is taken as input and applied within a RSA software that enables modeling, simulation and programming of robot operations.

However, traditional off-line programming systems are not suited to simulations that include sensor-guided robots. The normal process in off-line robotic programming using a RSA is to build the work-cell, create trajectories and finally generate a program for the particular target robot. Generally, the program will run correctly on the target system if the work-cell is created with sufficient accuracy and fixtures prevent changes in geometry during the manufacturing. For robots using sensors, however, such a procedure is inadequate if they are used for trajectory guidance in real-time.

There is no trivial way to simulate the behavior of sensors and sensor-controlled robots. Most (robot) feedback systems are implemented as a local loop, which only considers the specific instructions used to define the robot task as a set of motions. Since a sensor-guided robot's trajectory may choose a trajectory somehow different from the nominal and static trajectory that is produced by the traditional off-line systems, creating and applying a nominal pre-calculated program is not possible.

6.3 Conceptual ideas

Considering the number of robots used in industrial automation, the use of advanced sensors such as vision, laser scanners or force/tactile sensors is still not comprehensive. One area where sensors are important is arc welding where products based on new materials decrease the overall dimension (plate thickness) and increase the general need for keeping tight tolerances during welding. New processes such as laser welding further emphasize this.

Off-line programming systems are not suited to simulation that includes sensor-guided robots. But given the new opportunities for precision and calculation of speed allowed by specialized software and hardware closely collaborating with open control systems, sensors are likely to appear more often in new manufacturing systems that include robots in small batch and one-off manufacturing systems.

A promising way of meeting the demands of this situation is to use a simulation environment to test a work-cell that includes a sensor-controlled robot before its actual operation in real life. The idea of simulating the robot task with its use of

[2]Computer Aided Design

sensors is interesting since robots equipped with sensors with real-time connection to trajectory generation may later lead to malfunction of the real robot cell. Problems that may occur are numerous including out-of-joint limits, collisions with objects in the workspace, movement into singularities with resulting robot configuration changes, etc.

Thus, instead of feeding back instructions, the static image of the model normally used in today's programs, sensor information can be fed back to a model representation of the task. Real-time sensor feedback to the world model allows the information from sensors to be used to actually update the world model, including updating object positions in real time as required, or creating objects not included beforehand. Through this mechanism, the use of sensors can be validated in a simulated environment in the same way that similar tests are carried out in a real, physical set-up.

The idea is to control the robot by specifying the task rather than by using a set of predefined motions and logic, thus providing a higher level of abstraction in the formulation of instructions. Examples of how this can be used include resolution of singularity issues, necessary trajectory planning, process adaptation due to changing conditions and environment, motion planning that considers real-time update of geometrical objects in the world model, etc. As indicated by these examples, lack of coupling between the running robot program and the virtual model is not acceptable since the system must have the ability to emulate and act on possible situations in the real environment, which of course are not fully known in advance.

Also, by receiving sensor information in the same way regardless of whether a real or virtual sensor is producing the data, dynamic effects originating from the internal system relating to time delays, information flow and bandwidth are also taken into account. The interface between sensor and application facilitates these properties. This system structure makes it possible to test and validate the operation of a sensor-guided robot system during simulation, as well as to run the system on the shop floor without any change to the task program.

An important feature of such a task-oriented programming environment and run-time system should be usability. This is normally a foreseen feature of many technical systems and almost never considered of any interest, especially by technically oriented professionals, that prevent the systems to be used at all or to a limited extent. It is important that the user has a feeling of being in control. Instead of having to rely on intricate and automated system decisions on how things should be done, and what process variables should be chosen, the industrial acceptance will likely be higher if these conclusions are taken by the staff, while letting the automated system handle and conceal the underlying technology to create the necessary trajectories. Of course, when or if reliable intelligent decisions can be taken by machines and when or if this technique is accepted by human beings, machine responsibility may increase in the future and higher level of task abstractions may be reached.

The above mentioned ideas are all needed but must be supported by industrial control systems that allow trajectories to be sent in real-time during the process instead of, as done today, demanding a predefined and static trajectory. These open control systems exist today mainly for research purposes, but will most likely be commercially available in a five-year perspective.

6.4 Conclusions

We can conclude that there are some important features missing in traditional robotic programming systems. They need to be found for extending the usability of industrial robots to small-batch and one-off production systems:

- Traditional off-line programs are inadequate in serving small batch and one-off production systems. New systems are required that allow users to develop, simulate and run applications that includes sensors and sensor-controlled robots.
- Virtual sensors need to be developed to support these systems with adequate models, similarly to how traditional off-line systems are supported by robot libraries.
- Virtual sensor interfaces should hide the underlying logic of the sensor, virtual or real, from the user. The application using the virtual sensor should focus on the task, and not on whether a virtual or real sensor is used.
- Task-oriented programming systems that let the user keep the upper hand, not leaving difficult and important decisions to a machine that because of insufficient or unreliable technology is unable to take optimal decisions.
- Open Control Systems that allow real-time trajectory generation for industrial robots and let a high-level control system make necessary comprehensive decisions.

Chapter 7

Contribution

7.1 Introduction

Most contributions from other work in the task-level and task-oriented programming areas are generic solutions having complexity issues and limitations such as exponential growth of planning algorithms. In this work, planning is the responsibility of the user, who is supposed to have the expertise necessary to define the problem, and to choose, order and initialize subtasks. This strategy does, however, not impose that it would be impossible to automate planning in particular situations or for a certain application. Perhaps arc welding is such an application, but it has not been studied.

7.2 Scope and limitations

The author's contribution consists of identifying problem areas and resolving issues preventing a successful implementation of high-level control of industrial robots. The technique presented is intended for small batch and one-off manufacturing systems for which it should be advantageous to use sensor-controlled robots.

The main objective, described in section 7.3, has been to develop a semiautomatic model for task-oriented programming applied to sensor-controlled robotic arc welding. To reach this goal, three subsystems have been developed

1. a virtual sensor,
2. generic interfaces to sensors/robots, and
3. an underlying run-time library to real and virtual sensors/robots.

7.2.1 Development of virtual sensors/robots

Scope: To be able to simulate the use of sensors/robots it is necessary to develop virtual sensors/robots that behave sufficiently well compared to real sensors/robots and conform geometrically. Using these virtual components, different scenarios can be simulated without compromising safety and without having to use the actual equipment on the shop floor. In traditional off-line programming, libraries describing kinematical properties usually exist for standard industrial robots.

Limitation: A virtual laser tracker has been developed based on the M-Spot sensor from ServoRobot, Inc. The real sensor is of a type commonly used in the arc welding industry and uses a triangulation method for depth measurements. The sensor is validated both statically and dynamically by matching it with a commercial sensor through measurements in setups and by comparing a welding application performed in a real and a virtual work-cell created with a RSA. The experimental results successfully validate its performance. In this context, a virtual sensor is a software model of a physical sensor with similar characteristics, using geometrical and/or process-specific data from a computerized model of a real work-cell.

7.2.2 Generic interfaces to robots and sensors

Scope: A well defined virtual sensor/robot should be exchangeable with a real sensor/robot without affecting the application(s) using it. This is important since every mode added to the program will make it error prone and difficult to maintain. Seamless exchange of virtual and real components is also crucial to run experiments with a mix of virtual and real components. For instance, a real robot can be used along with a simulated sensor acting in the virtual environment. The simulated sensor may discover virtual obstacles that the physical robot avoids, despite the fact that no physical obstacles exist. Such arrangements allow the user to check and debug different parts of the system during the development of task objects.

Limitation: A generic interface to the M-Spot sensor and the virtual sensor has been developed. The interface covers a large subset (but not all) of the possible interaction between the client using the sensor capabilities and the sensor itself.

A generic interface to the real and simulated robot has also been developed that allows the application to send joint values in real-time (5-10Hz). The client is unaware of if a real or simulated sensor/robot is used.

7.2.3 Run-time library

Scope: A run-time library is needed to support the building of the different components that constitute an application. Robots, the description of which follows the Denavit and Hartenberg representation are supported. The library handles kinematical relationships, interpolation and functions for manipulating data types such as vectors, matrices, homogeneous transformations and quaternions. Furthermore, implemented events, communication, threads and synchronization routines may be needed to efficiently handle interaction between components.

Limitation: The library has been tested with the ABB2400/16 robot and the M-Spot sensor[1].

7.3 Main objective: A semiautomatic task-oriented programming model

Scope: The proposed approach could be considered semiautomatic task-oriented programming in the sense that the user manually defines the task by choosing a sequence from a set of predefined sub-tasks. One or more sequences of objects constitute the robot program needed to execute the task. A sub-task can be thought of as an "object". Several instances of objects may co-exist independently during the task. When the sequence is defined, the task execution is autonomous.

This approach handles a task differently than the existing parameterization technique, which is based on similarities in shape rather than, as is proposed here, in sub-tasks.

Limitation: Current planning research has generally not reached such a mature level that it can be used in industrial robotics. Still, human beings better accomplish perception of the work-cell – geometrical and process understanding as well as planning. The semiautomatic task-oriented programming model presented is tested on the weld application described in this thesis.

[1]These are the robot and sensor available for research in our robotics lab.

7.4 System philosophy

Previous work at the department has over the past years developed ideas on themes related to this thesis. In (Brink, Olsson, and Bolmsjö, 1995), a task-oriented robot programming method focusing on tasks connected to the objects in the robot work-cell, is presented. Tasks are described as states of objects and their dependencies, and the method described should avoid an Achilles heel in task-level programming, the intricate problem to describe complex systems in a language similar to written natural languages. The robot work-cell is expressed as a discrete event system where attributes (conditions) of the parts in the environment only change at a discrete set of points in time, and when certain conditions are fulfilled. But as with other generic methods, its weakness lies in the fact that although there is a finite number of possible states, the combinatorial explosion could make the number very large.

A theoretical description of a framework for higher level of control and autonomy is described in (Bolmsjö, Olsson and Brink, 1999). The research stresses the importance and advantage of a high-level task control system and the control structure layout is given. The framework uses a world model and virtual sensors are briefly mentioned.

In (Olsson, Cederberg and Bolmsjö, 1999*a*) and (Olsson, Cederberg and Bolmsjö, 1999*b*), a system is presented that integrates a simulation and execution environment for industrial robot tasks. Sensor feedback is used to update a virtual work-cell model and sub-tasks are autonomously executed based on the information currently available from the virtual model. The idea of the work presented in this paper was to add an interface to a commercial RSA, capable of interacting with the physical system in real-time. The authors refer to virtual sensors and sensor interfaces but at the time the paper was written, these ideas were fairly new. Nevertheless, this RSA centric solution, further developed in (Olsson, Cederberg and Bolmsjö, 2002) was an inspiring source for the development of the platform described here.

Research in industrial robotics implicitly means that any trust-worthy solution should be based on an engineering strategy even if the experimental work sometimes has to be done on a somewhat simplified experimental platform. It is natural for researchers to favor generic solutions, and they are normally also the most interesting. However, good engineering solutions in a particular domain are foreseen because of unsolvable difficulties with a general solution.

7.5 General system structure

A system structure to support high-level control of sensor-controlled industrial robots to support small batch and one-off production systems has been developed. A nominal model is defined and exported. The run-time model uses the nominal model along with kinematical robot models to create a world model. All tasks are then executed on this world model, which communicates during run-time through sensor and robot interfaces with real and simulated robots and sensors, see Figures 7.1 and 7.2.

The described system structure contains components developed by the author mixed with other components. In section 7.2, the scope and limitations of the thesis have been stated.

7.5.1 Definition of the nominal model

The technique of modeling work-cell components in CAD environments and creating off-line robot programs in RSAs is mature and applied by users of industrial robots today. Process data may originate from process databases, for example a weld database and from the CAD models. As we today are used to model processes in CAD/RSAs, these systems are a natural starting point for generating models. The model created is the *nominal* model, i.e. it describes our work-cell as well as we know it before information from sensors gives us reasons to change it.

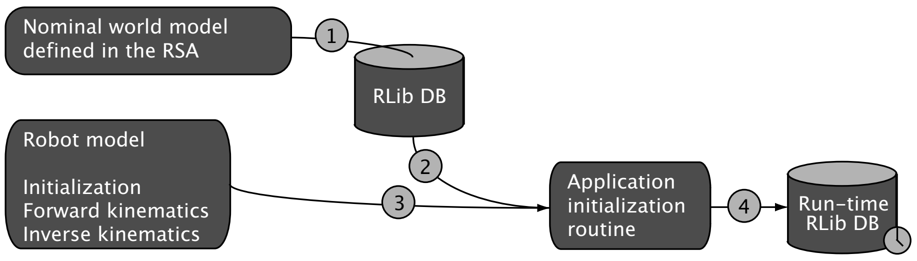


Figure 7.1: *A nominal model is defined and exported (1). The run-time library uses the nominal model along with kinematical robot models (2, 3) to create a world model (4). All tasks are then executed on this world model.*

Kinematic data from CAD/RSAs are related in an order that is considered efficient for the particular application and its purpose and is saved in a proprietary format or a common CAD format. Normally, these expensive applications also give the user the opportunity to view data via some functionality that recursively traverses the internal structures of the application. By either using this option or by traversing documented formats, data relevant to kinematic relationships and process relationships, when appropriate, can be collected. The developed system provides an API that makes it possible to save relevant information in a system-dependent, but CAD/RSA-independent way. After using the API, kinematic data is saved in a CAD/RSA-independent format but it may still not be related in a feasible way for its new purpose – realistic simulation and run-time handling of sensor-controlled robots. The system API therefore also let us change kinematic relationships "manually", i.e. with a few lines of code, after data are read from the system format into our application.

Process data from the CAD/RSA environment along with manually created data is also collected. Some process data can be collected during the traverse of internal structures, but not all. Sensor-specific data are not common in CAD/RSAs today, since they do not normally support sensors or modeling of sensors. Process data may be different for various parts of the workpiece, and the system take this into account by creating objects that relate to the process parts.

After kinematic and process data are made available to the system, the CAD/RSAs are mainly used as hosts for virtual sensors/robots and to simulate the running operations either as a full virtual process without any physical parts involved or to let the user monitor the real process. Despite their often extensive functionality CAD/RSAs may be more or less suitable for this purpose[2]. Another general problem is that these applications tend to be quite large and their multipurpose monolithic structure may harm their execution speed significantly. To overcome these and other problems it is suggested that efficient graphical environments are built and used instead.

7.5.2 The run-time model

At this point, nominal kinematic data, process and sensor data are well known to the system and available in appropriate structures. An efficient run-time library, a library API and a set of objects, serve as a foundation to which applications can be built upon. Today's RSAs are not suitable as engines in the run-time environment either. This is mainly because of their complicated and single-threaded structure and their different layers of APIs with functionality only reachable from vendor proprietary languages. One has to remember that these applications are optimized

[2]As an example, the graphics update process within the RSA may not be fully controllable from a user standpoint, and this significantly hampers the accuracy of measurements done by virtual sensors inside the graphical environment.

for simulation purposes only, and are not built for run-time usage. A better solution is a library where one simple API gives access to the required functionality.

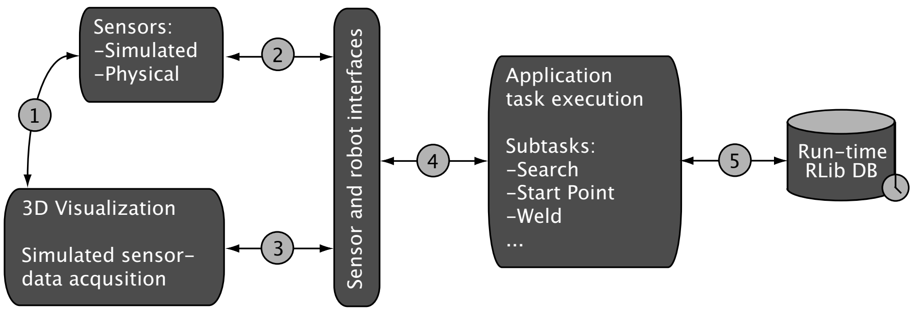


Figure 7.2: *The world model communicates during run-time through sensor and robot interfaces (4) to real and simulated robots and sensors (2). The task is visualized in 3D (3). When simulated sensors are present, sensor data acquisition is made in the 3D model (1). Changes in the world model are saved in the run-time database (5).*

The library consists of all or at least most of the functionality needed to create an application, including interpolation routines that can act upon the kinematical data that is saved in the database, service routines such as conversions between different mathematical representations of orientations, communication routines and others. Interpolation routines will be needed since the position and orientation of the tool is continuously calculated by the high-level control system, outside the traditional robot-control system.

It is important that this library is used both for simulation and for running the actual system. The application built will not have to act differently on virtual and real components, and will not even know which components it is operating. This is a major feature in the high-level control system that creates more realistic simulations, less errors and simpler applications.

7.5.3 Supporting libraries

A library of robots and sensors needed in the model is also required. The robot library is preferably based on routines provided by the robot manufacturer, “black boxes”, that will help the high-level control system to calculate correct accelerations, speeds, and limitations in the different situations. Accurate trajectory performance is important in the industry. Laser welding, for instance, demands very accurate performance to obtain a good weld quality. The black box includes an accurate dynamic model of the robot consisting of mechanical[3], electrical/digital and robot control models.

The sensor library is also based on manufacturer routines and each virtual sensor is essentially built around this black box consisting of routines providing the same interface and accuracy as the real sensor. Also, limitations in hardware must be taken into account, for example bandwidth and, if possible, failures that will occur with some measured or expected probability. Force-sensors and other tactile sensors will be more difficult, but not impossible to simulate. In cases where there is a major difference between reality and simulation, this discrepancy can be quantified.

It is realistic that generic robot/sensor interfaces are provided for groups of robots/sensors with similar characteristics and usability. By using generic interfaces, the user will have the opportunity to compare similar solutions from different manufacturers. Virtual components are also valuable before a work-cell is built, or when different alternatives are to be assessed. In other words, two dimensions of generic interfaces are needed: one that hides if a component is virtual or real, and one that groups similar components together, see Figure 7.3.

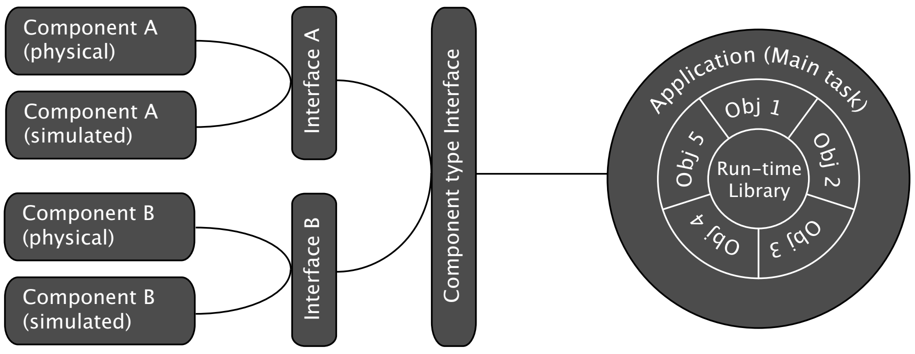


Figure 7.3: *Two dimensions of generic interfaces, one that hides if a component is virtual or real, and one that groups similar components together.*

7.5.4 Process-oriented parameterization

In normal industrial robot programming, the path is created and the process is based upon the decided path. Here a process-oriented method is proposed, based on the hypothesis that it is natural to focus on the process, since the path is in reality merely a result of process needs. Another benefit of choosing process focus is

[3]Calculated, for instance, using a non-linear finite element formulation.

that it automatically leads us into task-oriented thoughts, which in turn can be split in sub-tasks, one for each part of the process with similar process-characteristics. By carefully choosing and encapsulating the information needed to execute a subtask, this kind of encapsulation, or *object*[4] can be re-used whenever the actual subtask appears. For each subtask, an instance of the object is created, see Figure 7.4.

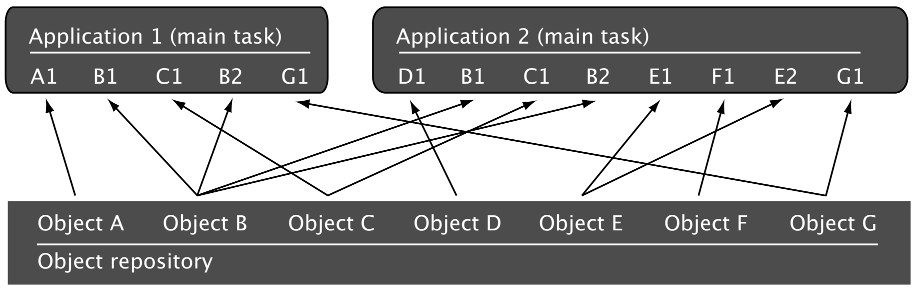

Application 1 (main task)
A1 B1 C1 B2 G1
Application 2 (main task)
D1 B1 C1 B2 E1 F1 E2 G1
Object A Object B Object C Object D Object E Object F Object G
Object repository

Figure 7.4: *Applications are built upon object instances taken from the object repository. Each object can be re-used whenever the actual subtask appears.*

The path is often created during run-time and is based on sensor readings. This put clear demands of openness of the robot control system. Luckily, in many concrete industrial applications, a nominal path is normally known before hand and a deviation from the nominal path over a specific threshold would be considered an error that can stop the process. For those processes, a control system that allows a certain deviation from a nominal path will probably suffice.

7.5.5 Visualization of work-cell components

The system also consists of APIs to visualize general components besides robots and sensors in the work-cell. Thus, an application built with the system is independent of visualization software used and can basically be seen as a controller in the *model-view-controller* paradigm. Portions of the model are spread in different parts of the system, in the graphical environment, in the process database, in kinematical and other databases. The view is, not surprisingly, appearing in the visualization software. As mentioned before, each object has responsibility over a subtask and may occur in as many instances as needed to fulfill the particular task.

[4]The name is given because of the resemblance with object classes in Object-Oriented Programming, OOP.

7.5.6 Object aspects

When an application is running, one of the objects normally has the actual control. The underlying run-time system is the framework that essentially permits high-level control and supports the objects and the main program in all major aspects. The object may create the path on-line-dependent on sensor readings. Since the object encapsulates most of the control of the process it has important duties:

- Initialization and execution
- Sensor-process interaction
- Interaction with other objects
- Graphics interaction
- Interaction with run-time and operating system

Object-initialization and execution

Each object must be initialized with relevant sensor parameters, process variables and world model parameters and these are usually different for each object. Then, the objects are executed in sequence order. Some objects are restricted to start the process, some defines the process end, and other objects can occur in arbitrary order between start- and end objects.

Sensor-process interaction

The objects rely on data from a world model and are bound to output from one or more sensors and a process. A good example is the experimental weld object that is bound to a laser tracker sensor and weld process variables. The chosen level of abstraction is intended to hide sensor-process interaction from the user. The objects' behavior is predetermined by their initialization data and run-time rules.

The coupling between sensor and process makes it possible to create a library of objects that can be re-used for tasks within similar context. In the robotic arc welding experiment, the weld object is invariant of the weld distance. Thus, a single weld object can be used for all possible distances. If two separate welds are necessary to complete a particular task, two separately initialized instances of the weld object are needed.

Object-object interaction

Although objects normally are independent they are part of a continuous task. The weld object, for example, is designed to weld forever unless an error occurs, i.e. sensor readings are indicating a lost seam. To end a weld properly an end-weld object[5] has been chosen, which interpret a lost seam as an normal end-of-weld condition instead of an error.

In this situation it is feasible to stop execution of the weld object close to where the weld is supposed to end without signaling an error condition, whereafter control is returned to the main program, which then invoke the end-weld object. There are, of course, several possible solutions to accomplish transfer between objects.[6]

Object and graphics interaction

The kinematic definition of the nominal work-cell is separated from the execution of tasks, and can preferably be made in a RSA. The work-cell can then be saved in a RSA-independent text-based format. When a program is started, the nominal world-model can be imported. The model describes the kinematic relationships between different objects (including robots) in the work-cell.

During execution, the user interface and the motion control logic should be separated whenever possible[7]. An advantage of separating graphics and logic is that it, under execution, becomes possible to accomplish high-level control using a real-time operating system, which is needed to actually run the system as a component sharing resources with an open control system. However, it is beyond the scope of this thesis to propose actual hardware solutions.

Object and run-time system interaction

The implementation of the underlying run-time system, *RLib*, calls the objects regularly during execution if the object uses interrupt procedures for tags and tasks. RLib calls the task procedure before and after the task is executed. The call includes the current task state. The tag procedure is called before, during and after

[5]This is an implementation decision. Another solution could be to introduce an end-weld mode in the weld object.

[6]The implementation uses procedure hooks that are called every interpolation step, i.e. each time the object itself is called from the underlying run-time system.

[7]Although it is not realistic to use a RSA as a container for virtual sensors during execution of a system with hard real-time demands, it has been used in the experiments.

each tag is processed. Here, “during” denotes an interrupt after each interpolation step when executing the tag.

RLib holds the state of the task in a structure. The structure holds the current path and tag[8] along with other information pertinent to the current motion and current state, and is passed to the task- and tag procedures. By using this information, the object can read the current motion state and use it after the main program has invoked it.

Run-time system and operating system interaction

A problem with today’s off-line systems is that logic and graphics are mixed and dependent on each other, which results in that actual load on the graphical system strongly affects the control performance. In reality, the performance of embedded systems is usually needed in parallel to high performance 3D graphics. If graphics need to be updated in tight loops which might be the case when virtual sensors are used, today’s commercial RSAs do not provide a realistic solution because of their low and unpredictable performance. Customized solutions are probably required. This does not impose major difficulties taking into account today’s computing potential and available software tools.

7.6 Conclusions

With current programming technique, robotized small batch manufacturing and manufacturing of one-off products demand individual robot programs. As a result, robotized manufacturing is not cost-effective for this type of production. The fact that today’s industrial robot programming is motion-centered probably results from that robots normally just iterate through a predefined and downloaded program. Apart from local adjustments, most robot programs still do not use sensors which output affects the actual path of the tool. The use of sensors has potentially to cope with the different set-ups for one-off products, but the increased system complexity and uncertainty that sensors implicate must then be handled.

The methodology addresses the mapping problem between observable and controllable variables and the focus is on controlling the process/task rather than the motion alone. As indicated above, such motion may include orientation changes of the weld object (with a positioner), changes of poses of the robot and changes of travel speed. In turn, this will require to monitoring issues related to out-of-joint

[8]A tag contains a pose and one or more tags constitute a path. The trajectory is then created from one or more paths.

limits, collisions, configuration changes and singularities, which, during sensor guidance, has to be done in real-time.

The proposed process-sensor oriented methodology parameterizes on sub-tasks. Each sub-task is based on process-sensor data and this information is encapsulated in an *object*. Motion is generated by the object, taking into account limitations imposed by initialization and process restrictions. The ideas have been submitted for publication in (Cederberg, Olsson and Bolmsjö, 2004).

Chapter 8

Experimental system structure

8.1 Introduction

A test bench has been developed based on the general system structure described in Chapter 7 for combining simulated and real components and verifying their functionality, both during simulation and execution. The experimental structure and arc welding application used a laser tracker with control hardware, which existed both as simulated and as real components, a commercial six-axis robot with an unmodified control system, a distance-sensor, which only existed in the simulated world, and two cameras[1]. The cameras were not simulated.

Several features were handled by the system components: remote compilation and loading of a specially designed program to the robot controller, on-line trajectory generation, and on-line collision-tests and singularity detection and avoidance. The commercial robot controller could only execute pre-loaded programs. To be able to execute on-line generated trajectories, the robot program downloaded to the robot was given a generic design that demanded continuous updated joint values to execute. This was very different from the static programs with pre-decided trajectories typically executing on the robot controller.

The arc welding application was assembled as a set of objects, which divided the logic of the arc welding task into sub-tasks. Each object encapsulated sensor-process related data and could potentially occur several times during the application process. Figure 8.1 gives a logical view of system components.

[1]Camera set-up and mathematical treatment of camera output are courtesy of Stefan Adolfsson (stefan.adolfsson@hbg.lth.se).

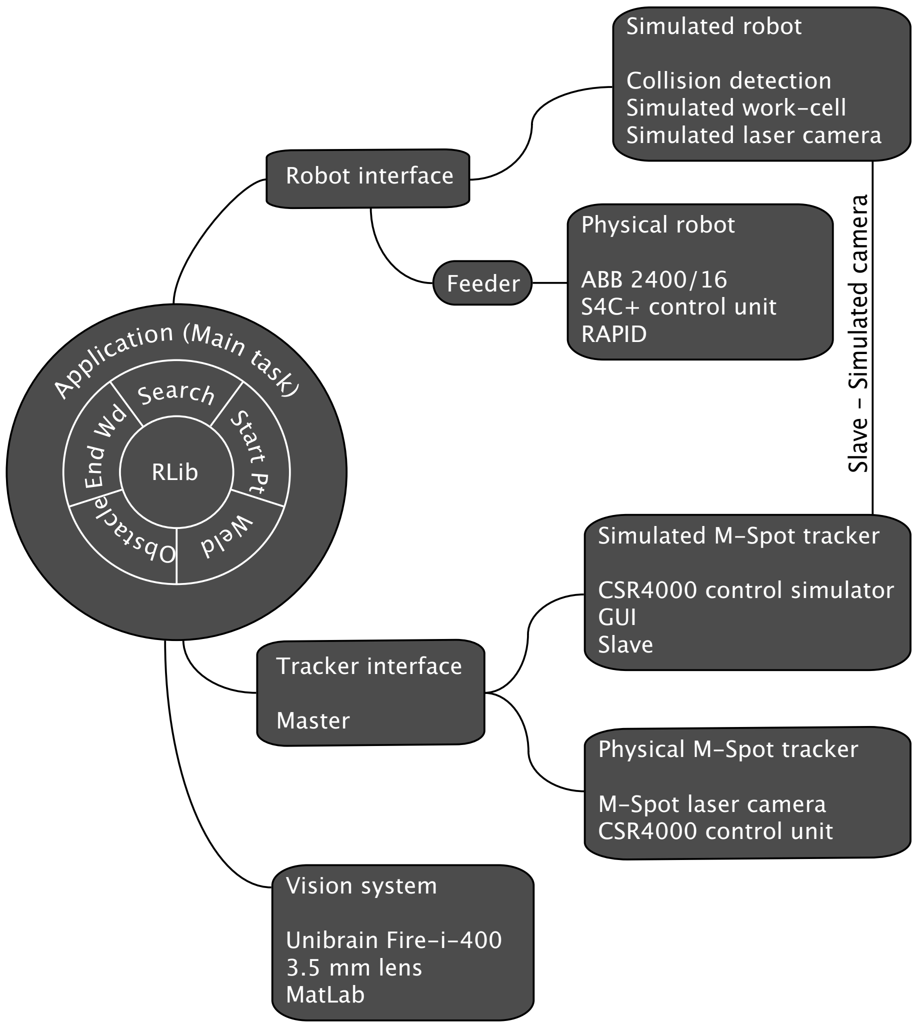


Figure 8.1: *Logical view of system components in the experimental setup.*

8.2 Application

The application was connected to the different sub-systems: the RSA, the tracker, the distance-sensor interface, the robot interface and the“feeder”. The information time rate needed for a tracker-guided robot during arc welding was in the interval of 40-60 ms (20Hz) which made local host TCP/IP communication a suitable choice.

8.3 Feeder

In this particular experiment an ABB 2400/16 robot was used. Its S4C+ control system executed programs written in the ABB RAPID language. To accomplish on-line trajectory generation the “feeder” component routed the joint values from the application to the generic program executing on the robot controller. This system component utilized a robot vendor proprietary protocol, the ABB Robot Application Protocol (RAP), to communicate with the robot controller. Thus, the feeder served as an interface to the robot and hid vendor-dependent protocols from the application. From the application’s point of view, the requirement was to deliver joint values whenever requested by the feeder. The feeder was also responsible for reading the actual position and orientation of the robot.

8.3.1 RAPID language

The program consists of a number of instructions that describe the work of the robot in a Pascal-like syntax. There are specific instructions for the various commands, such as to move the robot or to set an output, etc. There are three types of routines: procedures, functions and trap routines and three kinds of data: constants, variables and persistents. Persistents are variables that can be reached from the outside world. Other features in the language are: routine parameters, arithmetic and logical expressions, automatic error handling, modular programs and multitasking (*ABB Rapid Reference Version 3.2, RAPID Summary*, n.d.).

8.3.2 Remote Procedure Call and External Data Representation

Remote procedure calls are a high-level communication paradigm that allows programmers to write client/server network applications using procedure calls that hide the details of the underlying network. The RPC model is similar to the local procedure call model where the caller places arguments to a procedure in a well-specified location (such as a result register) and transfers control to the procedure. When the caller eventually regains control, it extracts the results of the procedure from the location and continues execution (*IRIX Network Programming Guide*, n.d.). RPC uses XDR to establish uniform representations for data types in order to transfer message data between machines (*AIX Version 4.3 Communications Programming Concepts*, 1997). Sun’s RPC and XDR are freely available on numerous platforms.

The Robot Application Protocol provides a set of services that makes it possible to monitor and control the robot from an external computer. These are grouped into four classes: general management, variable access, file management and program control services. The general management services are support services for all other services, e.g. opening and closeing a connection to a specified server and restart of the controller. RAP is using named variable objects to get information from the robot-system or affect the robot system, e.g. to read and write RAPID defined and predefined system variables and event handling. An event in the system can be subscribed for, and as a result of such subscription, a spontaneous message will asynchronously be sent to the external computer when the event occurs. RAP file management provides the functionality to access files on the memory devices in the robot system, e.g. to open, read, write, close, rename and delete a file (*ABB RAP Protocol Specification 1.05*, n.d.; *ABB RAP Service Specification 1.05*, n.d.; *ABB Ethernet Services 3.0*, n.d.).

8.3.3 High-level remote motion control

Typically, a standard RAPID program needs no invocation from the outside world after execution has been initiated. It is only possible to send data with RAP[2].

A special RAPID program with a designated sequence of move instructions was downloaded to the controller. The RAPID program needed to be designed for letting a remote application control the robot's motion in a master-slave fashion. The relevant part of the program consisted of a loop where move instructions continuously were executed as shown below.

```
!RAPID program executing on robot controller
...
PERS num           pnum  := -1;
PERS num           p0set :=  0;
...
PERS robtarget     p0:=[...];
...
WHILE NOT aborted DO
  WaitUntil p1set <> 0;
  p1set := 0;
  pnum := 0;
  MoveL p0, v, z, tool0;

  WaitUntil p2set <> 0;
  p2set := 0;
  pnum := 1;
```

[2]As oppose to instructions; a limitation that has been circumvented in (Pires and da Costa, 2000) by introducing a switch statement in the RAPID program where each selector defines a predefined and possibly complex service.

```
      MoveL p1, v, z, tool0;

      WaitUntil p3set <> 0;
      p3set := 0;
      pnum := 2;
      MoveL p2, v, z, tool0;

      WaitUntil p0set <> 0;
      p0set := 0;
      pnum := 3;
      MoveL p3, v, z, tool0;
    ENDWHILE
    ...
```

The loop represented a circular buffer of 4 `robtarget` structures `p0`...`p3` which contained position, orientation of tool center point (tcp) and configuration of robot and external axes. The variables were declared as `PERS` (persistent). During execution, these structures were dynamically set (with some latency) with joint values provided by the application. The feeder kept track of which move instruction that was to be executed by the robot controller, i.e. which `robtarget` structure to be updated at a specific time.

The *MoveL* directive refers to a via movement and v and z denoted tcp speed and the `zonedata` structure respectively. `zonedata` was used to specify how a position was to be terminated, i.e. how close to the programmed position the axes had to be before moving towards the next position. For instance, at some point of time during the move from `p1` to `p2`, both `p2` and `p3` had to be known to the robot control system. To be able to prepare for the next movement a fourth point was needed.

Even though a RAP call returns synchronously, there is no guarantee that the `robtarget` structure in RAPID is updated when `writeRobTarget()` returns. Since RAP calls that write data to RAPID variables in practice are asynchronous, there is need for a mechanism to be certain that a particular variable holds the data previously written to it.

This hand-shaking problem was resolved by using busy wait[3] in the RAPID code as well as in the feeder. The `pnum` variable in the running RAPID program was continuously monitored by the feeder and when it eventually became updated in the RAPID program, the next `robtarget` structure update was sent from the feeder to the robot controller.

```
/* Feeder pseudo code to handle */
/* routing of data and hand-shaking */
```

[3] A polling method.

```
/* between application and robot */

static int pnum = -1;

int pnumChanged()
{
  RAPVAR_DATA_TYPE data;

  readRAPIDVar("pnum", data);
  if (data.RAPVAR_DATA_TYPE_u.num != pNum) {
    pNum = data.RAPVAR_DATA_TYPE_u.num;
    return 1;
  }
  return 0;
}


void feedRobTarget(ROBTARGET robTarget)
{
  /* set two pnum’s ahead, i.e. */
  /* if pnum = 0, set p2set = 1 */
  /* in RAPID program */
  int pXset = (pnum + 2) MODULUS 4;
  writeRobTarget(robTarget, pXset);
  writepXset(pXset);
}


void routeJoints()
{
  ROBTARGET   robTarget;
  JOINTVALUES joints;
  do {
    if (pnumChanged()) {
      joints = sendJointRequestToApplication();
      robTarget = kinematicCalculation(joints);
      feedRobTarget(robTarget);
    }
  } while (!aborted);
}


void main()
{
  if (compileProgram() == SUCCESS) {
    connectApplication();
    /* initialize first two robtargets */
    initializeRobTargets();
    startRAPIDExecution();
    routeJoints();
  }
  else
    displayErrors();
}
```

To avoid having the speed of the application exceeding the speed of the robot, the application only sent joint values after receiving a request message from the feeder. If the application was unable to send values upon a request, the robot came to a temporary stop until the application was ready to resume delivery of joint values.

8.3.4 Remote editing and compilation of RAPID programs

Some of the strengths of RAP were shown in the supporting parts of the feeder. Besides handling the real-time issues of routing values from the application to the robot, the feeder provided a RAPID compile environment within *emacs*[4], a well known LISP-based editor. By issuing a compile command in emacs, the RAPID program would get syntax checked and any errors were shown with row, column and error message in a second window. By clicking on the error, the cursor marked the offending line in the RAPID program, see Figure 8.2.

Behind the scene, the actual compilation was performed remotely on the robot system. By using RAP, the RAPID program was sent to the robot controller and was loaded and checked. The errors were saved in a log file on the robot controller and were transferred back to the feeder and displayed to the user in an user-friendly fashion. If the RAPID program passed the syntax check, the user could choose to automatically run it.

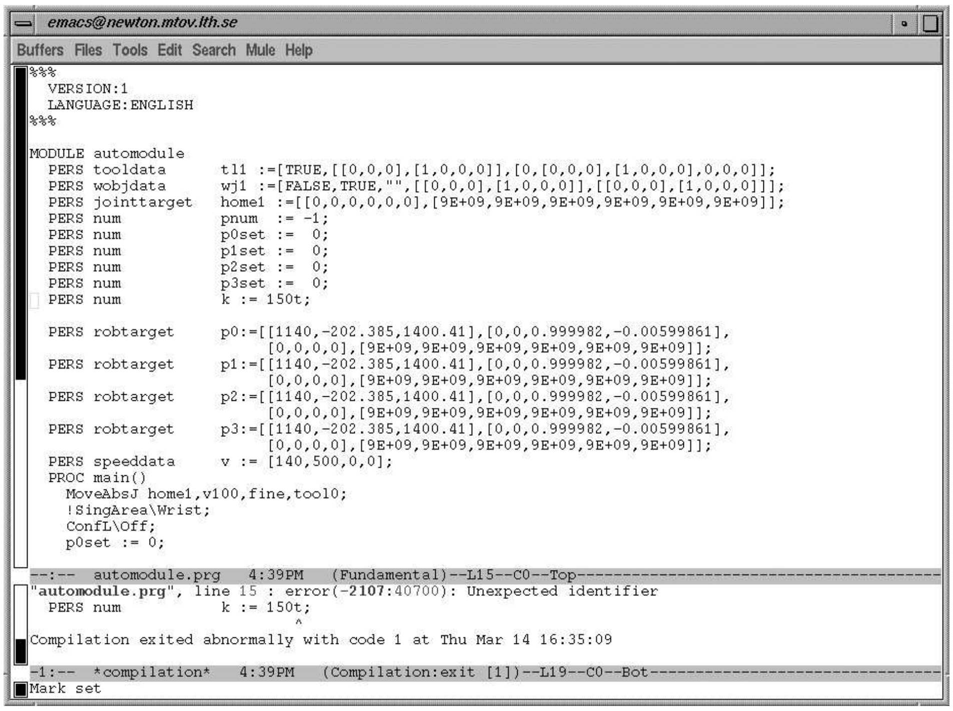
8.4 Stereo cameras

Two Unibrain Fire-i-400 cameras each equipped with a 3.5 mm lens were used to calibrate the nominal world model. The cameras were placed to get a bird view of the work area. Workpiece images were processed in Matlab and the result, a better nominal workpiece pose was written to a file before the application started.

8.5 Tracker

A tracker component was developed to allow applications to interact with the tracker control unit. The tracker interface permitted the application to use a single protocol for communication irrespective of if communicating with the physical

[4]Originally written by Richard Stallman in 1976, as a set of Editor MACroS for the TECO editor. Popular versions today are GNU Emacs, see http://www.gnu.org (also written by Stallman), and its close relative XEmacs.



```
%%%
  VERSION:1
  LANGUAGE:ENGLISH
%%%

MODULE automodule
  PERS tooldata       tl1 :=[TRUE,[[0,0,0],[1,0,0,0]],[0,[0,0,0],[1,0,0,0],0,0,0]];
  PERS wobjdata       wj1 :=[FALSE,TRUE,"",[[0,0,0],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];
  PERS jointtarget    home1 :=[[0,0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
  PERS num            pnum  := -1;
  PERS num            p0set :=  0;
  PERS num            p1set :=  0;
  PERS num            p2set :=  0;
  PERS num            p3set :=  0;
  PERS num            k := 150t;

  PERS robtarget      p0:=[[1140,-202.385,1400.41],[0,0,0.999982,-0.00599861],
                           [0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
  PERS robtarget      p1:=[[1140,-202.385,1400.41],[0,0,0.999982,-0.00599861],
                           [0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
  PERS robtarget      p2:=[[1140,-202.385,1400.41],[0,0,0.999982,-0.00599861],
                           [0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
  PERS robtarget      p3:=[[1140,-202.385,1400.41],[0,0,0.999982,-0.00599861],
                           [0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
  PERS speeddata      v := [140,500,0,0];
  PROC main()
    MoveAbsJ home1,v100,fine,tool0;
    !SingArea\Wrist;
    ConfL\Off;
    p0set := 0;
--:--  automodule.prg   4:39PM   (Fundamental)--L15--C0--Top-------------------------------------
"automodule.prg", line 15 : error(-2107:40700): Unexpected identifier
  PERS num            k := 150t;
                               ^
Compilation exited abnormally with code 1 at Thu Mar 14 16:35:09

-1:--  *compilation*   4:39PM   (Compilation:exit [1])--L19--C0--Bot----------------------------
Mark set
```

Figure 8.2: *The feeder provided an emacs compile environment which allowed the user to remotely edit and execute RAPID programs (the program in the Figure is just provided as an example, please disregard from details).*

or with the simulated tracker. The interface was simple and the application did not need to be aware of issues such as internals of data packets, error-handling procedures etc.

When the tracker component was used for simulation, a simulator emulated the tracker controller's actions. A simulated tracker, i.e. the tracker running a simulator subcomponent, could work in conjunction with a real or simulated robot. Again, this did not affect the application, which because of the tracker interface was truly unaware of whether the tracker connected to the simulator or to the tracker control unit.

Next will a description of the physical tracker, the ServoRobot M-Spot Laser and the CSR4000 control unit, follow.

8.6 Physical tracker components

8.6.1 Laser scanner

The laser beam projection is called the optical plane and the camera can only image objects that intersect the optical plane within the effective depth of field of the camera. The coordinate system of the camera is defined in the optical plane. A 3D model, as represented in the RSA, is shown in Figure 8.3. Depending on the camera head, it has a practical resolution ranging from less than 0.015 millimeter at close-working distances (100 mm) to 1.5 millimeters at far working-distances (1000 mm).

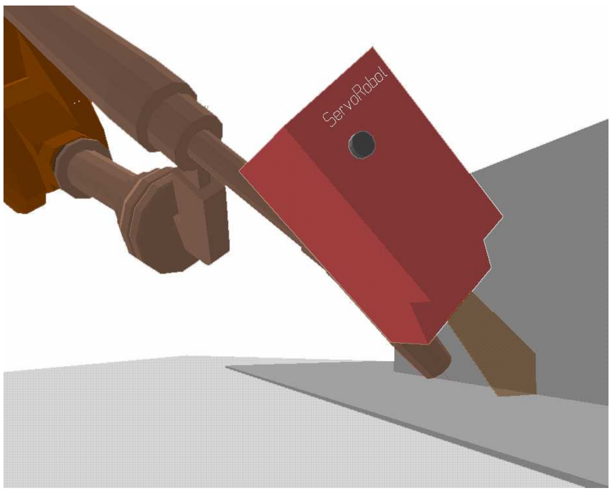


Figure 8.3: *A 3D model representation in the RSA of a welding torch with attached seam tracker.*

8.6.2 Control unit

The controller maintained the camera at its optimal operating level. It adjusted the power and/or sensitivity in order to cope with varying surface finishes and digitized the video signal of the camera and performed low-level vision-processing.

8.6.3 Processing algorithms

Several image-processing algorithms were provided dedicated to the five following joints: fillet, corner, lap, butt and v-groove, as shown in Figure 8.4. Several techniques were used to ensure the system robustness. A scratch on the surface of the plates or shiny surface conditions could cause outliers. An outlier is a point that is far away from most of the others. The outliers were detected and eliminated at the initial stage of image-processing. The algorithm of each joint included validation features taking into account the obstacles that can be seen in the field of view of the camera.

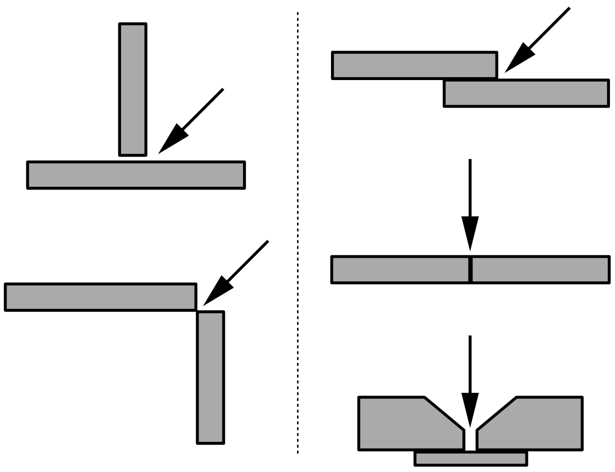


Figure 8.4: *Standard joints: fillet and corner in left column. Lap, butt and v-groove in right column.*

8.6.4 Image-processing region and breakpoints

Each profile contained 256 or 512 points. The boundary size defined the image-processing region, which could be specified for two reasons: to restrict the vision-processing region in order to avoid unnecessary features that may confuse the vision analysis, and to reduce the vision-processing time by eliminating unnecessary profile points.

Breakpoints are feature points extracted from the profiles. In each image-processing algorithm, the breakpoints were extracted from the profiles based on the joint features. They were numbered from 0 to 7 and were used to define the tracking point position or to extract further information. The quantity of breakpoints depended on the quantity of joint features but did not exceed 8. The breakpoints were labeled as B_0, ..., B_7.

8.6.5 Weld joint recognition

A basic function was to filter the data and create line segments that matched the criteria of specific weld joints and their tolerances. The details in this process are beyond the scope of this thesis but included in principle (1) elimination of outliers, (2) creation of line segments, (3) merging of line segments with similar geometrical characteristics and (4) validation of joint parameters, for instance angle and gap, see Figure 8.5. The process result consisted of a set of breakpoints, i.e. information describing the geometry of the chosen joint type. If no such joint was discovered, an error was returned.

8.6.6 Process results

The process results from both the physical and virtual tracker was used by the "master", which was the tracker sub-component closest to the application. Whenever the application wished to receive information from the simulated tracker, the request was handled by the master sub-component, which in turn talked to either the tracker control unit or the simulator, without knowledge of which. When running the tracker in a simulated mode, a tracker GUI could be used to monitor the joint and the calculated breakpoints.

8.7 Simulated laser camera

If a simulated robot was used, a simulated camera emulating the tracker, could be applied. A simulated robot could be an ordinary robot from a robot library in any RSA or other application providing visualization and an API that enabled the user to create a simulated camera for interaction with the work-cell objects. By having the simulated camera emulate laser rays sweeping over the camera's field of view and working-distance, an array of distances to objects in the work-cell could be collected. The sweep distance array could then be sent to the "slave", which was another tracker sub-component.

8.7.1 Virtual laser scanner

The virtual scanner was represented by only one function, which emitted virtual rays over a certain angle. For each ray, the RSA API `AxsEntityRayCast()` was called, which yielded the distance in millimeters and the point of intersection to the closest part (plates, weld joint). The distance was compared to a maximum

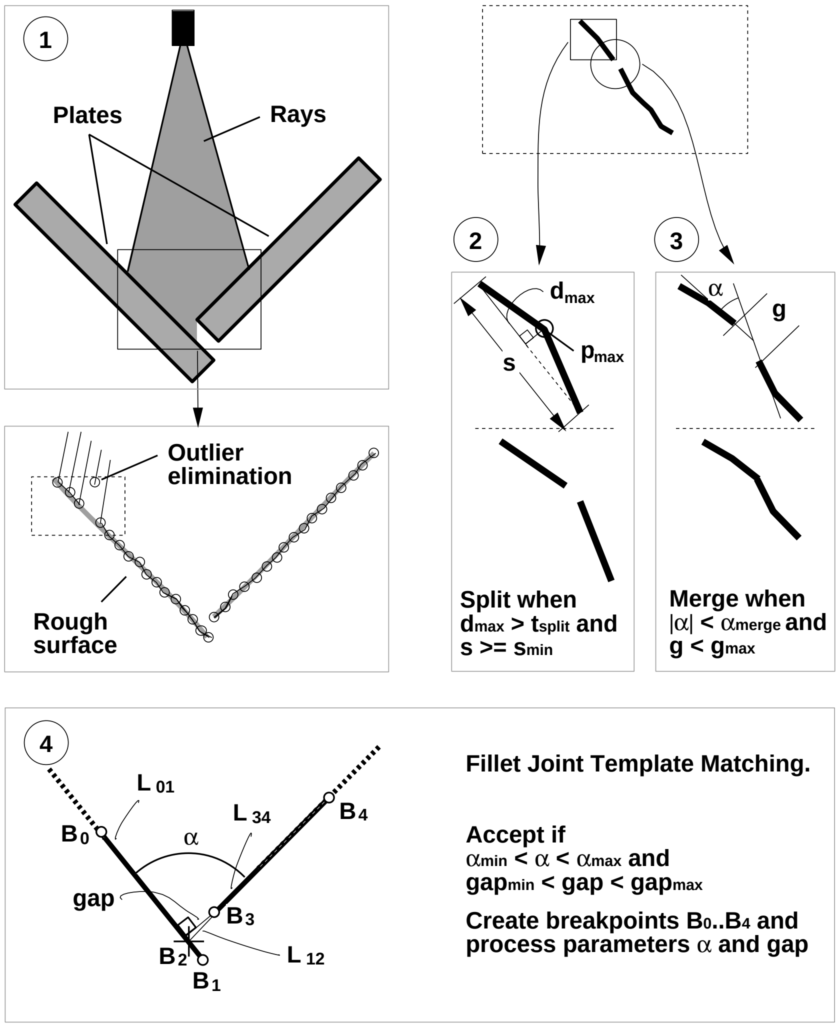

1
Plates
Rays
Outlier elimination
Rough surface
2
3
d_{max}
p_{max}
s
α
g
Split when $d_{max} > t_{split}$ and $s >= s_{min}$
Merge when $|\alpha| < \alpha_{merge}$ and $g < g_{max}$
4
L_{01}
L_{34}
B_0
B_4
α
gap
B_3
B_2
B_1
L_{12}
Fillet Joint Template Matching.
Accept if $\alpha_{min} < \alpha < \alpha_{max}$ and $gap_{min} < gap < gap_{max}$
Create breakpoints $B_0..B_4$ and process parameters α and gap

Figure 8.5: *Principle of the joint filtering process: (1) elimination of outliers, (2) creation of line segments, (3) merging of line segments with similar geometrical characteristics and (4) validation of joint parameters.*

hit distance that limited the measurable area. Measurements exceeding this distance were not taken into account. Finally, a complete sweep was returned as a *raylist* containing a vector with measured points of intersection in the coordinate system of the virtual camera and the total number of successful measurements.

8.8 Simulated control unit

The segmentation process and fillet joint template matcher represented the control unit in the physical world.

8.8.1 Segmentation process

The slave sub-component performed image-processing on the sweep distance array using similar algorithms as the physical tracker control unit. The segmentation process consisted of calling the *split* procedure followed by the *merge* procedure, see Figure 8.5. The points of intersection formed a profile of straight lines. Initially, the profile was considered consisting of only one segment. The *split* algorithm divided the initial profile into several according to t_{split}, s_{intv} and s_{min}. Only segments with length between end points larger than the split threshold, t_{split}, were split. To increase the calculation speed, the s_{intv} parameter could be set to an integer value greater than 1. A value of 2 means that every second ray were discarded in calculations. The segments were stored in *segmentlist* along with *num_segments*, the number of segments. Split was implemented as a recursive algorithm, which divided the segment until either d_{max} was less than t_{split}, or the region was smaller than the predefined minimum size s_{min}. d_{max} was the maximum distance between the intersection of a line between the end points of the segment and the normal to this line to a point included in the segment.

Next, the *merge* algorithm was called. Two segments that were close enough to each other and fulfill angular requirements, i.e., less than the maximum merge gap, g_{max} and the maximum merge angle, α_{merge} respectively were merged to one. A merge gap could occur after outliers were eliminated. The merge angle was the angle between two segments. The iteration continued through all segments in the profile.

8.8.2 Fillet joint template matching

The result from the segmentation process consisted of a number of segments that fulfilled stated terms of linearity. The simple template-matching module implemented used these segment to check if the segments corresponded to pre-defined angular and gap restrictions and to create breakpoints.

8.9 Distance-sensor

The range sensor, which measured the distance to obstacles in the weld direction, utilized a similar simulation technique as the tracker.

8.10 The RSA and its resources

The RSA provided resources for creating the nominal virtual work-cell and the simulated physical work-cell, visualization and collision detection in the simulated work-cell. In simulation mode, it also displayed the "physical" robot motion and distance measurements utilized by simulated sensors. These resources were reached by the RSA's proprietary APIs. RLib handled the kinematic relationships between these objects and the RSA was responsible for visualization, collision detection, export of the nominal kinematical relationships and for simulating the M-Spot camera.

8.10.1 Export of the nominal kinematic relationships

When the nominal work-cell was created in IGRIP, a RLib database was built by using IGRIP's proprietary *Axxess* API. During the recursive traverse of the work-cell objects, the kinematical relationships were saved outside IGRIP in RLib database format by using RLib object creation methods.

8.11 Application objects

The application consisted of a number of independent objects, where instances of some of the objects could be used and re-used in virtually any meaningful order. The objects encapsulated the logic in different process phases of the application; search phase, start point phase, weld phase, obstacle avoidance phase, end weld phase etc. For all application objects, the user defined object-specific behavior such as speed, weld current and other process-dependent data[5]. Several instances of an object could be utilized in one application, each instance with its own process characteristics.

[5]However, weld process dependent data was not utilized during the experiments.

8.11.1 Search object

The search object was utilized to determine the weld start point. When found, it created a start point path and a weld path. The search continued so as to measure the weld workpiece orientation and thus, the weld direction. This extended search distance was usually equal to the distance measured in the weld direction between the weld wire and the laser beam sweeping perpendicular to the weld direction. The search information was saved as a number of tags in the weld path, each containing a pose. A search path was a user-defined number of sweeps in the weld direction, and each sweep was performed at some distance from the nominal start point. After the extended search, the virtual workpiece was calibrated with respect to the actual pose of the (possibly simulated) workpiece measured in the search.

8.11.2 Start point object

This object consisted of the start point task that in turn executed motion along the start point path that was created by the search object. Usually, the start point path only consisted of a single start tag.

8.11.3 Weld object

The weld object continuously read data from the tracker and created tags during the welding process. In other words, new poses were appended to the weld path at the same time the as motion system used earlier added tags to guide the weld gun. Instances of this object could be utilized anywhere between the start point object and the end weld object.

8.11.4 Obstacle object

The obstacle object responsibility was to safely guide the robot in the vicinity of the obstacle. The robot followed a predefined obstacle path. The path was “owned” by the obstacle being described in the obstacle coordinates, and naturally depended on the shape of the obstacle. The obstacle object could be applied between weld two objects or between a weld object and an end weld object.

8.11.5 End weld object

The end weld object was similar to the weld object except that a lost weld message from the seam tracker was interpreted as end-of-seam instead of being interpreted as an error. It was normally used after a weld object but could be applied after an obstacle object as well. In any case, it was utilized close to the expected end weld point.

8.12 Application object interaction mechanisms

While it was comfortable to split the weld task into logically defined discrete sub-tasks using the above-mentioned objects, since robot motion is continuous, there was a need to know when one object should hand over the responsibility to another. Instances of, for example, a weld object could occur after a search object or after an obstacle object and therefore the object-initialization prerequisites also had to be defined. The strategy utilized in this implementation was to refer to common motion data by calls to objects owned by the motion control library, RLib (briefly described below). Application-specific data, such as references to the robot, tracker and distance-sensor, were handled through a shared process object that was passed to every application object during initialization.

8.13 RLib, the high-level motion control library

All components, except for the tracker GUI, were built upon objects from RLib, a lightweight library for high-level simulation and high-level control of advanced sensor-guided robots in soft real time. RLib was the run-time library that handled the interaction with the unmodified S4C controller and interfaced application objects. RLib was written in portable ANSI C in an object-oriented style and handled model building including frame dependencies, trajectory creation, singularity detection and motion. The library was based on POSIX threads and included APIs to handle threads and synchronization between objects. RLib objects were for example tags, paths, tasks and robots. RLib handled the kinematical relationships between these objects.

RLib also contained an API that was utilized to import kinematical relations from a work-cell in a RSA or any other system providing work-cell data. Provided that the application allowed its internal data to be read, an RLib database could be saved to disk and read into a run-time database during initialization of the application. The application objects could then modify the run-time database through an API. Modification of RLib objects included both adjustment of object content and changes in kinematical relationships between objects. Other features of RLib, besides motion handling, were:

- Device independence. Handled all devices that could be described with the Denavit-Hartenberg notation.
- Synchronization and thread handling routines.
- Kinematic expression of any RLib object in any other RLib object.
- Conversion functions between homogeneous coordinates and other representations.
- Event handling, notification and subscription routines.
- Communication, printing and serialization routines.
- Safety routines and debugging help.

8.14 The world model

In the experimental structure described, the world model was split between the application, RLib and the RSA. RLib was responsible for handling kinematic relationships and for generating robot motion, while IGRIP provided graphical feedback, measurements needed by simulated sensors and collision tests. The two environments were kept synchronized during run-time by the application and any change in RLib therefore immediately affected the graphical model in IGRIP. The opposite also hold; a detected collision in IGRIP propagated instantly to the application, which in turn asked RLib to halt execution.

If the application besides the virtual work-cell also simulated the real robot, the world model was extended to comprise the simulated robot and its attached sensors.

8.15 Conclusions

A system of real and simulated components was built to enable a task-oriented application to operate by executing instances of reusable objects, each with a particular sub-task responsibility. The structure let the application use real or simulated components without change of object code and it could therefore be used for both simulation and execution of the robot program. The structure even made it possible to mix simulated and real components. Focus was on these ideas and not on the implementation issues, but virtual sensors, sensor and robot interfaces, and a run-time library were nonetheless developed to support the experimental platform.

The integration of computerized and real components made it possible to create applications that included sensor-guided robots off-line and run them unmodified on-line. In this particular experimental platform, arc welding was the application in mind but the method can of course be utilized in other areas as well.

Chapter 9

Experimental work

9.1 Introduction

An arc welding application was chosen to verify the conceptual ideas in Chapters 6 and 7 of dividing a sensor-guided process into sub-tasks using the experimental structure described previously. The use of sensors implied deviation from some nominal condition and in robotized arc welding a change of path could lead to several malfunctions that had to be detected. In this particular experiment, singularity detection, collision detection and out-of-joint limits were caught. Actual welding was not performed. Figure 9.1 shows the experimental set-up.

The experiment showed a capability to handle process-related events during run-time in a system where real and simulated objects (robots, sensors, workpieces) were transparently interchangeable. The on-line events were

- finding the workpiece actual pose,
- calibration of world model objects based on sensor input,
- trajectory creation based on calibrated objects,
- singularity and out-of-joint limit detection performed in the calibrated world model,
- collision detection of simulated objects in the calibrated world model, and
- handling of objects that interrupted the trajectory.

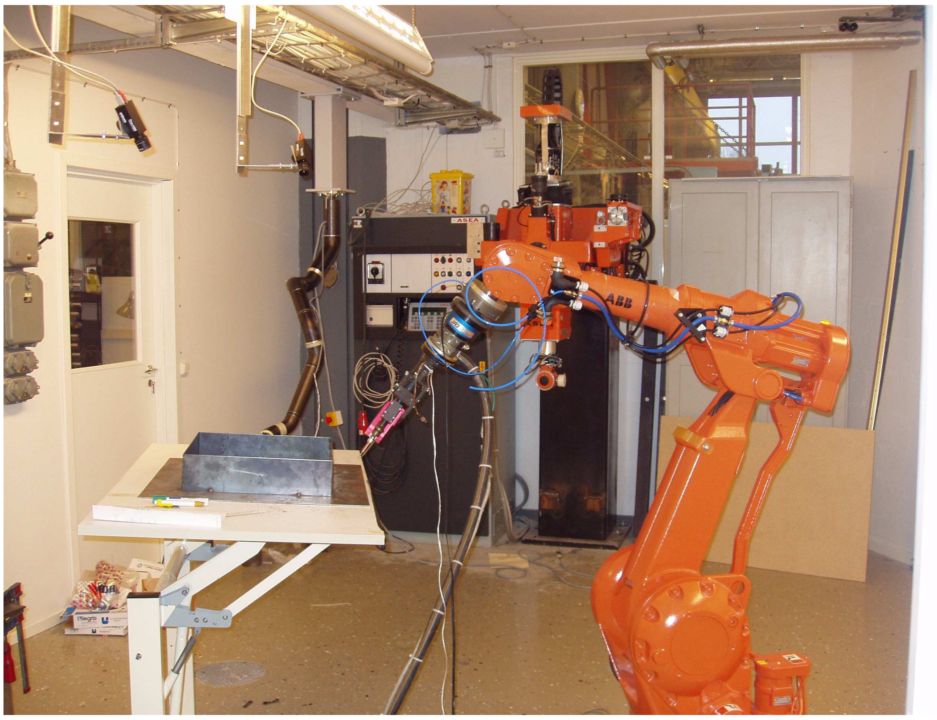


Figure 9.1: *Acutal experimental set-up. An ABB 2400/16 robot with unmodified S4C controller (outside view), the M-Spot Laser Scanner attached 50 mm ahead of the weld torch with its CSR4000 controller (outside view) and two Unibrain Fire-i-400 cameras, each equipped with a 3.5 mm lens.*

The application execution was essentially based on a number of objects that controlled robots and sensors without knowledge of if they were simulated or real. Simulating the entire welding application on the computer did most of the experimental work. The delay of two interpolation steps caused by the design of the generic RAPID program running on the real robot controller influenced the tracker and was therefore also simulated.

The program was defined by the subset of chosen objects and the order between these objects. The order of objects was laid out by hand and was compiled in a "normal" C program. The program then ran autonomously without any human intervention.

The application remained unchanged independently of whether real or simulated components were used. By using, for instance, a real robot and a simulated laser tracker attached to a simulated robot in IGRIP, the real robot followed the simulated robot's trajectory and the real robot's motion could be studied without any chance of collision with real obstacles. In other experiments, a real laser tracker was used with a virtual robot. By using this set up it was easy to make sure that the tracker worked as supposed without having to run the risk of colliding with the physical robot.

9.2 Creating and importing the nominal model

The nominal model was created in IGRIP and included the workpiece, the robot, the seam tracker and the distance-sensor. The cameras[1] were not included in the model since they were not simulated. The work-cell in IGRIP was recursively traversed. RLib APIs were used to save the work-cell to a file in the RLib format still keeping the hierarchical kinematical relationships between items in the work-cell. In the imported work-cell, only the robot's base coordinate system was represented. The internal relations between robot joints, forward and inverse kinematics, were accessed by RLib from a shared library.

9.3 Reading nominal data and initiating application objects

After the application was launched, a run-time database was built from the saved file. After the nominal work-cell was defined, the application objects were initialized. The following application objects were used to build the experimental application:

1. Start object. Contained the path from the home position to the position of the search start point.
2. Search object. Controlled the search path, the creation of the start point path and the first 50 mm of the weld path. Created a new workpiece pose and calibrated the simulated workpiece according to it.
3. First weld object. Was responsible for the weld from the start point to the reinforcement[2].
4. Reinforcement object. Handled welding just before and just after the reinforcement, and guided the robot safely around the reinforcement.

[1]Camera set-up and mathematical treatment of camera output are courtesy of Stefan Adolfsson (stefan.adolfsson@hbg.lth.se)

[2]It is a type of reinforcement that supports the structure during welding only and is sometimes removed depending on the application.

5. Second weld object. Controlled welding from the reinforcement and almost to the end of the seam.
6. End weld object. Was responsible for the last weld distance.
7. Retract object. Retracted to a safe pose after the weld was done.

The first and second weld objects were instances of the weld object but initialized separately. Besides these active objects, simulation objects were also initiated. These objects, which were used to simulate the operations before actual searching, welding, etc., served to minimize chances for entering singular conditions, out-of-joint limits and collisions.

In the main experiment, described below, stereo cameras were used to improve the nominal pose and to measure the position of the reinforcement. As a second experiment, a simulated distance-sensor was used to measure the reinforcement position during the first weld. After the distance was measured, the simulated work-cell in IGRIP was updated, and since the reinforcement path was expressed in reinforcement coordinates, the path was updated accordingly. Since the distance-sensor did not exist in the real world, this experiment was done without a real reinforcement, but the real robot acted as if a real reinforcement existed and followed the simulated trajectory to avoiding it.

Only a minor change to the code was needed to handle the case when the reinforcement's position was known from the beginning, and the case when its position was computed during the weld. This procedure, which was called during each interpolation from the weld object, either just stopped the weld object after the fixed distance given from camera readings, or called the distance-sensor to measure the range and then stopped the weld object at a specified distance from the reinforcement.

9.4 Improving the workpiece nominal pose

The workpiece, seen in Figure 9.2, was essentially a four-sided box with a size and geometry known in advance. The pose of the workpiece and exact position of the reinforcement along the weld was, however, not known. The objective of the first part of the experiment was to find a good estimate of the workpiece's actual position and orientation by using the stereo cameras. The calibrated cameras yielded a rough estimate of the parameters as well as of an accurate relative position of the reinforcement compared to the weld start point, represented by a corner of the box. The cameras yielded a good estimate of the workpiece position

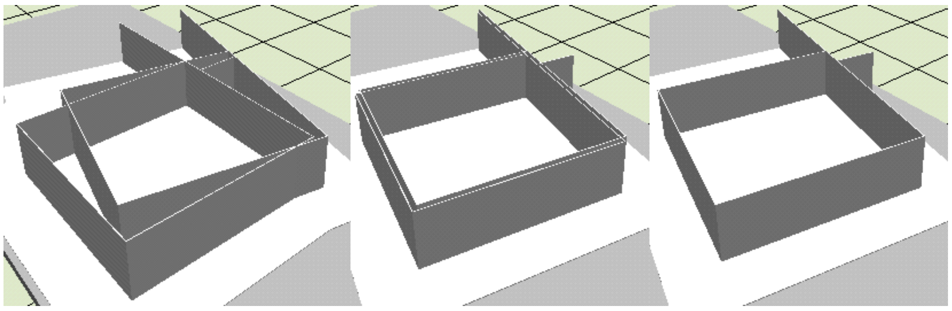


Figure 9.2: (Left) *The workpiece before calibration from the vision system.* (Middle) *After the camera calibration but before search with the laser tracker.* (Right) *After searching with the tracker, the position and orientation error of the workpiece is sufficiently small to create a good quality weld*

and orientation. In the vertical direction, the error was less than five millimeters and in the horizontal direction less than 20 mm. The reinforcement's pose was accurately given as an offset from the weld start point.

The tack-welded reinforcement had a known size and orientation perpendicular to the weld seam but had an unknown absolute position measured in the weld direction. Having knowledge of what kind of obstacle is expected and of their approximate position of where to expect them is relevant in industrial applications and should not be interpreted as a restriction.

The application adjusted the position of the virtual workpiece and reinforcement to coincide with camera output, see Figure 9.2. The simulated model in IGRIP was updated as well. Under lab conditions, the workpiece position and orientation interpretation of the camera-produced data were in the range of two centimeters from its actual pose. The path from the home position to the searched starting point was represented in workpiece coordinates and, therefore, a good estimate of the nominal position and orientation of the workpiece assured a collision-free and safe trajectory to the position where the search started.

9.5 Execution of start and search simulation objects

After the nominal pose of the workpiece had been improved by camera readings, the start and search simulation objects were executed. A simulated robot in IGRIP followed the start path to the search start point and then a complete search including all sweeps was then conducted, see Figures 9.3 and 9.4. If a singular condition occurred, or if an out-of-joint limit condition or collision was encountered, the ap-

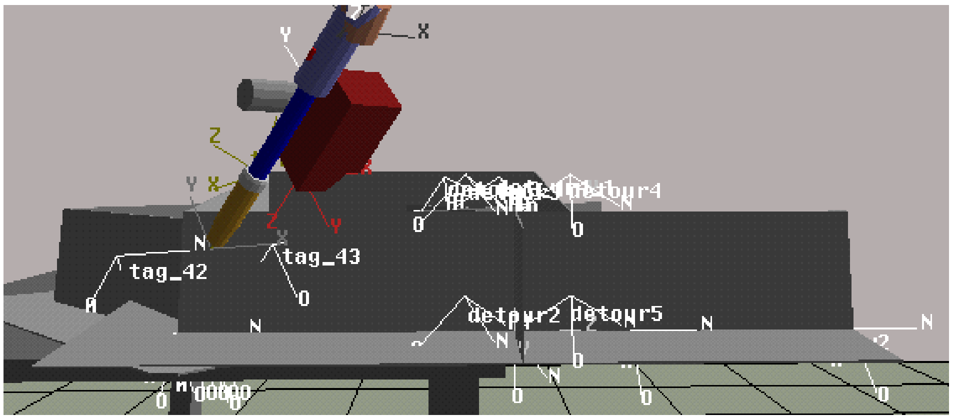


Figure 9.3: *The search phase. The semi-hidden box in the background represented the nominal placement of the workpiece. In the foreground, the actual pose of the box was simulated. The search path followed the nominal weld direction. The tags already created in the lower left corner of the Figure represented the start point and other poses found during the extended search. These tags became first poses in the weld path.*

plication was aborted signaling an error condition.

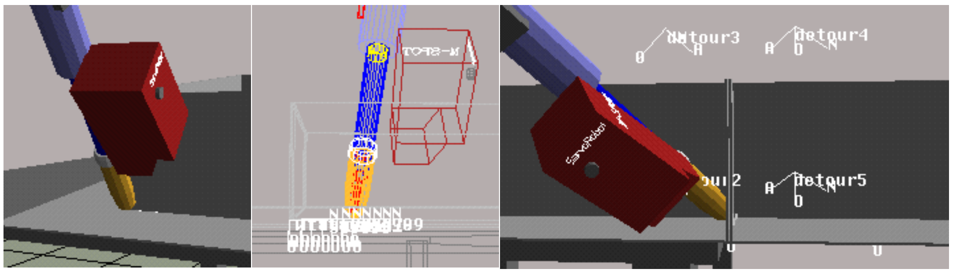


Figure 9.4: (Left) *The start phase.* (Middle) *The weld phase. The tags already created at the bottom of the middle picture represented the start point and other poses found during the extended search.* (Right) *The reinforcement phase. The reinforcement path is marked as detour.*

9.6 Start point search and trajectory creation

A search path, representing the workpiece's frame and parallel to the nominal weld direction covered the position of the nominal start point, was followed. Us-

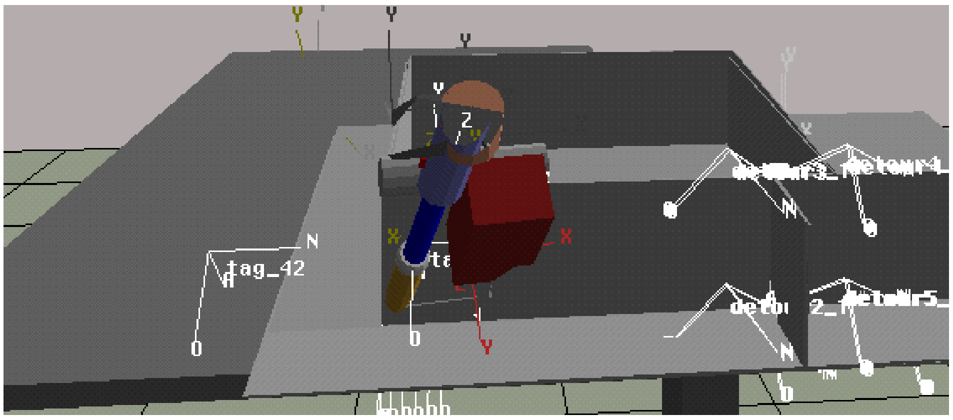


Figure 9.5: *The start point phase. After the search phase the virtual workpiece was calibrated by the application, the laser was lit, and the weld torch approached the weld start point.*

ing the cameras to adjust the nominal pose of the workpiece guaranteed a safe search path. Figure 9.3 shows a simplified search between just two tags. The search volume occupied at the most $10 \times 10 \times 5$ cm but normally less, dependent on the quality of the output from the cameras for the given workpiece and light conditions. The search could contain several sweeps from side to side.

9.7 Finding the start point and weld path generation

When the joint was eventually found, the start point was calculated based on the breakpoint data provided by the tracker. If the tracker was unable to find the joint during the search, the process was aborted and the robot stopped. The start point was then saved as the first tag in the newly created trajectory – the weld path, see Figures 9.4, 9.5 and 9.7.

9.8 Workpiece calibration

The laser camera was mounted 50 mm ahead of the tool tip and the search continued over this distance and accumulated information about the joint. This extended search phase, see Figure 9.3, yielded joint-local information and the actual orientation of the workpiece. The extended search resulted in an update of the nominal world model to the workpiece real position and orientation, see Figure 9.2. All objects referring to the workpiece – paths, tags and the reinforcement – also automatically received their correct pose.

9.9 Singularity check and collision detection

Before the robot followed the weld path, a singularity check was conducted. It was performed as a simulated weld; a virtual robot was moved along the weld path and the check was done for each interpolated pose. Since the position of the reinforcement was known, the detour around it, the second weld and the end weld were also simulated at this point. The application was halted if the robot came to close to a singular pose or if it was out-of-joint limits or collided with anything in the work-cell.

The actual pose of the workpiece could possibly induce a collision between the robot and objects in the work-cell and therefore collision detection was also performed in parallel with the singularity check during the simulated weld. Optimized collision avoidance and trajectory re-planning, which are specific research areas, were not particularly studied in this thesis but the experiment showed the benefits of using a world model to detect and avoid collisions.

9.10 Welding from the start point towards the reinforcement

After a successful singularity and collision check, the application was ready to perform the first weld, from the start point to the reinforcement. The simulated tracker sent measured data to the tracker slave, which calculated breakpoints describing the joint profile. The application used the breakpoints to calculate tags, which it appended to the weld path. These tags were then used by RLib to calculate joint values that created the trajectory for the (simulated) robot, see Figures 9.6 and 9.7. The laser tracker had already scanned the first 50 mm path length and tags taking local deviations into account were created during the extended search. When the weld started, new tags were added to the weld path at the same time as the weld torch followed previously created tags.

9.11 Handling the reinforcement

The precise location of the reinforcement was defined after the initial calibration. The vision system yielded the information needed, i.e. the distance from the weld start point to the reinforcement. The reinforcement was designed to let the weld

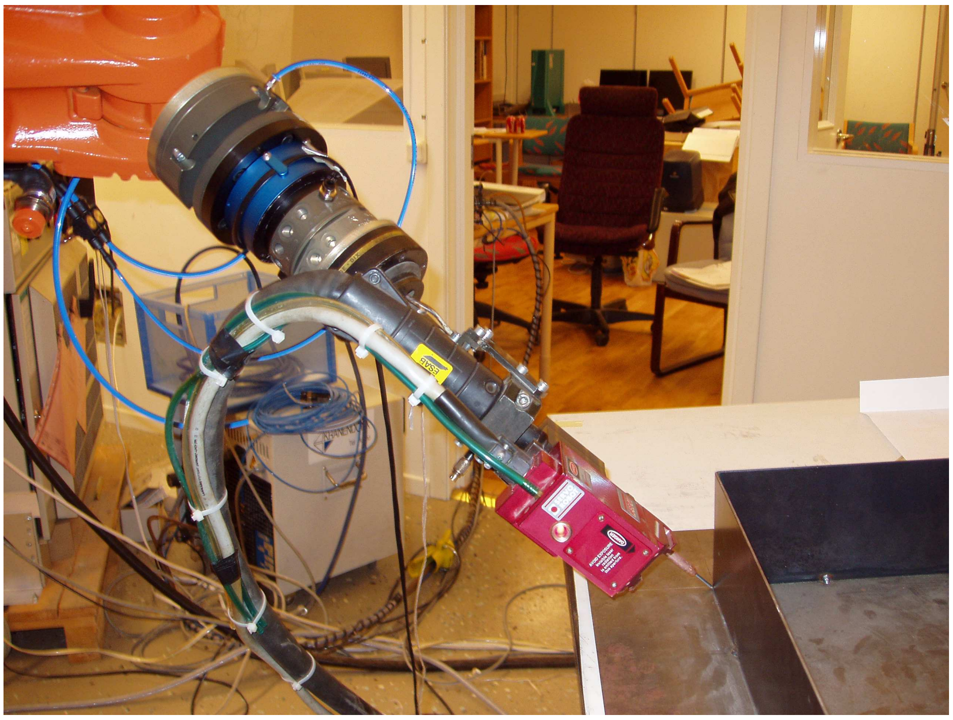


Figure 9.6: *After executing the start point object, the application performs the first weld.*

continue uninterrupted below it. To be able to approach the reinforcement and to create a continuous weld, the M-Spot camera was rotated 120 degrees, away from the workpiece and the weld torch was tilted by 45 degree angle from the weld direction. During the time when the weld torch was rotated and tilted, the seam tracker was temporarily put in a passive mode and welding performed without sensor readings.

The path around the obstacle was expressed in virtual obstacle coordinates, which in turn were expressed in the virtual workpiece coordinate system. Since the virtual workpiece at this point was calibrated, the obstacle path created a valid trajectory around the reinforcement after it was calibrated. Moreover, by using a path expressed in obstacle coordinates, it was easy to exchange obstacle without affecting application code, see Figures 9.4 and 9.8. When the weld on the left side of the box was complete, the weld torch was retracted in the opposite torch direction and followed a trajectory relative to the reinforcement.

By using a world model, it was easy and desirable to create trajectories relative to

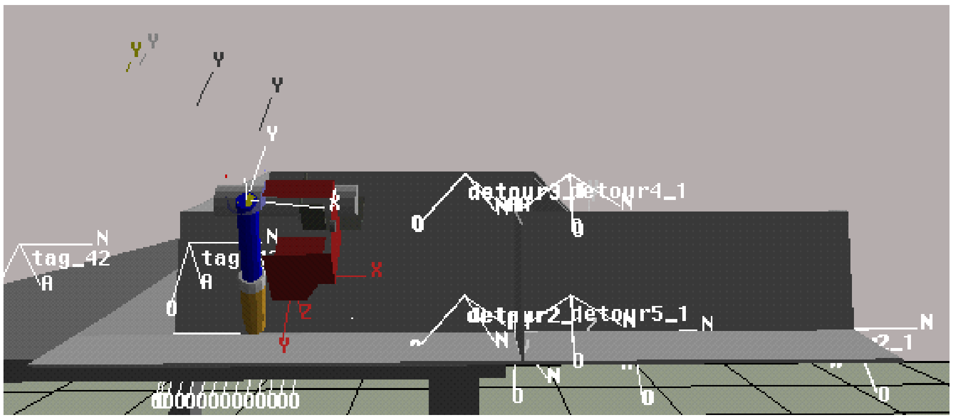


Figure 9.7: *The weld phase. The simulated tracker sent measured distances to the tracker slave, which calculated breakpoints describing the joint profile. The application used the breakpoints to calculate tags, which it appended to the weld path. These tags were then used by RLib to calculate joint values that created the trajectory for the (simulated) robot.*

other objects. This produced an association between the sub-task program, the trajectory and the real object, and encapsulated their dependencies. The association could then be reused to handle similar sub-tasks in different contexts.

9.12 Welding from the reinforcement

The second weld started when the weld torch and laser tracker reached the normal perpendicular orientation to the seam. The second weld object started welding without local sensor readings over the first 50 mm and continued welding until 90% of the weld was made. At this point, the second weld object stopped and the end weld object started and continued until the end point was reached and the retract object moved the torch from the workpiece, see Figures 9.9 and 9.10.

9.13 Conclusions

The experimental system running an arc welding application shows how a system that includes sensor-controlled robots can be built to allow simulation and execution without any change of the task oriented program. Independent and reusable components, here called objects, autonomously detected singularities, collisions

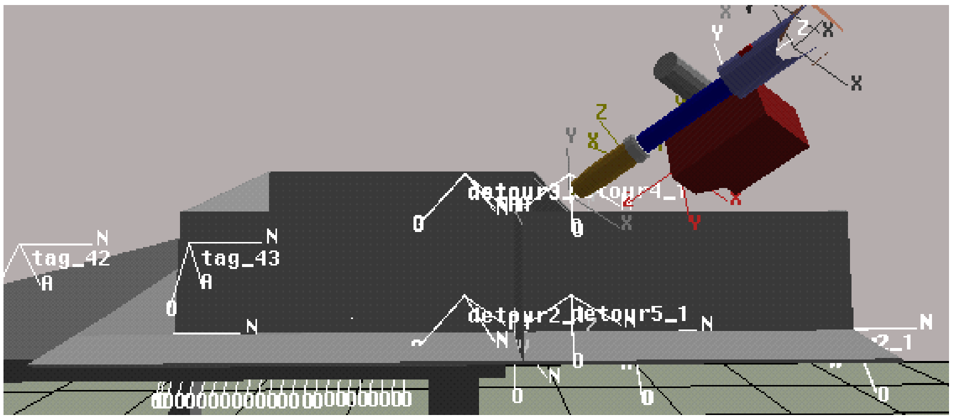


Figure 9.8: *The reinforcement phase. The reinforcement path is marked as detour.*

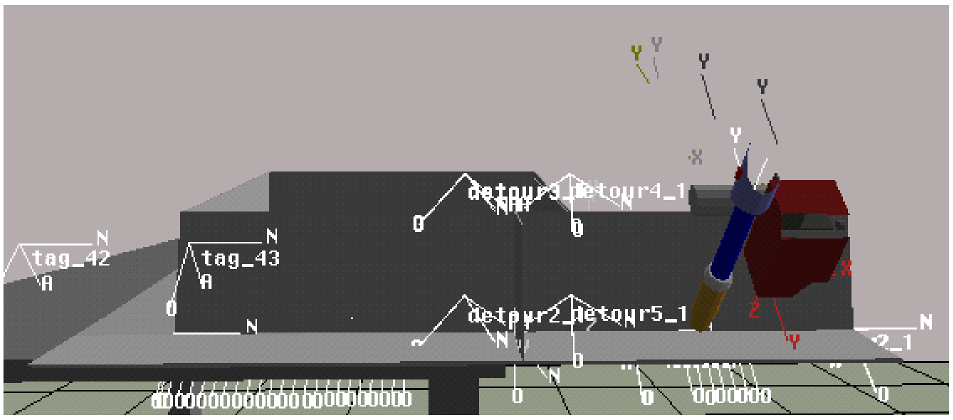


Figure 9.9: *The second weld phase. A second instance of the weld object is used.*

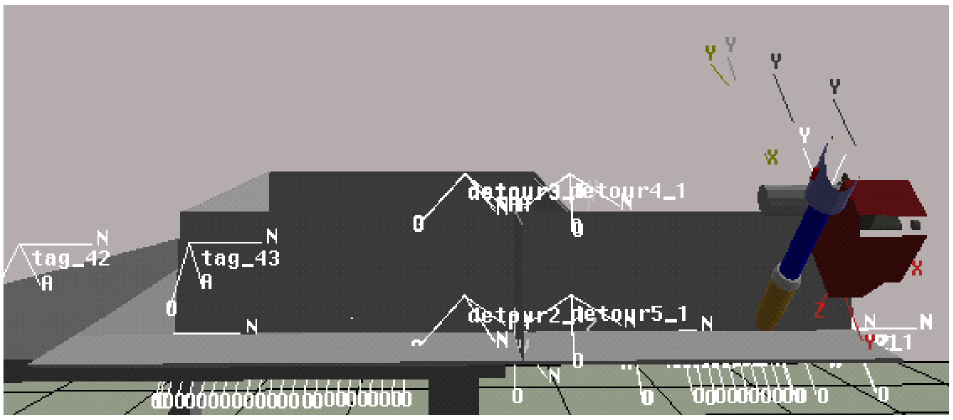


Figure 9.10: *The end weld phase. To highlight the idea of a unique process information per object, the end weld object used has been given an increased speed compared to the weld object that shows in the Figure as greater distances between tags.*

and out-of-joint limits. These are common problems even without sensors, but can be resolved before the application is run on the shop floor. However, when sensor-guided robots are used, these limitations cannot be resolved beforehand. Some of the advantages with the experimental platform are summarized below:

- The overall performance, robustness and algorithms for information feedback could be investigated and analyzed at an early stage.
- The implementation of the sensor to the real, physical robot was a seamless procedure and no major changes had to be made except for issues related to calibration. Accurate sensor calibration is, however, critical to the system.
- The sensor interface developed has the advantage of being a platform for both simulation and actual operation. The same code was used in both modes.
- The "requester"[3] of sensor data could be exchanged without affecting the sensor and vice versa by separating the requester from the sensor itself.
- During execution, the requester using sensor data was unaware of which sensor, virtual or real, it was receiving sensor data from since both used identical communication protocols and channels.

[3]The requester is the client which is provided with sensor data. In this case, the requester resides in the application.

- During execution, dynamic conditions were preserved independently of if a virtual or real work-cell was used. Deficiencies such as delays in the real system were also simulated.
- During development, time was saved because modularity allowed specific tests on component level during testing and debugging.

By using a mix of virtual and real sensors and robots, it is possible to gradually test different parts of the application. This gives a robust way of creating sensor-based applications. Most of the development of the experimental platform has been done by simulating the system and only a limited amount of time has been spent in the lab with the actual robot.

The problem of sensor calibration is always important and became an issue during the experiments. Small positional and rotational errors in the wrist-sensor transformation matrix yielded large (10mm) positional errors, when sensor readings where done from different distances to the workpiece.

The high-level control system shows a parameterization technique based on subtasks where instances of a limited set of objects can be reused in similar contexts. This generalization is one of the most important outcomes of the author's work and indicates applicability of the worked out techniques in a large number of industrial activities. In the experiment, the weld object was used both between the start point object and the reinforcement object as well as between the reinforcement object and the end weld object.

The utilization of a world model makes it possible to use and accumulate work-cell information and to act on events and information which normally would not be available to the robot program otherwise. This is of course even more important when a system of robots and sensors is controlled.

Chapter 10

Discussion

10.1 Introduction

The increasing emphasis on more personalized products and shorter product life-cycles as well as reduced production cost will result in major changes in manufacturing practices. Competition requires higher quality of manufactured products. Different strategies will be needed to cope with these changes and the thesis has included study of the consequences of resolving the sensor issues among several other matters. The use of sensors has turned out to be particularly important and the discussion therefore focuses on this issue.

Sensor feedback must include several levels in time space and information complexity and a comprehensive control perspective to meet aims set up in a task or process specification. Traditionally, sensors are used to feed back information to a low-level type of actuator or process control. Advanced application processes and a higher level of autonomy implies more complex relationships; observable variables are not necessarily those that are controllable and the controllable variables are not necessarily those that define the task. Hence, in complex industrial operations there are mapping issues in both directions between not only observable variables that are detected by sensors and controllable variables, but also between the task specification described in terms of how to reach productivity and quality measures and how to control the process to obtain such goals. In most cases this is not a trivial problem, as many controllable variables are contradictory. As a result, many such issues must be considered as optimization problems that, due to the complexity and incoming information from sensors in real-time, must be solved on a case-by-case basis.

The research responds to the demands for flexibility in the use of sensors in coping with various facets of production. Examples of increased flexibility include reduced requirements on feeders (vision), seam tracking during welding (triangulation scanners), inspection (various type of sensors) and so on. However, applying sensors in industrial automation is not as easy as it may appear. Sensors not only add valuable information needed to perform a task, but also cause system complexity. Thus, the robustness of the system may be degraded as a result of poorly integrated subsystems.

The research responds to the demands for quality, safety and a reliable view of the advancing process during the process of development. The robot programs developed for real-time decisions require a completely different simulation environment. The traditional way of off-line programming and downloading of ready-to-go programs will not work as greater autonomous behavior with sensor feedback is needed. Today's methods are only sufficient for large-scale manufacturing such as in the automotive industry. In one-off and small batch production, sensors give the robot system more flexibility and speed up product changeover.

10.2 Sensor modeling, simulation and integration

The sensor model mimics closely the behavior of the real sensor by using similar characteristics and aspects related to sensor and robot control. This is different from previous research in sensor modeling, where generic (non existent) type of sensors are modeled. The use of simulated sensors and the methodology developed to model sensor-guided robot systems provide a good match between reality and simulation. Such modeling allows the manufacturer to deal with the shorter product life cycles required by current and future customer demands. It permits sensor-based systems to be analyzed at an early stage. A further step would be to introduce sensor models from sensor vendors in a similar way as robot vendors now offer software for RSAs for more realistic robot simulation (RRS).

The developed sensor model is fully integrates with a simulation and execution model by the use of generic interfaces. The interfaces are generic to such an extent that they handle simulation and operation without any change of the application. The sensor developed is modeled after an existing sensor, and is utilizing the same protocols as the real counterpart. The generic sensor interface developed lets the sensor act as a separate component in a simulation or run-time system.

In (Chen and Trivedi, 1994), sensor interfaces are utilized that allows the user to decide whether the sensor and robot should act in simulated or real mode. In the

Chen and Trivedi system, one of the three modes: *Operator Interface/Monitor Mode, Real Controller/Virtual Robot Mode*, and *Off-Line Visualization Mode* as discussed earlier, had to be selected. However, modes could not be selected on individual sensor (and robot) basis, and therefore the system lacked the potential of mixing real and virtual components, one of the significant strengths of the system described in this thesis.

Most virtual sensors developed, for instance (Li et al., 1998; Brunner et al., 1999; Fridenfalk, 2003) do not have a real counterpart. They are developed to solve a specific problem, for instance the force-sensor described in (Li et al., 1998) or have general objectives as the 6-D seam tracker reported in (Fridenfalk, 2003). They may also be of a generic[1] type such as the camera, seam tracker and force sensor utilized in the impressive DLR system (Brunner et al., 1993; Brunner et al., 1999; Landzettel et al., 2000; Landzettel et al., 2001). In the DLR system, most of the controller code from the simulation environment is also used in the real system, but generic sensor interfaces are not utilized and virtual and real components cannot be used concurrently.

The developed sensor has been verified in (Cederberg et al., 2002*b*) and the ideas about generic sensor interfaces are described in (Cederberg et al., 1999).

10.3 Simulation and execution of sensor-guided robots

The task-oriented framework developed has the advantage of being a platform for both simulation and actual operation. It operates on the system level and cooperates with modern RSAs but can easily be connected to any program with a graphical model. The separation between graphics and motion generation makes it possible to interface an open control system such as the one described in (Johansson et al., 2004; Blomdell et al., 2004).

By using the author's approach, task-oriented control strategies can be validated through controlled experiments in a simulated environment. As the same components and protocols are used for both environments, dynamic effects originating from the internal system are taken into account. As mentioned before, this is not fully the case with the DLR system, which lacks of generic interfaces to components.

The methodology described makes it possible to produce and validate sensor-guided robot programs. Robustness is increased in the sense that the robot operation has been verified in a simulated environment. With defined tolerances in

[1]Generic in the sense that no specific physical sensor has been modeled.

the world model, a nominal robot program can be produced that will most likely succeed in a real time operation.

Robotized manufacturing of unique products and small batch manufacturing need programming models where the process rather than the shape of the product is important. The parameterizing technique that is used today focuses on product families with similar appearance and does not support sensors. The presented methodology is focused on parameterizing on sub-tasks which could appear on any product independent on shape. The sub-tasks, which include sensors, which either can be used in simulated or real mode and act independently of other sub-tasks, have not been found in literature. The mix of real and virtual components that can be run with the example system developed assure a transparent transfer between simulation and execution. This functionality has not been seen in other developed systems such as those described in (Chen and Trivedi, 1994) and (Brunner et al., 1993; Brunner et al., 1999; Landzettel et al., 2000; Landzettel et al., 2001). The methodology is submitted in (Cederberg et al., 2004).

By using objects that operate on a continuously updated world model, simulation and process execution can be run on a single (nominal) model despite changes to the actual work cell. Virtual robots and sensors may simulate changed conditions in real-time and in advance of real robots and sensors, and may be able to predict and possibly avoid difficulties.

10.4 Future Research

Today's robot systems do not provide the flexibility needed to create industrial solutions that include sensors. Future enhancements with respect to the developments described in this thesis will consist of extending the capabilities of the model by implementing it on a real-time operating system and incorporate an open control system.

The model implementation should be simplified as well. This thesis has been focused on the model structure and the development of a graphical user interface for initialization and ordering of objects that should enhance implementation. Also, there is a need for replacement of IGRIP, the RSA used in the implementation. A customized implementation that includes collision detection and tightly integrates with other system scomponents is necessary if the system should run efficiently in real-time.

The simulated sensor developed could be improved by adding the capability to handle different joint-types besides fillet joint, for instance corner, lap butt and vgrove in a similar way as the real sensor operates. This would create a foundation for simulation of more complicated welding scenarios than the test case described.

Future work should also generalize the sensor interface's operation to support different sensors and different requesters of sensor information.

Chapter 11

Conclusions

It can be concluded that performing high-level control with a world model updated in real-time from sensors in a work-cell, real or virtual, using a sensor interface yields several advantages:

1. High-level control can be moved outside the actual work-cell. More effective coordination in a particular work-cell and between different work-cells is therefore possible.
2. Robot programs can be tested in a virtual world and later in a real work-cell without rewriting of code. This cuts development time and increases robustness.
3. On-line tests, collision tests, out-of-joint limit tests and alike can be performed in advance or in real-time as soon as sufficient information regarding the real world becomes available to the virtual world model.
4. Since the virtual model is updated continuously, knowledge of the process can be accumulated. This can effectively be used to recover from errors during autonomous robot operations.

Performing sensor-guided operations using an on-line geometry model outside the robot control hardware may appear more futuristic than it really is. Our experimental setup used an ABB S4 system robot controller with its RAPID programming environment where features of RAPID were available through remote procedure calls (RPC). Thus, it is already technically possible to remotely control a standard robotic system and it is not unrealistic to assume that sensor-guided robots will be controlled from world models updated in real-time in the future.

The method described to model and implement sensor functionality opens new possibilities for simulation and programming of robot systems in realistic industrial applications. This is important for more advanced manufacturing systems and specifically for rapid and virtual development of products where time is important for developing systems that produce the product. Thus, the combined simulation and run-time environment must be able to represent real world processes as they appear in the context of industrial automation. The virtual sensor developed acts in the tested cases similar to its real counterpart and has been shown to be easily managed in a simulation environment.

Bibliography

ABB Ethernet Services 3.0 (n.d.).

ABB Rapid Reference Version 3.2, RAPID Summary (n.d.).

ABB RAP Protocol Specification 1.05 (n.d.).

ABB RAP Service Specification 1.05 (n.d.).

Adolfsson, J., Ng, A., Olofsgård, P., Moore, P., Pu, J. and Wong, C.-B. (2002), 'Design and simulation of component-based manufacturing machine systems', *Mechatronics* **12**, 1239–1258.

AIX Version 4.3 Communications Programming Concepts (1997).

Andersson, C. (2003), Register Allocation by Optimal Graph Coloring, *in* 'Compiler Construction: 12th International Conference, CC 2003, held as part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003. Proceedings', Springer-Verlag Heidelberg, Warsaw, Poland, pp. 33 – 45.

AWS, ed. (1976), *American Welding Society Welding Handbook, Section 1: Fundamentals of welding*, Palgrave Macmillan, New York, NY.

Barraquand, J., Langlois, B. and Latombe, J.-C. (1992), Numerical potential field techniques for robot path planning, *in* 'Proceedings of IEEE Transaction on Systems, Man and Cybernetics', Vol. 22, pp. 224–241.

Barraquand, J. and Latombe, J.-C. (1991), 'Robot motion planning: A distributed representation approach', *International Journal of Robot Research* **10**(6), 628–649.

Blomdell, A., Bolmsjö, G., Brogårdh, T., Cederberg, P., Isaksson, M., Johansson, R., Haage, M., Nilsson, K., Olsson, M., Olsson, T., Robertsson, A. and

Wang, J. J. (2004), 'Extending an industrial robot controller with a fast open sensor interface – implementation and applications'. To appear in IEEE Robotics and Automation Magazine., 2004.

Blume, C. and Jakob, W. (1986), *Programmiersprachen für Industrieroboter*, Berlin ; New York : Springer Verlag. ISBN 0387163190.

Boddy, M. and Dean, T. L. (1989), Solving Time-Dependent Planning Problems, *in* 'Proceedings of 11th Int. Joint Conf. on Artificial Intelligence', pp. 979–984.

Bohlin, R. and Kavraki, L. (2000), Path planning using lazy PRM, *in* 'Proceedings of the International Conference on Robotics and Automation', Vol. 1, pp. 521–528.
URL: *citeseer.nj.nec.com/bohlin00path.html*

Bolmsjö, G., Olsson, M. and Brink, K. (1999), *Increased autonomy in industrial robotic systems: A framework*, Kluwer Academic Publishers, Hingham, MA, USA, chapter 2. ISBN 0-7923-5580-6.

Bolmsjö, G., Olsson, M. and Cederberg, P. (2002), 'Robotic Arc Welding - Trends and Developments for Higher Autonomy', *Industrial Robot* **29**(2), 98–104.

Boving, K., ed. (1989), *NDE Handbook: Non-destructive Examination Methods for Condition Monitoring*, Woodhead-publishing.
URL: *http://www.woodhead-publishing.com*

Brady, J. (1989), 'Special issue on sensor data fusion', *International Journal of Robotics Research* **7**(6), 1–161.

Brink, K., Olsson, M., and Bolmsjö, G. (1995), Event Based Robot Control, Focusing on Sensors, *in* 'Proceedings of the International Symposium on Measurement and Control in Robotics', Bratislava, Slovakia, pp. 507–512.

Brooks, R. R. and Iyengar, S. S. (1997). OE Reports 164.
URL: *http://www.spie.org/web/oer/august/aug97/sensor.html*

Brunner, B. et al. (1993), Multisensory shared autonomy and tele-sensor-programming – key issues in the space robot technology experiment ROTEX, *in* 'Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems', IEEE/RSJ, pp. 2123–2139.

Brunner, B. et al. (1995), Tele Sensor Programming - A task-directed programming approach for sensor-based space robots, *in* 'International Conference on Advanced Robotics)', ICAR.

Brunner, B. et al. (1999), A Universial Task-Level Ground Control and Programming System for Space Robot Applications, *in* ‘5th International Symposium in Artificial Intelligence, Robotics and Autonmation in Space’, SAIRAS.

Bruyninckx, H., Soetens, P., Issaris, P. and Leuven, K. U. (2002), ‘The OROCOS Project’.
URL: *http://www.orocos.org/*

Burdea, G. C. (1999), ‘Invited Review: The Synergy Between Virtual Reality and Robotics’, *IEEE Transactions on Robotics and Automation* **15**(3), 400–410.

Canny, J. F. (1989), On Computability of Fine Motion Plans, *in* ‘Proceedings of IEEE Int. Conf. on Robotics and Automation’, Scottsdale, AZ, USA, pp. 177–182.

Cary, H. B. (1997), *Modern Welding Technology (4th Edition)*, Prentice Hall; 4 edition (June 30, 1997), Stanford, CA, USA. ISBN 0132418037.

Cederberg, P., Olsson, M. and Bolmsjö, G. (1999), A Generic Sensor Interface in Robot Simulation and Control, *in* ‘Proceedings of Scandinavian Symposium on Robotics 99’, Oulu, Finland, pp. 221–230.

Cederberg, P., Olsson, M. and Bolmsjö, G. (2001), ‘Virtual Triangulation Sensor Simulation Integrated in a CAR Environment’, Digitally recorded by the Division of Robotics, 2001-12-18.

Cederberg, P., Olsson, M. and Bolmsjö, G. (2002*a*), Remote control of a standard ABB robot system in real time using the Robot Application Protocol (RAP), *in* ‘Proceedings of the International Symposium on Robotics, ISR2002’, IFR, Stockholm. paper No. 113.

Cederberg, P., Olsson, M. and Bolmsjö, G. (2002*b*), ‘Virtual triangulation sensor development, behavior simulation and CAR integration applied to robotic arc-welding’, *Journal of Intelligent and Robotic Systems* **35**(4), 365–379.

Cederberg, P., Olsson, M. and Bolmsjö, G. (2004), ‘A semiautomatic task oriented programming system for sensor-controlled robotised small batch and one-off manufacturing’. Submitted to Robotica at the time of printing of this thesis.

Chen, C. and Trivedi, M. (1994), ‘Simulation and animation of sensor-driven robots’, *IEEE Transactions on Robotics and Automation* **10**(5), 684–704.

Chen, J. and McCarragher, B. J. (1998), Robot Programming by Demonstration-Selecting Optimal Event Paths, *in* ‘Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA ’98)’, ICRA, pp. 518–523.

Chen, J. and McCarragher, B. J. (2000), Programming by Demonstration - Constructing Task Level Plans in a Hybrid Dynamic Framework, *in* ‘Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA ’00)’, ICRA, pp. 1402–1407.

Chen, J. and Zelinsky, A. (2001*a*), Generating a Configuration Space Representation for Assembly Tasks from Demonstration, *in* ‘Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA ’01’, ICRA, pp. 1530–1536.

Chen, J. and Zelinsky, A. (2001*b*), Programming by Demonstration: Removing Suboptimal Actions in a Partially Known Configuration Space, *in* ‘Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA ’01)’, ICRA, pp. 4096–4103.

Chester, R. (2004), ‘Introduction to arc welding’.
URL: *http://www.aussieweld.com.au/arcwelding/*

Chou, J. C. K. (1992), ‘Quaternion kinematic and dynamic differential equations’, *IEEE Trans. of Robotics and Automation* **1**(8), 53–64.

Chou, J. C. K. and Kamel, M. (1988), Quaternions Approach to Solve the Kinematic Equation of Rotation AaAx = AxAb of a Sensor Mounted Robotic Manipulator, *in* ‘Proceedings of the IEEE International Conference on Robotics and Automation’, Philadelphia, PA, USA, pp. 656–662.

Corke, P. (1996), ‘A Robotics Toolbox for MATLAB’, *IEEE Robotics and Automation Magazine* **3**(1), 24–32.

Crowe, D. (2001), ‘Designing for successful robotic arc welding automation’.
URL: *http://www.thefabricator.com*

DaCosta, F., Hwang, V., Khosla, P. and Lumina, R. (1992), An integrated prototyping environment for programmable automation, *in* ‘SPIE/OE92 International Symposium on Intelligent Robot in Space’, SPIE.

Dai, W. and Kampker, M. (2000), User oriented integration of sensor operations in a offline programming system for welding robots, *in* ‘Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA ’00)’, Vol. 2, IEEE, pp. 1563–1567.

DeepBlue (2004).
URL: *http://www.research.ibm.com/deepblue/*

Denavit, J. and Hartenberg, R. (1955), 'A kinematic notation for lower pair mechanisms based on matrices', *ASME Journal of Applied Mechanics* pp. 215–221.

Dyson, J. (2004), 'Welding Certification, A Basic Guide'.
URL: *http://www.gowelding.com/*

Eckart, F. and Francoeur, M. (2002), 'Welding Techniques'.
URL: *http://www.sensorsmag.com*

Ernst, H. A. (1961), A Computer-Controlled Mechanical Hand, PhD thesis, Massachusetts Institute of Technology, Camebridge, MA. Sc.D. thesis.

Fahim, A. and Choi, K. (1998), 'The UNISET approach for the Programming of Flexible Manufacturing Cells', *Robotics and Computer-Integrated Manufacturing* **14**(1), 69–78.

Farson, D. and Duhamel, R. F. (2001), 'Taking advantage of laser welding'.
URL: *http://www.thefabricator.com*

Feldman, J. (1971), The Stanford Hand-Eye Project, *in* 'First International Conference on Artificial Intelligence', London, England, pp. 350–358.

Finkel, R., Taylor, R., Bolles, R., Paul, R. and Feldman, J. (1974), 'A Programming System for Automation'. Stanford AI Memo 177, Stanford University, Stanford, CA 94305.

Flaig, T., Grefen, K. and Neuber, D. (1996), Interactive graphical planning and design of spacious logistic environments, *in* 'Proceedings of the Conference FIVE Working Group', Scuola Superiore S. Anna, Italy, pp. 10–17.

Freund, E., Ludemann-Ravit, B., Stern, O. and Koch, T. (2001), Creating the architecture of a translator framework for robot programming languages, *in* 'Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA '01', Vol. 1, ICRA, pp. 187–192.

Fridenfalk, M. (2003), Development of intelligent robot systems based on sensor control, PhD thesis, Lund University, Lund, Sweden. ISBN 91-628-5550-6.

Friedrich, H., Holle, J. and Dillmann, R. (1998), Interactive generation of flexible robot programs, *in* 'Proceedings of the IEEE/RSJ Intl. Conf. on Robotics and Automation', Vol. 1, IEEE, pp. 538–543.

Fuller, J. L. (2004), ‘Introduction to Robotics Programming’.
URL: *http://www.tvcc.cc/staff/fuller/cs281/cs281.htm*

Gopel, W., Hesse, J. and Zemel, J. N. (1997), *Sensors - A Comprehensive Survey*, Wiley. ISBN: 3-527-26538-4, Hardcover.

Gourdeau, R. (1997), ‘Object Oriented Programming for Robotic Manipulators Simulation’, *IEEE Robotics and Automation Magazine* **4**(3), 21–29.

Hall, D. and Linas, J. (2001), Handbook of Multisensor Data Fusion, *in* ‘CRC Press’.

Halperin, D., Kavraki, L. E. and Latombe, J.-C. (1999), Robot Algorithms, *in* M. Attalah, ed., ‘Algorithms and Theory of Computation Handbook’, CRC Press, Boca Raton, NY, chapter 21.

Helms, E., Schraft, R. D. and Hagele, M. (2002), rob@work: Robot Assistant in Industrial Environments, *in* ‘In Proc. of the 11th IEEE Int. Workshop on Robot and Human interactive Communication, ROMAN2002’, pp. 399–404.

Hicks, J. (2000), *Welded Joint Design (3rd Edition)*, Industrial Press, New York, NY, USA. ISBN 0132418037.

Hirzinger, G., Albu-Schäffer, A., Hähnle, M., Schäfer, I. and Sporer, N. (2001), On a new generation of torque controlled light-weight robots, *in* ‘Proceedings of the IEEE Int. Conference on Robotics and Automation’, Seoul, Korea, pp. 1087–1093.

Hirzinger, G., Brunner, B., Koeppe, R. and Vogel, J. (1997), Advanced Telerobotics, *in* ‘Advanced Research Workshop “Autonomous Robotic Systems”, ARS 97’, Universidade de Coimbra, Portugal.

Hissam, S. A. and Klein, M. (2004), ‘A Model Problem for an Open Robotics Controller’. Carnegie Mellon Software Engineering Institute, Technical Note CMU/SEI-2004-TN-030.
URL: *http://www.sei.cmu.edu/publications/*

Hoppe, H., Kuebler, C., Raczkowsky, J., Woern, H. and Hassfeld, S. (2002), A Clinical Prototype System for Projector-Based Augmented Reality: Calibration and Projection Methods, *in* ‘Proceedings of Computer Assisted Radiology and Surgery (CARS)’, Paris, France, p. 1080.

Huissoon, J. P. (2002), ‘Robotic laser welding: Seam sensor and laser focal frame registration’, *Robotica* **20**(3), 261–268.

Hwang, Y. K. and Ahuja, N. (1992), ‘Gross motion planning - a survey’, *ACM Comput. Surv.* **24**(3), 219–291.

IRIX Network Programming Guide (n.d.).

Jacobsen, N. J. (2004), ‘Can CIM using robots fulfill its promise for flexible production? Experience and visions for the first decade in “one of a kind steel production”’. Odense Steel Shipyard Ltd., Denmark.

Johansson, R., Robertsson, A., Nilsson, K., Brogardh, T., Cederberg, P., Olsson, M., Olsson, T. and Bolmsjö, G. (2004), ‘Sensor Integration in Task-Level Programming and Industrial Robotic Task Execution Control’, *Industrial Robot* **31**(3), 95–102.

Korein, J. U. and Ish-Shalom, J. (1987), ‘Robotics’, *IBM Systems Journal* **26**(1), 55–95.

Landzettel, K., Brunner, B., Hirzinger, G., Lampariello, R., Schreiber, G. and Steinmetz, B. (2000), A Unified Ground Control and Programming Methodology for Space Robotics Applications – Demonstrations on ETS-vii, *in* ‘31st International Symposium on Robotics’, Montreal, Canada.

Landzettel, K., Brunner, B., Schreiber, G., Steinmetz, B. and Dupuis, E. (2001), MSS Ground Control Demo with MARCO, *in* ‘i-SAIRAS 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space’, Montreal, Canada.

Lapham, J. (1999), ‘Robotscript: the introduction of a universal robot programming language’, *Industrial Robot* **26**(1), 17–25.

Legnani, G., Casolo, F., Righettini, P. and Zappa, B. (1996), ‘A Homogeneous Matrix Approach to 3D Kinematics and Dynamics. Part 1: theory’, *Mechanism and Machine Theory (the scientific journal of IFToMM)* .

Legnani, G., Casolo, F., Zappa, B. and Righettini, P. (1996), ‘A Homogeneous Matrix Approach to 3D Kinematics and Dynamics. Part 2: applications’, *Mechanism and Machine Theory (the scientific journal of IFToMM)* .

Li, Y. F. and Wang, J. G. (1999), ‘Incorporating contact sensing in virtual environment for robotic applications’, *IEEE Transactions on Instrumentation and Measurement* **48**(1), 102–107.

Li, Y. F., Wang, J. G. and Ho, J. K. L. (1998), Using physics based models in virtual reality for dynamic emulation of robotic systems, *in* 'Computer Graphics International, 1998. Proceedings', IEEE Computer Society, pp. 388–390.

Li, Z., Ring, P., MacRae, K. and Hinsch, A. (2003), 'Control of Industrial Robots for Meat Processing Applications'. Presented at the ACRA 2003 conference in December 1-3 in Brisbane, Australia.

L.Lieberman and Wesley, M. (1977), 'AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly', *IBM J. Res. Dev* **21**(4).

Lozano-Perez, T. (1987*a*), *AI in the 1980s and Beyond: An MIT Survey*, The MIT Press, Cambridge, MA.

Lozano-Pérez, T. (1987*b*), An algorithm for planning collision free paths among polyhedral obstacles, *in* 'Communications of the ACM', Vol. 22, pp. 560–570.

Lozano-Pérez, T. and Winston, P. H. (1987), LAMA: A Language for Automatic Mechanical Assembly, *in* 'International Joint Conference', Artificial Intelligence.

M. A. Lavin, L. I. L. (1982), 'AML/V: An Industrial Machine Vision Programming System', *Intl. J. Robotics Research* **1**(3), 42–56.

McCabe, R. (2003), 'Robotic Welding'.
URL: *http://www.weldingengineer.com/*

McHaney, B. (2001), 'Automated welding for job shops'.
URL: *http://www.thefabricator.com*

Merriam-Webster's Collegiate Dictionary, 11th Edition (2003). Merriam-Webster, Springfield, MA 01102, USA. ISBN 0877798087.

Meyer, J. (1981), 'An emulation system for programmable sensory robots', *IBM Journal of Research and Development* **25**(6), 955–962.

Meynard, J. P. (2000), Control of industrial robots through high-level task programming, PhD thesis, Linköping Studies in Science and Technology, Linkp̈ing University, Sweden. Tech. Lic. ISBN 91-7219-701-3, ISSN 0280-7971.

Miller, A., Knopp, S., Christensen, H. and Allen, P. (2003), Automatic Grasp Planning Using Shape Primitives, *in* 'Intl Conf on Robotics and Automation', IEEE, Taipai, Taiwan.

Mosemann, H. and Wahl, F. M. (2001), 'Automatic Decomposition of Planned Assembly Sequences Into Skill Primitives', *IEEE Transactions on Robotics and Automation* **17**(5).

Mujtaba, S. and Goldman, R. (1979), 'AL User's Manual'. Stanford AI Memo 323, Stanford University, Stanford, CA 94305.

Munoz, M. A., Rodriguez, M., Favela, J., Martinez-Garcia, A. I. and Gonzalez, V. M. (2003), 'Context-Aware Mobile Communication in Hospitals', *IEEE Computer* **36**(9), 38–46. ISSN 0018-9162.

Myers, B. A. and Beigl, M. (2003), 'Handheld Computing', *IEEE Computer* **36**(9), 27–29. ISSN 0018-9162.

Nilsson, K. (1996), *Industrial Robot Programming*, Dept. of Automatic Control, Lund University, Lund, Sweden. Ph.D. Thesis.

Nilsson, N. J. (1980), *Principles of Artificial Intelligence*, Tioga Publishing Co., Palo Alto, CA.

Nitzan, D. (1990), *Encyclopedia of Artificial Intelligence*, John Wiley & Sons, New York.

Nnaji, B. O. (1993), *Theory of automatic robot assembly and programming*, Chapman & Hall. ISBN 0-412-39310-7.

Nourbakhsh, I. R. (1996), Interleaving, Planning and Execution, PhD thesis, Dept. of Computer science, Stanford University, Stanford, CA, USA.

Olsson, M. (2002), Simulation and execution of autonomous robot systems, PhD thesis, Division of Robotics, Department of Mechanical Engineering, Lund University, Sweden. CODEN: LUTMDN/(TMMV-1051)/1-100/2002, ISBN 91-628-5120-9.

Olsson, M., Cederberg, P. and Bolmsjö, G. (1999*a*), Integrated system for simulation and real-time execution of industrial robot tasks, *in* 'Proceedings of Scandinavian Symposium on Robotics 99', Oulu, Finland, pp. 201–210.

Olsson, M., Cederberg, P. and Bolmsjö, G. (1999*b*), Tele-Robotics for Sensor Driven Industrial Robot Tasks, *in* 'Proceedings of Deneb User Conference 99', Troy, MI, USA.

Olsson, M., Cederberg, P. and Bolmsjö, G. (2002), Integration of Simulation and Execution in Industrial Robot Systems, *in* 'Proceedings of the International Symposium on Robotics, ISR2002', IFR, Stockholm. paper No. 112.

Onda, H., Suehiro, T. and Kitagakiand, K. (2002), Teaching by demonstration of assembly motion in vr - non-deterministic search-type motion in the teaching stage, *in* 'Proceedings of the IEEE/RSJ Intl. Conf. on Intelligent Robots and System', Vol. 3, IEEE/RSJ, pp. 3066–3072.

OROCOS (2004).
URL: *http://www.orocos.org/*

Park, F. C. and Martin, B. J. (1994), 'Robot Sensor Calibration: Solving AX equals XB on the Euclidean Group', *IEEE Transactions on Robotics and Automation* **10**(5), 717–721.

Patel, R. and Tebelius, U. (1987), *Grundbok i forskningsmetodik, (In Swedish)*, Studentlitteratur, Lund.

Paul, R. (1972), Modeling, Trajectory Calculation and Servoing of a Computer Controlled Arm, PhD thesis, Stanford University, Stanford, CA 94305.

Paul, R. (1981), *Robot Manipulators*, MIT Press, Cambridge, MA.

Paul, R. P. (1977), 'WAVE, A model based language for manipulation control', *The Industrial Robot* **4**(1), 10–17.

Pieper, D. L. (1968), The kinematics of manipulators under computer control, PhD thesis, Stanford University, Stanford, CA 94305, Department of Mechanical Engineering.

Pires, J. N. and da Costa, J. M. G. S. (2000), 'Object-oriented and distributed approach for programming robotic manufacturing cells', *IFAC Journal Robotics and Computer Integrated Manufacturing* **16**(1), 29–42.

Popplestone, R., Ambler, A. and Bellos, I. (1980), 'An Intepreter for a Lanugage Describing Assemblies', *Artificial Intelligence* **14**(1).

Raczkowsky, J., Däuber, S., Engel, D., Hoppe, H., Korb, W., Schorr, O., Hassfeld, S. and Wörn, H. (2003), 'Karlsruhe Surgical Robotics Research'. Presented at the ACRA 2003 conference in December 1-3 in Brisbane, Australia.

Ránky, P. and Ho, C. (1985), *Robot Modelling: Control and Applications with Software*, Kempston, Bedford, England : Berlin ; New York : IFS (Publications) ; Springer-Verlag. ISBN 0-903608-72-3.

Rehg, J. (1997), *Introduction to Robotics in CIM Systems*, Prentice-Hall, Upper Saddle River, NJ.

Reif, J. H. (1979), Complexity of the Mover's Problem and Generalizations, *in* 'Proceedings of FOCS', pp. 421–427.

Sacerdoti, E. D. (1977), *A Structure for Plans and Behavior*, Elsevier/North-Holland, New York.

Schaeffer, J. and Plaat, A. (1997), 'Kasparov versus Deep Blue: The Re-match', *ICCA Journal* **20**(2), 95–102.

Sedlenieks, M. (2004), 'MIG/MAG Welding'.
URL: *http://www.linde-gas.com*

Selman, B., Brooks, R. A., Dean, T., Horvitz, E., Mitchell, T. M. and Nilsson, N. J. (1996), Challenge Problems for Artificial Intelligence, *in* 'Proceedings of AAAI-96, Thirteenth National Conference on Artificial Intelligence', AAAI, Menlo Park, California, pp. 1340–1345.

Shimano, B. E. (1979), VAL: A Versatile Robot Programming and Control System, *in* 'COMPSAC 79'.

Shiu, Y. C. and Ahmad, S. (1989), 'Calibration of Wrist-Mounted Robotic Sensors by Solving Homogeneous Transform Equations of the Form AX = XB', *IEEE Transactions on Robotics and Automation* **5**(1), 16–29.

Steil, J., Heidemann, G., Jockusch, J., Rae, R., Jungclaus, N. and Ritter, H. (2001), Guiding attention for grasping tasks by gestural instruction: the gravis-robot architecture, *in* 'Proceedings of the IEEE/RSJ Intl. Conf. on Intelligent Robots and System', Vol. 3, IEEE/RSJ, pp. 1570–1577.

Steinberg, A. N., Bowman, C. L. and White, F. (1999), Revisions to the JDL Data Fusion Model, *in* 'Proceedings of SPIE AeroSense (Sensor Fusion: Architectures, Algorithm and Applications III)', SPIE, pp. 430–441.

Stork, D. G. (2001), 'The end of an era, the beginning of another? HAL, DeepBlue and Kasparov'.
URL: *http://www.research.ibm.com/deepblue/*

Strobel, M., Illmann, J., Kluge, B. and Marrone, F. (2002), Using spatial context knowledge in gesture recognition for commanding a domestic service robot, *in* 'Proceedings of the 11th IEEE Intl. Conf. on Robot and Human Interactive Communication', IEEE, pp. 468–473.

Strommer, W., Neugebauer, J. and Flaig, T. (1993), Transputer-based virtual reality workstation as implemented for the example of industrial robot control, *in* 'Proceedings of the Interface Real Virtual Worlds Conference', Montpellier, France, pp. 137–146.

Taylor, R. H. (1979), 'Planning and execution of straight line manipulator trajectories', *IBM Journal of Research and Development* **23**(4), 424–436.

Taylor, R. H., Summers, P. D. and Meyer, J. M. (1982), 'AML: A Manufacturing Language', *Intl. J. Robotics Research* **1**(3), 19–41.

Thomas, U. and Wahl, F. M. (2001), A System for Automatic Planning, Evaluation and Execution of Assembly Sequences for Industrial Robots, *in* 'International Conference on Intelligent Robotics and Systems', IEEE/JR.

Tsai, R. Y. and Lenz, R. K. (1989), 'A New Technique for Fully Autonomous and Efficient 3D Robotics Hand/Eye Calibration', *IEEE Transactions on Robotics and Automation* **5**(3), 345–357.

Ude, A. and Dillmann, R. (1995), 'Robot motion specification: A vision-based approach', *Surveys on Mathematics for Industry* **5**, 109–131.

Udupa, S. M. (1977), Collision detection and avoidance in computer controlled manipulators, *in* 'Fifth International Joint Conference on Artificial Intelligence, Camebridge, MA', pp. 737–748.

United Nations, U., ed. (2003), *World Robotics*, United Nations. ISBN 92-1-101059-4.

van der Smagt, P. P. (1994), 'Simderella: a robot simulator for neuro-controller design', *Neurocomputing* **6**(2), 281–285.
URL: *citeseer.ist.psu.edu/vandersmagt94simderella.html*

Wahl, F. M. and Thomas, U. (2002), Robot Programming - From Simple Moves to Complex Robot Tasks. Workshop.

Wei, G.-Q., Arbter, K. and Hirzinger, G. (1997), 'Real-Time Visual Servoing for Laparoscopic Surgery', *IEEE Engineering in Medicine and Biology* **4**(16), 40–45.

Wu, H. (2003), Sensor Data Fusion for Context-Aware Computing Using Dempster-Shafer Theory, PhD thesis, The Robotics Institute, Carnegie Mellon University, USA. CMU-RI-TR-03-52.

Zhuang, H. and Shiu, Y. C. (1993), ‘A Noise-Tolerant Algorithm for Robotic Hand-Eye Calibration with or without Orientation Measurement’, **23**(4), 1168–1175.

Zollner, R., Rogalla, O., Dillmann, R. and Zollner, M. (2002), Understanding users intention: programming fine manipulation tasks by demonstration, *in* ‘Proceedings of the IEEE/RSJ Intl. Conf. on Intelligent Robots and System’, Vol. 2, IEEE/RSJ, pp. 1114–1119.