RESEARCH ONLINE

University of Wollongong
Research Online

University of Wollongong Thesis Collection

University of Wollongong Thesis Collections

2014

Automated offline programming for low volume robotic manufacturing

Joseph Polden
University of Wollongong

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au

RESEARCH ONLINE

UNIVERSITY OF WOLLONGONG

COPYRIGHT WARNING

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site. You are reminded of the following:

Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

Automated Offline Programming for Low Volume Robotic Manufacturing

A thesis submitted in fulfilment of

the requirements for the award of the degree

Doctor of Philosophy

by

Joseph Polden, B.Eng. Mech. (Hons.)

UNIVERSITY OF WOLLONGONG

Faculty of Engineering

January 2014

DECLARATION

I declare that this thesis, submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the Faculty of Engineering, University of Wollongong, is my own work unless otherwise referenced or acknowledged. This thesis has not been submitted for qualifications at any other academic institution.

Joseph Polden, B.Eng. Mech. (Hons.)

January 2014.

ACKNOWLEDGEMENTS

The research carried out in this thesis would not have been possible without the support of many different people. First and foremost, I would like to thank the staff and fellow students of UOW's Welding Engineering Research Group (WERG). In particular I would like to extend my thanks to Dr Zengxi Pan, Mr Nathan Larkin and Dr Stephen Van Duin for sharing with me their technical expertise and experience. Additional recognition also goes to Professors John Norrish, Gurcel Alici and Weihua Li who piqued my interest in entering the field of engineering research. I would also like to acknowledge the Defence Materiel Technology Centre (DMTC) and the Welded Structures Foundation (WSF), whose contributions made this research possible.

Finally, I must say that without the love and support of my entire family, none of this would have been possible. Cheers for everything.

ABSTRACT

Programming robotic systems to carry out industrial manufacturing processes is often a difficult, time consuming and costly exercise. The goal of the research presented in this thesis is to develop a means to reduce the time and difficulty associated with programming these industrial robotic systems. This will allow robotic automation to be more easily justified for use in low volume manufacturing scenarios, where frequent re-programming of the robotic devices is necessary.

In this thesis, a novel approach to the efficient programming of industrial robotic systems is presented. This method, termed automated offline programming (AOLP), involves the incorporation of computer automation into the existing offline programming (OLP) methodology. By automating many of the steps in the programming process that were previously performed manually, the overall programming process can be performed in a much more rapid and optimised manner.

Supplementary research in the fields of motion planning and robot path optimisation was required in order to develop algorithms tailored for our application. A review of recent advancements in motion planning algorithms uncovered the variety of different approaches used for planning paths of robotic devices. New and novel algorithms, specifically developed for industrial manipulator style robots, were conceptualised, developed and tested during this phase of the research project. An improved robot path optimisation algorithm is also presented and tested.

The AOLP system developed in this body of work was tested in a real world robotic welding cell. This system was found to drastically reduce programming times for a robotic welding process, when compared to the methods used previously to program the cell. It was found that the developed AOLP system provides significant benefit to the overall operation of the robotic cell, allowing its use to be justified in situations where, in the past, fully manual fabrication was used due to anticipated difficulties associated with programming.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ABBREVIATIONS

In alphabetical order:

AOLP – Automated offline programming

API – Application programming interfaces

AWD – Adaptive weighted distribution

CAD – Computer aided design

C_{space} – Configuration space

C_{free}, C_{forbid} – Free and forbidden configuration space, respectively.

DOF – Degrees of freedom

GUI – Graphic user interface

GMAW – Gas metal arc welding

HMD – Head mounted display

LPRM – Lazy probabilistic roadmap method

LSEA – Lazy Significant Edge Algorithm

OLP – Offline programming

PLC – Programmable logic controller

PRM – Probabilistic roadmap method

P-Sc – Partial shortcut

RRT – Rapidly exploring random tree

SBL – Single-query bi-directional lazy (planner)

SME – Small to medium enterprise

STL – Standard tessellation language

TCP – Tool centre point

VR – Virtual reality

1 INTRODUCTION

1.1 BACKGROUND

Manufacturing performed by small to medium enterprises* (SMEs) makes up a significant portion, approximately 48% [1], of the total value of the Australian manufacturing industry output. This phenomenon can be attributed to the size of Australia's internal market, which is small and dispersed, and to the remote location of Australia relative to the larger global markets [2].

The manufacturing operations of these SMEs can be characterised by two key factors. Firstly, SMEs most often exist in the manufacturing industry due to their expertise in producing items of an intricate or technical nature. Secondly, SMEs mostly manufacture these complex parts in small volumes before either altering their design, or developing entirely new products altogether. Put more broadly, SMEs are generally not interested in the mass production of simple or common products. They have found their niche in manufacturing complex items for use in more technical or specialised applications.

In more recent years, a number of compounding factors have placed significant economic strain on the Australian manufacturing industry. The high value of Australian dollar combined with low productivity growth, rising energy costs and a slowing demand have resulted in a tougher competitive environment with over 100,000 job losses being recorded in the years between 2007 and 2012 [2]. This coincides with drastic expansions in the manufacturing capacity of emerging economies, which has produced many low cost competitors, further increasing the pressure on this sector. As SMEs form a large portion of the Australian manufacturing industry, this is of particular concern as they are less resilient in surviving these periods of economic difficulty.

Much work is being carried out to ensure the on-going economic validity of the manufacturing operations of these SMEs. The research presented in this thesis relates to realising this goal through the efficient and cost effective implementation of robotics-

* The Australian Bureau of Statistics defines a small to medium enterprise as an active business with fewer than 200 employees

based automation. Robotic automation presents many unique benefits for a SME: It offers a means to significantly improve both the speed and quality of the manufacturing output, whilst reducing on-going costs. However the complexity of the parts most often manufactured by these SME's, combined with the small volumes typically produced in each batch, presents series of challenges for the cost effective implementation of these robotic systems. As a result, a large portion of Australian SMEs involved in manufacturing do not make effective use of robotic automation, which carries with it a significant negative flow of effect to the overall manufacturing industry.

The remainder of this Chapter aims to explore this problem by first discussing the general concepts of manual versus automated production and also by characterising some of the difficulties associated with applying automation to a SME-based manufacturing operation. The remainder of the Chapter then presents the research objectives and outline of the work carried out in this thesis.

1.1.1 Manual and Automated Production

For industrial manufacturing, production can be divided into three categories: manual fabrication, fixed automation and programmable automation.

There are many benefits which come as a result of using fully manual labour in manufacturing applications. Most beneficial is the fact that setup costs are low. The equipment and tools required for manual fabrication are minimal and relatively easy to procure, which provides for a low initial capital investment. Furthermore, manual production involves less financial risk than attempting to develop or adapt complex machinery into production. However, manual production typically features high on-going costs related to worker salaries and training. Compounding this is the fact that manual production requires increased personnel, who also typically require a high level of skill (depending on the type of fabrication being performed). When compared to automated production, manual labour generally has a much lower speed of production and greater variance in manufacturing process which can introduce defects into the final product. Manual labour also comes with disadvantages relating to increased risk of injury and material waste. The combination of these factors result in a relatively

constant, but high, unit price of the items being manufactured, regardless of the volume being produced, as shown in Figure 1-1 [3]. Nevertheless, manual labour is still widely prevalent in Australian manufacturing industries primarily due to drastically lower initial capital requirements (compared to automated production), ease of procurement and reduced risk.

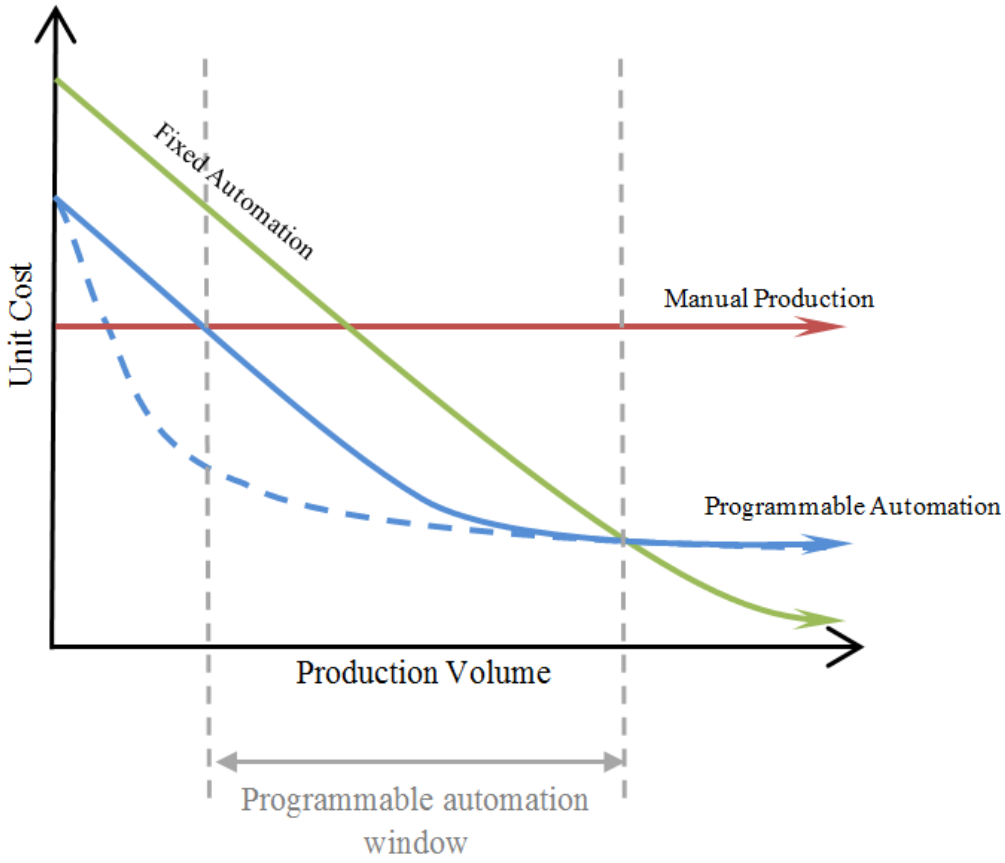


Figure 1-1 Unit cost comparisons for manual and automated production.
Adapted from[3]

Machine-based automation is a common method used for increasing productivity in manufacturing, whilst also helping to reduce or eliminate many of the on-going costs associated with manual fabrication. Today, there are two main types of automation typically used: fixed and programmable automation.

Fixed automation involves the use of specialised machinery to replace human labour in the manufacturing process. This type of automation is referred to as fixed due to the fact that manufacturing processes are stored in the machines configuration in the form of

gears, cams, timing devices and other hardware that is not easily changed or reconfigured for use in manufacturing other items. By being 'fixed', this type of machinery is able to achieve very high rates of production. However this comes at significant cost of large initial investment requirement and low capability of being reconfigured for other uses. The 'fixed' nature of the machinery used in these automation systems is troublesome when either product design changes are made or entirely new products are developed. As a result, fixed automation is typically only economically justified for use when producing items in large volumes, where the high speed of production coupled with the sheer amount of units produced are enough to offset its high investment requirements. In these situations, however, very low unit costs are achievable, as shown in Figure 1-1.

Programmable automation is far more suited for use in applications where production is characterised by the smaller volume of items manufactured in each batch. This is due to the fact that the equipment used in this type of automation is designed primarily for efficient product changeover, rather than for a specialised manufacturing task. As per its namesake, programmable automation features equipment and machinery which has the production sequence encoded into it via programmable code. When design or product changes are made, new or modified program codes can be created and uploaded into the hardware to accommodate these changes.

In a programmable automation system, two categories of equipment are most typically utilised:

- Machinery that is designed for one manufacturing task, but can be easily reprogrammed for product design changes. A good example of this type of device is a CNC milling machine.
- Robotic devices that make use of multiple tooling, allowing them to be re-configured for different manufacturing tasks. These robots can also be re-programmed to accommodate design changes in the items being manufactured.

Whilst the first category of machinery used in programmable automation allow for efficient reprogramming, robotics based manipulators offer a significant step beyond this due to their capacity to be used for multiple manufacturing tasks. For instance, the tool mounted on an industrial manipulator can be changed, allowing the one robotic

device to perform multiple manufacturing tasks such as pick and place operations, or arc-welding. Whereas a CNC mill, whilst it can be reprogrammed easily, will always be confined for use in milling operations.

A robotic manipulator is a specialised robotic device, commonly used for industrial applications. The manipulator structure, shown in Figure 1-2 below, generally consists of a serial chain of links, extending from a fixed base to the manipulators tool centre point (TCP). These robotic devices are equipped with end-effector tooling, which is used to interact with workpieces so that a specific industrial task can be performed.

A manipulator's motion is actuated by electro-mechanical joints, found at the interface between each robot link. These joints are either revolute or prismatic, the number of which defines the overall degree of freedom (DOF) of the manipulator mechanism. Industrial manipulators typically have six revolute joints, which provides the robot with 6-DOF. This allows the manipulator sufficient dexterity to position its end-effector in any of the six individual DOF's of Cartesian space. This dexterity, combined with a large working envelope, is what allows these manipulators to perform a wide array of different manufacturing processes.

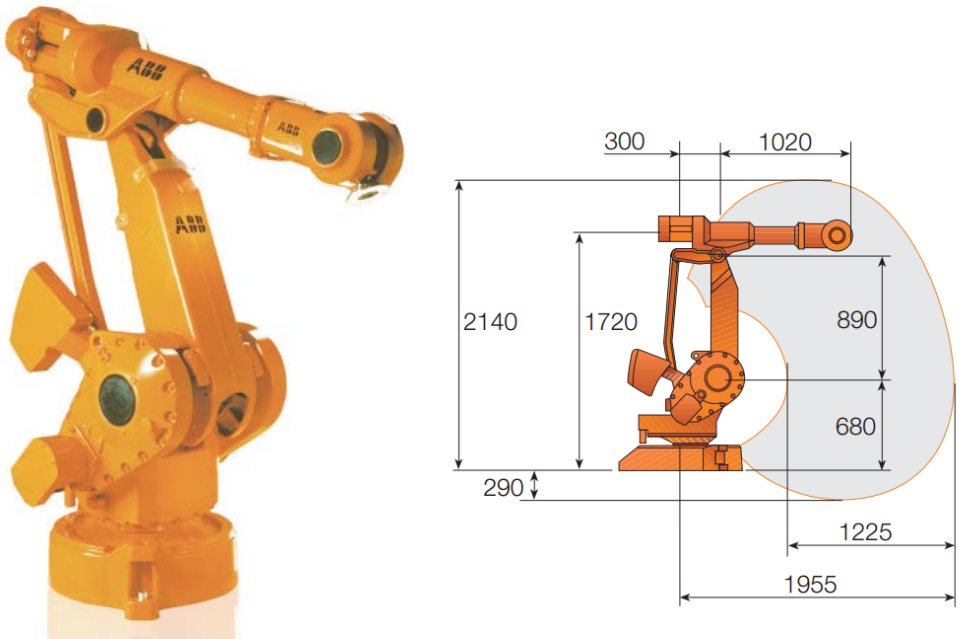


Figure 1-2 An ABB4400 industrial manipulator with six rotational joints [4] (mm).

Whilst this type of robot operates with high repeatability (< 0.2mm for the ABB IRB4400 robot [4]), the accuracy of these robots is poor if not meticulously calibrated for a specific task. This can restrict the use of manipulators for certain applications, such as in the aerospace industry where required machining tolerances are often very high [5].

1.1.2 Programmable Automation for Small to Medium Enterprises

In order to remain competitive, manufacturing operations are turning towards automated production to replace traditional manual fabrication. As discussed previously, there are some key considerations that must be made when developing an automated production system for use in the Australian manufacturing setting:

- A large portion of companies involved in the Australian manufacturing industry are SMEs. SMEs typically have few resources at their disposal and, as a result, they are far more reluctant to implement automation into their manufacturing operations due to perceived financial risk.
- Batch sizes of the items manufactured by SMEs are typically low in volume. After the manufacture of each batch it is common that either an updated design, or a completely new product, is manufactured in the next batch. An automated manufacturing system used in these scenarios must be able to be reprogrammed or reconfigured to for these design changes in an efficient manner.

Fixed automation offers a poor fit for use in these applications. First of all, the required initial outlay of capital for a fixed automation system is very high. The system reconfigurations required with each new batch of items manufactured will prove troublesome, if not impossible, to achieve in an economically feasible manner.

Programmable automation, on the other hand, presents a more intriguing prospect for use in this type of application. The initial capital requirements are generally lower than with fixed automation, as the robotic devices used are more generalised and are mass-produced (as opposed to the often customised nature of the machinery used in fixed automation). The programmability of these automation systems also make them far

more suitable, as they are able to be re-configured for product design changes in an efficient and straight forward manner.

However there still exist many barriers that have prevented the wide-spread adoption of programmable automation into the SME operations in Australia. Reasons for this can be attributed to perceived risks relating to the costs of procuring, setting up, programming and commissioning these robotic systems. Some of the key cost considerations for a programmable automation system are summarised in Table 1-1 below.

Item	Cost	Frequency	Skill Req.
Robotic Hardware Procurement	**High**	Once	Low
Tooling procurement & development	Low	Infrequent	Medium
Hardware Setup	Medium	Once	**High**
Programming	**High**	**Often**	**High**

Table 1-1 Cost evaluation for robotic automation

From this table it can be seen that the programming of these robotic systems poses a significant barrier for cost effective implementation. For complex manufacturing processes, this programming process usually requires both high levels of expertise and significant effort. This is an issue for SMEs as their production volume is typically too small to justify the high cost of this convoluted programming process, which must be performed each time product design changes are made. The result is that, currently, for a complex manufacturing process with a small to medium production volume, very few robotic solutions are used to replace conventional manual fabrication.

The plight of an industry partner involved in this research provides an interesting example of the many difficulties associated with efficiently programming robotic manipulators for an industrial application. In order to keep up with increasing production demand, a robotic welding cell was developed and implemented to replace the manual (human) welding processes used for the construction of an armoured vehicle's hull. Due to the inherent complexity in the design of the workcell,

programming the robotic system to perform these welding operations posed a significant challenge for the team of robot operators and engineers, taking over 6000 man-hours for the task to be completed. Compared to the workcell's production cycle time of 16 hours, programming this cell represents a time that is well over 350 times that of the production cycle.

With new variants of this armoured vehicle expected, the manufacturer understandably wishes to avoid this expensive programming overhead by simply reverting to entirely manual (i.e. by hand) welding processes. This decision comes even though the robotic welding cell is already established. This is but one example commonly encountered in many different manufacturing industries. The economic benefit of automation cannot be realised due to the fact that the anticipated production volume is not large enough to offset the high costs associated with programming the robotic system itself. It is clear that in cases such as this, a more efficient and cost-effective approach to programming these robotic manufacturing systems is urgently needed.

1.1.3 Online and Offline Programming

In modern industrial applications, there are two main categories of robotic programming methods: Online programming (sometimes referred to as lead-through or walk-through programming) and Offline Programming (OLP) [6].

Online programming is a manual process carried out by a human operator. This programming method requires no additional hardware or software, and is done *in situ* with the robotic device. The programmer makes use of a teach pendant, shown in Figure 1-3, to manually move the robots end-effector to a desired position and orientation. These positions can then be stored in the robots memory and organised into a specific sequence. During subsequent replay, the robot is able to retrace the stored sequence of positions to carry out the robot program in a repetitive fashion. This approach to robot programming far more suited for uncomplicated manufacturing processes featuring simple robot paths and geometry. In addition, the quality of the replayed program is limited by the skills of the operator and once the program is generated, it is difficult to

adjust or to make further amendments. In spite of these drawbacks it is widely used in industry due to its intuitiveness, low programming skill requirement and low initial cost.

Figure 1-3 ABB teach pendant used to control a 6-DOF manipulator.

OLP, from a high level stand-point, uses a similar approach to online methods. The major difference, however, is that the entire programming process is carried out in a 3D computer-modelled environment. This approach, although having many benefits, does not always reduce programming overhead. Instead the workload is shifted from robot operators jogging the industrial manipulator in the factory to software engineers, who 'jog' a simulated robot in the virtual computer-based environment, as shown in Figure 1-4. These simulated motions can then be translated into robot code and uploaded into the real world robotic system for use. OLP's strengths are in programming complex systems and are more efficient and cost-effective for large volume production [7]. Compared to online methods of programming, OLP is more reliable and provides additional flexibility when changes in product design occur. Since it relies heavily on the modelling of the robot and workpieces, additional calibration procedures are usually inevitable to meet any process accuracy requirements. Although there are many

different OLP software packages available on the market, employing an OLP system usually requires great programming effort, significant capital investment and a long delivery time. These factors can significantly reduce the feasibility of incorporating OLP into the operations of a SME as their production volume is typically too small to overcome the time and high costs associated in both procuring and effectively utilising commercial OLP software.

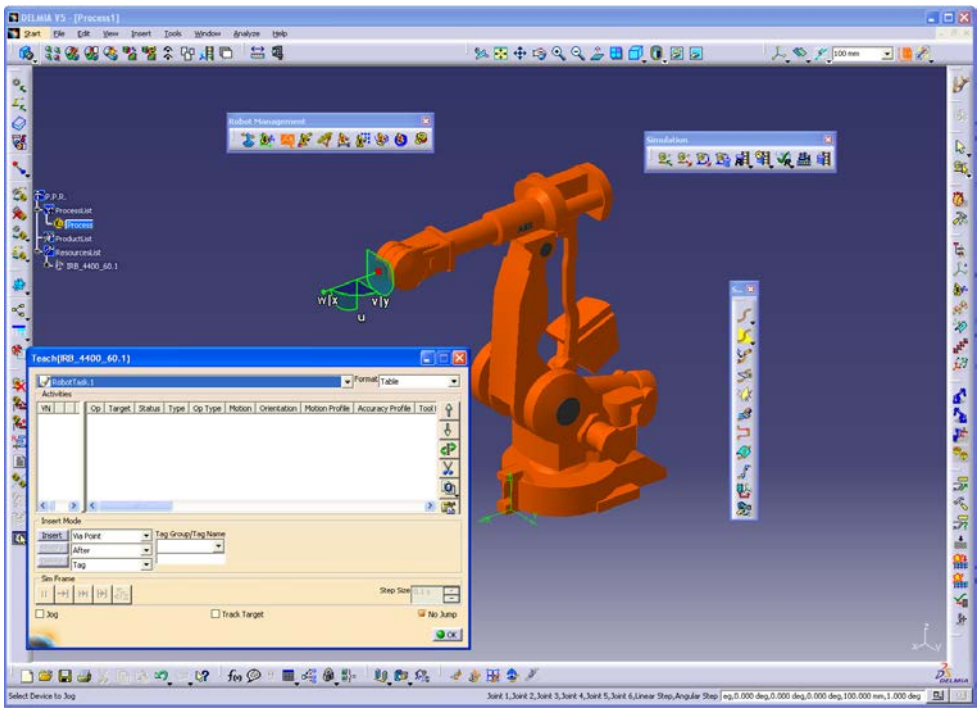


Figure 1-4 The OLP environment of DELMIA.
The window (bottom left) provides a similar function as the teach pendant. Motions can be generated, modified and sequenced for later play back.

1.1.4 Improving Offline Programming for Industrial Robotic Systems

Whilst the OLP method has many benefits, an important observation is that it is still largely a manual process. A computer operator will use an OLP software package to manually generate and simulate various robotic motions, before generating a robot code to upload into the real world system for use. Whilst this approach to programming can provide benefit in many areas, including efficiency and flexibility, it is often still a time consuming and expensive process that requires considerable expertise and skill to carry out. This means that the advantages provided by OLP are generally not sufficient

enough to justify the integration of this technology into the manufacturing operations of SMEs. One must consider, however, that the OLP software commercially available today has only been developed in recent years. This suggests that OLP technology has not matured and is still undergoing significant amounts of research and development.

A novel concept which could improve OLP methods involves integrating computer automation to more effectively carry out the individual steps of the OLP process. It is entirely feasible that computer automation could be utilised to rapidly carry out many of the repetitive and mundane tasks that are performed manually in the OLP process.

Consider, for example, the task of generating robot targets for hundreds of weld seams on a particular work object, such as the vehicle hull presented in Chapter 5. Using a commercially available OLP package, such as ABB RobotStudio or DELMIA, to carry this out will most likely take weeks. This is because the operator not only has to program robot targets for each seam individually, but points for approach and retreat, temperature checks, laser scanning, home positions etc. must also be generated. This procedure is further complicated by the fact that the operator must simultaneously consider task sequencing constraints, optimal robot placements for reachability and mobility, and tool orientations.

However, the repetitive nature of this task makes it an ideal candidate for automation. By replacing human interaction with computer automation, drastic improvements in the effectiveness of OLP can be achieved. For example, instead of the programmer clicking through menus and functions of the software's interface in order to generate a set of weld points, CAD data can be used to automatically generate weld seams at each intersection of two plates and local geometry can then be used to automatically assign optimal weld torch orientations. Furthermore, kinematics-based algorithms can calculate optimal locations to position a robot to carry out these welds whilst adhering to specified joint limits or kinematic constraints. There are also motion planning algorithms that can be used to generate a series of robot motions to guide the robot about its environment without collision. These examples alone highlight the potential to increase OLP's effectiveness by introducing automation into the programming process.

Currently there is no OLP system on the market which has automated the complete OLP process chain, although many components exist separately. For example, DELMIA

provides functions for target creation and trajectory planning. However, these still need be created manually in the OLP environment. Although some components for motion planning and process optimisation, such as collision detection, layout planning and cycle time measurement are available, a complete motion planning function does not exist so this work must be carried out by the human operator. As these processes then rely on the experience and expertise of the user, optimal results cannot be guaranteed.

Developing programming modules that automate many of the time-consuming tasks in the OLP process presents a significant development in robot programming technologies, including online programming and conventional OLP. The entire robot code can be generated in an automatic fashion, taking only a fraction of the time required using current programming methods, which increases the cost-effectiveness of robotic automation enough for low-volume or one-off production. The developed programming modules intend to minimise the programming time and make the system easy to use, even for robot operators without extensive computer programming skills. A design goal of this approach would be to ensure that it can be easily generalized for a large domain of manufacturing applications such as painting, riveting or assembly tasks.

Throughout the remainder of this thesis the programming approach described above will be referred to as automated offline programming, abbreviated as AOLP, in order to capture both the automatic and offline nature of the overall methodology.

1.1.5 Summary

- A large portion of Australian manufacturing is performed by SMEs who typically manufacture complex items in small batches. However these SMEs are under increasing threat from overseas competition as well as increasing operational costs.
- Programmable automation offers an intriguing prospect to assist these SME-based manufacturing operations. However programming these robotic systems is complex, expensive and requires significant technical skill.

- Offline programming has a proven ability to offer many benefits for robot programming. However, current OLP platforms are not suited for SME type applications due to factors relating to costs and development difficulties.
- An approach to develop computer algorithms to automate the OLP process would allow programming to be performed much more rapidly than previously possible, whilst also drastically lowering technical skill requirements. It is anticipated that this approach to programming could improve the cost-effectiveness of robotic automation enough for low-volume or one-off production.

1.2 OBJECTIVES

As highlighted in this introductory Chapter, current practices in robotic programming have resulted in a number of barriers that have prevented the wide spread adoption of programmable automation systems in manufacturing characterised by low volume production. This has a particularly negative effect in the Australian manufacturing industry, as a large portion of the total manufacturing output is performed by SMEs who typically manufacture items in low volumes. This indicates that there exists great potential benefit in the development of more efficient programming methodologies for these industrial robotic systems.

Offline programming is a newer form of robot programming that features a number of intrinsic benefits for programming robots, particularly in industrial applications. The main objective of the work presented in this thesis is to develop a novel programming methodology that can be used to improve the way in which offline programming is performed for industrial robotic applications. The proposed approach is to incorporate computer automation into the offline programming process in order to reduce human interaction and effort and improve on existing programming times. By improving the overall effectiveness of the offline programming approach, its benefits can be enjoyed in applications where its use was not economically justifiable in the past, as depicted graphically in Figure 1-1 (page 3).

The development of this automated programming system for industrial robotic applications will involve contributions in four key areas:

1. **The planning of a conceptual framework for AOLP**. Before an automated approach to offline programming can be developed, there is a need to properly conceptualise an appropriate methodology. This will involve a detailed analysis of how the offline programming process is currently performed by human operators, and evaluating how these processes can be automated via algorithmic processes.

2. **Development of a functional AOLP system.** This relates to the development of the algorithms that will be used to automate each step of the programming process. This task will entail significant amounts of development, testing and debugging of these algorithms before a complete AOLP system can be compiled. Once complete, the effectiveness of this initial AOLP system must be evaluated in a real world robotic manufacturing scenario. A specific design goal of the proposed AOLP system is to ensure that it is developed independent of the target manufacturing process and hardware. This will ensure that it can be re-configured for other industrial tasks involving different robot hardware, thereby providing future opportunities for the developed AOLP system in a large domain of potential applications.

3. **Research and development in the field of motion planning algorithms**. The general field of motion planning algorithms is extensive and has been established for a significant amount of time. However, as it will become clear over the next Chapters, there is no widely accepted consensus as to which approach to motion planning is the most effective. Compounding this is the vast array of end uses these algorithms are developed for. Considering the specific application that forms the basis of this thesis (programming manipulators for industrial processes), significant amounts of background research and testing will be required in order to develop an algorithm that operates effectively enough for real world use.

4. **Develop a means to produce robot paths that are of good quality.** In the purely theoretical applications of motion planning, the goal of the algorithm is typically to simply solve a given motion planning query. However in this work, the paths produced by these motion planning algorithms will be utilised by real world robotic devices to perform complex manufacturing processes. This makes the quality of a generated path an issue of significant importance. To address this, a number of non-trivial issues must be considered, such as: what property of motion must be measured when evaluating an 'optimal' path? And how does one go about producing a path of such quality? These questions relate heavily to how motion planning problems are addressed, however since a large number of different approaches exist, it is considered as a separate research challenge.

1.3 THESIS OUTLINE

Before concluding this introductory Chapter, an outline for remainder of the Chapters presented in this thesis are described.

The first Chapter in the body of this thesis, Chapter 2, presents an in-depth literature review of several important topics related to robot programming. First, a review of recent progress made in the field of robot programming over the last 10 years is presented. In addition, separate reviews in the fields of robot motion planning algorithms and robot path optimisation are conducted.

In Chapter 3, the concepts and methodology behind the proposed automated approach to offline programming are presented. The main focus of this Chapter is to highlight the individual steps of the AOLP process, and to discuss how computer automation can be incorporated in order to improve the way in which they are carried out.

Chapter 4 presents the work done in the field of automated motion planning and optimisation algorithms. This work is an extension of Chapter 3, however since the work carried out in these topics involved significant amounts of research, development and testing, these topics are presented in their own Chapter.

In Chapter 5, a case study involving the development of an AOLP programming system for use in a real world robotic manufacturing application is presented. The Chapter

begins with a summary of the difficulties experienced when programming the existing workcell. The developed AOLP system is then presented, including details of how a user operates the programming system in order to program the robotic devices for a complex welding process.

In Chapter 6, a discussion relating to the effectiveness of the AOLP approach used in the case study is presented. The purpose of this discussion is to highlight the advantages gained from the AOLP approach, as well as to present a brief example of how the developed AOLP system can be reconfigured for use in another robotic manufacturing system. The Chapter ends with a conclusion summarising the work presented in this thesis, which includes a note on recommendations for future work in AOLP.

2 LITERATURE REVIEW

In order to properly present the proposed AOLP approach, a literature review spanning several topics is required. First, a review of recent advancements made in the field of robotic programming technology is presented. In addition, research topics relating to automated motion planning algorithms and optimal path generation are presented. These topics themselves form a wide field of research featuring many different approaches, each with many positive and negative aspects that must be considered in relation to the target application of robotic manufacturing.

2.1 PROGRAMMING INDUSTRIAL ROBOTS

The following Section provides a comprehensive review of research progress in robotic programming methods over the last ten years. It can be seen that the majority of research efforts are focused on providing a suitable robotic programming method for SMEs by improving both online and offline programming methods. Welding (mainly arc-welding) and machining (mainly de-burring), are the most widely investigated processes as welding is the most common task for an industrial robot and machining is considered a challenging application.

2.1.1 Sensor-assisted Online Programming

For industrial robotic applications, conventional online programming methods are widely used today, even though the general method features a number of key drawbacks. Firstly, jogging a robot using a teach pendant is not intuitive as many coordinate systems are usually defined in a robotic system. The operator must always track which coordinate frame the robot is set in when jogging. Guiding the robot through the desired motions accurately while never allowing a collision with an object in the workspace is usually a difficult and time-consuming task, especially when the workpiece has elaborate geometry or the process itself is complex in nature. In addition, when a program is generated, a lot of testing work has to be done before the program is satisfactory in terms of reliability and safety. Thirdly, the robot program generated using the lead-through method lacks flexibility and reusability. If a workpiece is

modified even only slightly, this tedious programming process has to be repeated again. Other drawbacks of lead-through method include the fact that the robot cannot be used for production during the teaching period, the operator is exposed to a hostile environment and the quality of motions taught rely heavily on the skill of the operator.

To address these drawbacks, a number of online programming methods which make use of intuitive human-machine interfaces (HMI) and additional sensory capability have been proposed by several institutions. Table 2-1, below, lists many of these recent research efforts targeted at improving upon conventional online programming methods.

Table 2-1 Sensor-assisted online programming summary

Ref. & year	Sensors	Features	Dependence on additional Markings	Type of Path
[9] 2001	Micro-switch	Assisted jogging	N/A	Any
[8] 2003	Mechanical	Assisted jogging	N/A	Any
[12] 2005	Vision Touch	Simple and robust	Draw on screen	3D plane
[10] 2006	Force	Voice command and PDA interface	No	3D Curve
[13] 2006	Force Vision	Hybrid controller with visual servo	Draw on workpiece	3D Curve
[14] 2007	Vision Laser dots	Closed 3D path	Laser matrix	3D Curve
[11] 2007	Force	Force controller Automatic path learning	No	3D Curve
[15] 2007	Vision Laser	Complete 3D cloud-point	Laser line scan	3D Surface
[16] 2008	Stereo vision	Relies on geometric features	No	2D Curve
[17] 2008	Vision Virtual touch	Combined with CAD model	Draw on workpiece	3D Curve

To make jogging a manipulator in 3D space more intuitive, a few devices which assist the teaching process have been developed for lead-through robot programming. Sugita

[8] presented a method using teaching support devices developed for a de-burring and finishing robot. Two support devices were introduced to measure the position and direction vector of the dummy tool on the tip of the posture measuring unit, which were used to generate a robot program in the robot coordinate system.

Choi [9] presented the development of a force/moment direction sensor named COSMO that can improve pendant-based robot teaching. A 6-DOF industrial robot is taught using a force sensor mounted on the robots wrist. The sensor is hardwired to the robot controller, allowing the operator to hold the sensor with a hand, and move the robot about its workspace by pushing, pulling, and twisting the sensor in the direction of the desired motion. No prior knowledge of the coordinate system is required by the operator. The sensor used in the device is a micro-switch, and this intuitive approach can be implemented at a very low cost.

Schraft [10] proposed an intuitive teaching method to use a lead-through attempt to provide a tool for fast and effective teaching of industrial robots. The user guides the robot with a handle that is equipped with a force/torque sensor and commands the robot using a speech dialog system. The acquired trajectory can be adapted by using a Personal Digital Assistant (PDA) and 3D graphical user interfaces.

Using assistant teaching devices usually results in additional sensors and calibration procedures being added to an already complex robotic system. Pan [11] developed a programming by guidance approach, based on the ABB IRC5 controller with the optional force control feature. Two major functions provided by the commercially available force controller make the entire programming process collision-free and automatic. The first function is a customised walk-through, in which robot is compliant in selected directions (force control directions) and stiff in other directions (position control directions). To change the position or orientation of the robot, the robot operator can simply push or drag the robot with one hand. The second function is called path-learning, in which the robot is compliant in the normal-to-path-direction to make the tool in constant contact the work piece. As the accuracy of the final program is determined by the robot force controller and does not rely on the skill of the robot operator, a 3D robot path with higher accuracy can be generated automatically. This is

of great benefit for applications when process tools have contact with the workpiece, such as in machining processes.

While Pan and Zhang's method still requires jogging during the first stage of programming to guide the robot motion, some other researchers have eliminated jogging from the entire programming process by involving other sensor technologies. Zhang [13] used the same controller platform and extended the concept by adding a visual servo. The system configuration is shown in Figure 2-1. A hybrid position/force/vision control platform was developed to control the robot motion in different directions using feedback from various sensors. The system is able to generate a robot program by automatically following a path marked with a standard marker pen. The position control is used to maintain the tool orientation, vision sensing is used to follow the curve, force sensing is used to maintain contact between the tool and the workpiece.

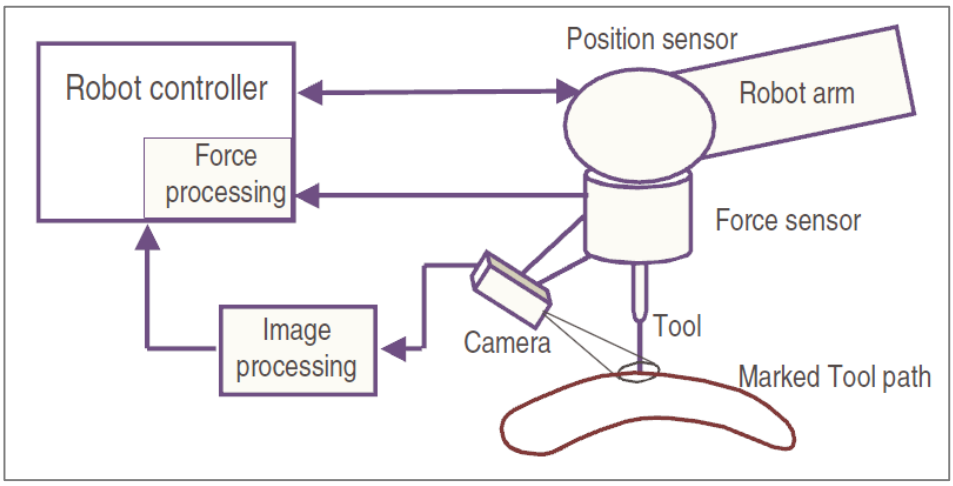


Figure 2-1 Guiding a robot's path with a marked tool path [13].

Solvang [17] also presented a vision-based programming methodology by identifying a path drawn onto the workpiece. This line is captured by a single camera and 2D (X and Y) surface coordinates are extracted from the image data. The depth coordinate (Z) is found by a virtual "hit and withdrawal" procedure using a commercially available simulation program that uses the industrial robot to map the surface of the workpiece. During the mapping process, the robot moves along the existing 2D path and at every point of the path contacts the work-piece surface. When contact occurs, the Z coordinate

is stored establishing the position. Although a CAD model of the workpiece is used, the major part of this method is still sensor-robot interaction rather than offline motion planning.

Nicholson [12] developed a rapid robot programming method which makes use of image data for weld reclamation repair work. Instead of drawing marks on the physical workpiece, the user interacts with the image to select/define a 2D robot path. This selection is done via a drawing module which allows the user to generate a path onto the picture of the workpiece. The Z coordinate is determined using the inbuilt touch sensing functionality of the welding system. Unlike most vision-based systems, which rely on calibration results and are sensitive to lighting conditions, this method provides robust results due to its simplicity.

In some situations, projecting structured light using a laser is more feasible than drawing marks on the workpiece. González-Galván [14] presented work relating to the generation and tracking of closed trajectories over a surface of unknown geometry using structured lighting in the form of a laser spot matrix. Simple image analysis algorithms can be used to detect the centre of laser spots in the images. After the process of surface characterisation is complete, the user selects, in camera-space, a starting point and a direction of reference over the surface for the robot path. As the image plane information gathered from the projection of structured lighting is limited, a second order polynomial function is defined to approximate the 3D curve welding path considering the best fit to the surface. A closed 3D path is achieved by connecting the starting and ending points of neighbouring trajectory segments.

Hu [15] developed a strategy to automate a surface roughening process using structured light and 3D machine vision for object profile perception. The structured light scanning system consists of an analogue camera, laser line generator and a motorised linear slide to provide scanning motion for the camera and laser. Non-Uniform Rational B-Spline interpolation is applied to reconstruct a smooth continuous trajectory from discrete path coordinates.

Stereo vision has also been used to acquire 3D coordinates for robot programming, where distinct features such as corners and edges could be easily identified from a workpiece. Takarics [16] attempted to use the stereo vision technology to program weld

trajectory, based on the intelligent space concept using two fixed cameras. The weld seam is recognised in two images by edge detection algorithms and the path trajectory is generated by a 3D reconstruction of both images. The method is capable of generating a 2D planar curved path for arc welding processes.

Although dramatic progress has been carried out to make online programming more intuitive, less reliant on operator skill and more automatic; most of the research outcomes are not commercial available aside from [13]. This is partially because most of these methods are limited by their specific setups and are yet to be applied to general applications. As cost-effective sensor assisted online programming solutions become commercially available, the installations of robotic automation cells will become more cost effective for SMEs.

2.1.2 Developments in OLP

OLP methods, which utilise 3D CAD data of a workpiece to generate and simulate robot programs, are widely used for automation systems with larger product volumes. Herein the entire robot cell, including the robotic devices, is modelled in 3D. The user programs the robots in this virtual environment, giving them access to many useful operations that aid in the programming process. The programmer can test for reachability, fine-tune properties of robot motion or handle process related information before generating a program that can be uploaded to the real world robot for use.

OLP offers many advantages over online methods:

- Firstly, the programming process does not require actual robot hardware, minimising production down time. Robot programs can be developed earlier in the design/production cycle and programming can be carried out in parallel with production rather than in series with it.
- Secondly, programs generated offline are more flexible than jog-and-teach methods. Program changes can be incorporated quickly by only substituting the necessary part of the program and previously developed routines can be easily included in new programs.

- Thirdly, simulation is usually incorporated into the OLP method. As a result, programs can be pre-checked, thereby confirming the robot's movements which minimises the chance of error and improves productivity and safety. There is also a greater possibility for optimisation of the workspace layout and the planning of robot tasks.

Though OLP has the above-mentioned advantages, it is not popular with SMEs due to its drawbacks. It is difficult to economically justify an OLP system for smaller product volumes due to the high cost of the OLP software package and the programming overhead required to customise the software for a specific application. Development of customised software for OLP is time-consuming and requires high level programming skills. Typically, these skills are not available from process engineers or the operators who often perform online robot programming. As OLP methods rely on accurate modelling of the robot and work cell, calibration procedures using addditional sensors are in many cases inevitable in meeting process requirements.

While OLP software providers place emphasis on making the OLP package more powerful, modular, and flexible to reduce secondary development for specific applications, academic researchers have dedicated attention to improved process planning algorithms and have developed a few OLP software package using open source technology.

2.1.3 OLP Key Steps

OLP is more complex than online programming as the method not only needs to acquire 3D robot targets but also needs to plan subsequent robot trajectories and optimise process sequences. The following Sections aim to outline the OLP process, the key steps of which are depicted in Figure 2-2 below.

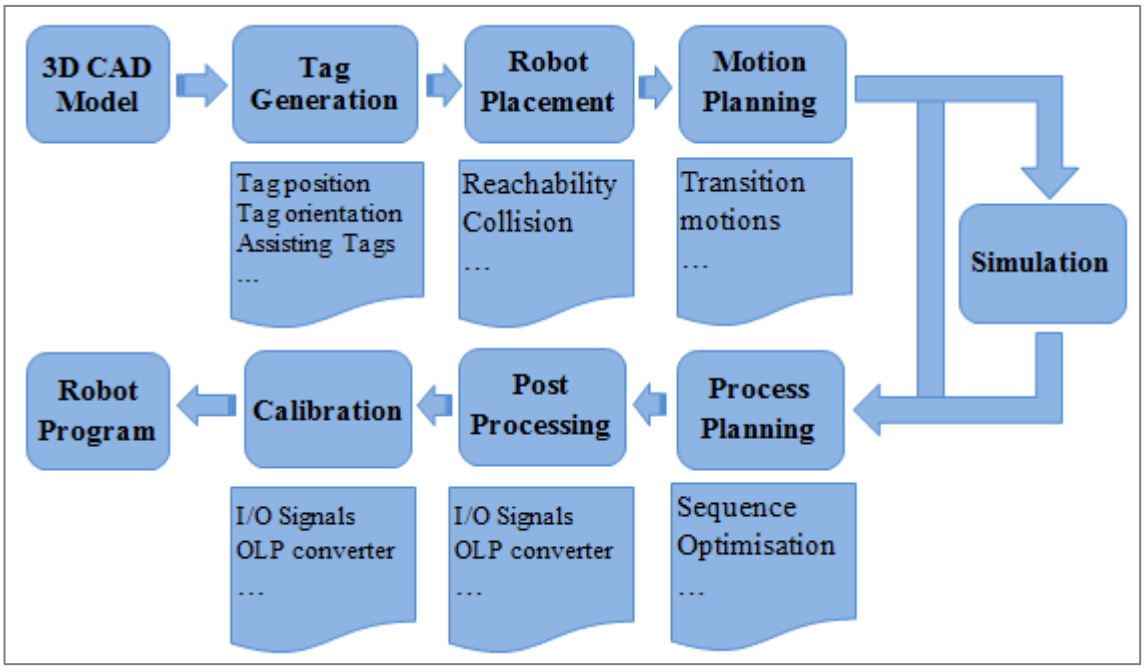


Figure 2-2 The steps of the OLP process.

2.1.3.1 *Generation of CAD*

OLP starts from a 3D CAD model of the workpiece. While it is very common for a product to have a CAD model, for parts without a 3D model or a product that has changed after its CAD model is finalised, there are several methods available to generate the required 3D computer model.

In some situations, a 3D scanner can be used to directly capture the workpiece geometry [18]. The collected points-cloud is converted into a surface model of the workpiece and a smoothing/filtering procedure removes sensory noise before the model can be used for target creation. This approach is also utilised by Larkin in [19], where a 3D time of flight camera is used to capture the layout and configuration of a series of metal plates, which are to be welded together. In other situations, when only 2D CAD data is available, the 3D model of the workpiece can be obtained from either multiple views of a 2D drawing [20], by additional sensors, or the robot is simply programmed in 2D [21].

Although there are various types of CAD files, most modern OLP software packages are capable of converting other types of CAD data to a compatible format. Conversion between different types of CAD files is less of a problem these days with developments in industry.

2.1.3.2 *Target Generation*

This step involves extracting a robot position target from 3D CAD data with a specific robot TCP. It is usually a time consuming process and may require secondary programming for automatic target recognition. OLP software is available that provides built-in functions to generate targets from features, such as corners and edges, from CAD data. The position and orientation information of the tool must be generated from a combination of CAD model and process requirements. Assistant targets such as robot home positions, approach points, and retreat points are also specified manually in a CAD environment. Attempts have been made to automatically extract robot motion information from the CAD data, such as the system proposed in [22].

2.1.3.3 *Robot Placement*

Robot placement relates to locating an optimal position to place the robot so that it can efficiently move to targets generated during the previous step. Since the inverse kinematics of industrial articulated robots usually have multiple solutions in Cartesian space, robot configurations need to be selected considering issues such as reachability, minimising transitional motions, collision avoidance, etc. As most existing OLP software is not able to provide an optimal solution automatically, either manual assignment or secondary software development using Application Programming Interfaces (APIs) is necessary.

2.1.3.4 *Motion Planning*

Motion planning is the process of generating motions that will guide a robot through the surrounding environment, between two individual robot configurations. With OLP software, this is typically carried out manually by the human operator. A series of motions can be generated with various individual motion commands that navigate the robot from an initial configuration to a goal without colliding with any obstacles. In more recent years, motion planning algorithms have been developing to automate this process. As with robot placement, however, most existing OLP software does not include this automated functionality, so it remains a process that must be carried out manually by the user. For DELMIA OLP software, there exists a number of secondary API platforms designed to automatically carry out this motion planning process. These

platforms can be restricted though, as the motion planning problem relies heavily on suitable kinematic models to generate its solutions, meaning that many of the robot's featured within the DELMIA OLP environment are not supported. This secondary API software also reportedly comes at a significant additional cost to the user, which reduces its suitability for many applications.

2.1.3.5 *Simulation*

The ability to simulate generated robot motions is one of the distinct advantages of the OLP approach. Simulation allows a user to pre-check and confirm robot movements, which minimises the chance of error and helps improve productivity and safety. Simulation is also useful in that the suitability of a generated program can be verified without the use of any existing physical robot hardware, reducing the overall downtime of the robotic system. On occasion, simulation is only an optional requirement in the OLP process.

2.1.3.6 *Process Planning*

Planning a complex manufacturing process involves a higher level of optimisation for resource assignment and cooperation of multiple robots to minimise cycle time. As this step is more relevant to the requirement of a specific process, it is rarely available in commercial OLP software. For robotic welding of large structures, the task sequencing of a large number of welds within limited cycle time can be treated as the general "travelling salesman problem", solutions based on a genetic algorithm have been proposed by a few researchers [23-25].

2.1.3.7 *Post Processing*

The post processing stage includes adding necessary input/output (I/O) control signals for equipment in the workcell, smoothing and fine tuning the path if necessary, as well as conversion to the program language of the specific robot hardware. Post processing is more of an issue for generic OLP software as they require compatibility with several different robot manufacturers.

2.1.3.8 *Calibration*

Ideally, a program generated in an OLP system would be downloaded to the robot controller and put into action immediately [26]. In practise, however, the deviation between the actual geometry of elements in the work cell, such as the workpiece, and the nominal geometry of the CAD environment makes calibration almost necessary for all OLP systems.

2.1.4 Existing Robotics OLP software

Robotic manipulators are highly complex systems. Consequently, the development of computational platforms that allow for their precise modelling, and close to real-life simulation of their behaviour, provides a fundamental tool for robot designers, users, and students of the field. This has inspired the creation of numerous graphical software environments from non-robot manufacturers, academic researchers and also from the robot manufacturers themselves.

2.1.4.1 *OLP Software from Robot Manufacturers*

It can be seen from Table 2-2, below, that almost every major robot manufacturer has its own OLP software. The cost of this type of OLP package is generally lower than the generic OLP software variants as the hardware and software is packaged together. Another benefit is that this in-house OLP software is tailored for specific robot hardware, allowing for greater compatibility and reliability. This helps explain why ABB's RobotStudio software package is by far the most widely used OLP software in industry.

2.1.4.2 *Generic OLP software*

This category includes the two most powerful OLP software packages: DELMIA (formally IGRIP and ENVISION with third party add-ons from Kineo, CENIT etc) from Dassault Systems and RobCAD (Em-Workplace) from Technomatix. The advantages these generic packages share are that they are able to integrate and control robotic hardware from different manufacturers. Additionally, they are often able to link into product lifecycle management (PLM) packages to provide production line

optimisation. Major automobile and airplane manufacturers use these packages to integrate robotic systems into their general automated production line. Also, both software packages have virtual reality (VR) functionality, allowing the user to be fully immersed into the simulation environment.

Today, OLP systems are able to do more than just simulate robot trajectories and perform assembly simulation. Simulation technologies are also able to model the interaction of several manufacturing processes, manufacturing resources, and product maintenance issues.

2.1.4.3 *Open Source or Academic OLP Software*

Due to the high cost and limited accessibility of commercial OLP software, a number of research institutions have developed alternative OLP software packages. While some researchers [22, 27-29] have developed OLP packages based on the existing CAD software, such as AutoCAD and Solidworks, others [26, 30, 31] have started from scratch using OpenGL, VRML and Java technology.

Table 2-2 Summary of existing OLP software

Category	software & Ref.	Company and Feature
Generic robotics software	DELMIA (IGRIP, ENVISION), Kineo, CENIT; [18] [32]	Dassault Systems; VR
	RobCAD (Em-Workplace); [33] [34]	Technomatix; VR
	Robomaster	Robomaster
	Robsim; [35]	Camelot
	Workspace 5	Wat Solutions
	Cosimir	Festo
Robotics software from robot manufacturers	RobotStudio	ABB; Most Popular
	MotoSim	Motoman
	KUKA-Sim, CAMrob; [36]	KUKA
	Roboguide	Fanuc
	Wincaps III	Denso
	3D STUDIO	Stäubli
	MELFA WORKS	Mitsubishi
	Pc-ROSET	Kawasaki
	AX on Desk	Nachi
Academic / open source robotics software	[37]	Various MATLAB based software
	[27]	Aristotle University of Thessaloniki, Greece; Based on Solidworks
	[28]	Orebro University, Sweden; Based on standard CAD
	[22] [29]	Based on AutoCAD, Autolisp
	PIN [26]	European Centre for Mechatronics; OpenGL based macro programming
	ROBOMOSP [30]	OpenGL
	[31]	Daegu University, Korea; VRML, Tribon
	RoBott [38]	University of Minho, Portugal; OOP Java

2.1.5 Summary

Conventional online programming is a completely manual process. The robot operator has the freedom to move the robot, select the configuration and plan the process. It is an efficient and cost effective solution for a simple robotic system. However as the target manufacturing processes becomes more complex, the suitability of online programming is significantly reduced. On the other hand, OLP is well suited for these complex robotic manufacturing applications. OLP provides a level of flexibility in its programming that is impossible to achieve with online approaches. If a robot program requires modification, due to changes in setup or design, these changes can be rapidly updated in an existing program and implemented into the real world robot cell. However this flexibility comes at the expense of a typically long and costly setup time, which involves accurately modelling the workcell and robots as well as developing the software to be able to handle the specific manufacturing process requirements. As this setup usually generates a large cost overhead, OLP is only economically justified for production with large volumes, usually by large enterprises.

In the last ten years, extensive research has been carried out on the methodologies for programming industrial robots suitable for SMEs. The boundary between online and OLP methods are becoming blurred as many of the new methods proposed include components from both approaches. Progress in online programming is largely based around sensor and control technologies to assist the operator in generating complex robot motions more easily. Developments in OLP bifurcates into different directions. While the commercial OLP providers are developing more powerful, modular and compatible OLP packages, academic researchers have not given up on low-cost open-source OLP solutions. With the development of more powerful 3D CAD and product lifecycle management software, computer vision and sensor technology, new programming methods suitable for SMEs are expected to grow in years to come.

2.2 MOTION PLANNING FOR ROBOTICS

This Chapter begins by summarising the general motion planning problem. The core principles of motion planning are presented, as well as historical approaches to addressing the general motion planning problem. This leads into a review of sampling-based approaches to motion planning, which are widely considered as a state of the art approach used today.

2.2.1 Introduction

Motion planning is a critical component of the OLP process. Put simply, motion planning is focussed on finding a collision-free path for a robot, guiding it from one given configuration to another. A human's cognitive ability can allow one to easily solve these types of problem; however, developing an automated computer algorithm to do the same thing is much more complicated. This has resulted in vast amounts of research spanning many years, effectively spawning the field of automated motion planning algorithms.

Motion planning algorithms operate in a computer-based environment. In this setting, they are tasked with solving the motion planning problem by generating a valid path from one robot configuration to another. The notion of a 'valid path' refers to a path that satisfies a set of constraints; which can vary, but commonly relate to:

- Collision avoidance
- Orientation or configuration limits
- Speed and acceleration

Throughout the remainder of this thesis, individual robot configurations or robot paths that are found to lie entirely in C_{free} and do not infringe upon any manipulator-specific kinematic constraints (joint limits and singularity) will be referred to as *valid*.

Motion planning problems are presented in either multiple or single query formats. Multiple query approaches are used when many different path planning problems will be queried within the same static environment. In these situations the algorithm can be allowed to spend a great deal of time modelling the surrounding space to a high degree

of accuracy. This allows for new queries presented to the algorithm to be solved rapidly and efficiently. Single query approaches are necessary when either the geometric shape of the surrounding environment or the robot change frequently. With each change in geometry, the problem must be re-calculated in some way so as to ensure that collision will not occur. Single query problems dominate the research sphere, as the need to rapidly re-calculate a collision-free path in the most efficient way possible presents a significant problem with many possible approaches.

The time taken for a motion planning algorithm to generate a solution depends on the type of the robot used and the complexity of the environment that is to be navigated. For instance, Kuffner [39] demonstrated an algorithm that is able to solve motion planning queries for a robot operating in a 2-dimensional (2D) environment in only fractions of a second. Whilst Sanchez [40] required hundreds of seconds to find a valid path for a multiple manipulator type problem, even though a state-of-the-art planning algorithm is used.

The type of robot being used also has significant bearing on the difficulty of a motion planning problem. A seemingly limitless array of different types of robots currently exist, with many new types also being developed. The structure or shape of these robots is not so important; these factors can be accounted for by the planning algorithm. What must be considered is the mode of motion that the robot utilises to change its configuration; namely whether the robot uses holonomic, or non-holonomic motion.

Early approaches to motion planning relied on explicitly modelling a robot's surrounding free space, so that a mathematical approach to navigating it could be applied. Some of these algorithms, such as [41], proved effective in low dimensional problems. However as motion planning problems became more complex, the computation time required to generate a solution using these explicit methods became too long to be of any practical use. This severely limits their functionality in real world industrial applications, such as planning motions for high DOF manipulators, or navigating geometrically complex environments.

A breakthrough came with the development of probabilistic, sampling-based, motion planning algorithms. These approaches sacrifice a complete understanding of an environment in favour of incrementally capturing a simplified representation of the free

space surrounding the robot. These sampling-based methods bypass the need to generate an explicit model of the surrounding environment, drastically reducing the computational load needed to solve these more complex problems.

2.2.2 Sampling-Based Motion Planning

Compared to early explicit approaches, the operation of sampling-based motion planning algorithms is relatively simple. This, however, does not take away from their flexibility or ability to solve complex problems. They have been used effectively in a range of applications, such as multi-robot queries [40], assembly and disassembly tasks [42], manipulation handling [43] and computational biology [44].

2.2.2.1 *Robot Configurations and Configuration Space*

Sampling-Based planners operate on the premise of a robot moving about its surrounding configuration space, C_{space}. As shown in Figure 2-3, a robot configuration q is the set of parameters required to fully define the position of the robot in space (be that 2D or 3D). Using this concept, C_{space} is then the set of all possible configurations a robot can make within its work envelope.

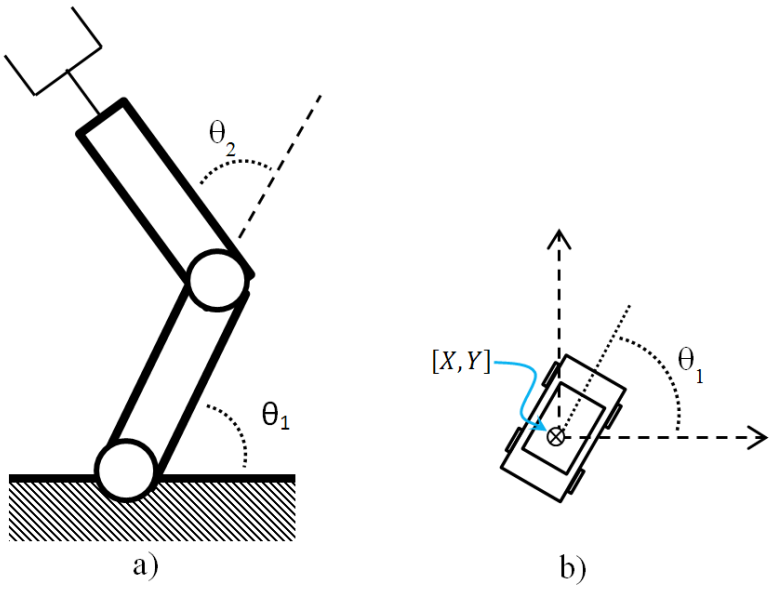


Figure 2-3 Defining configurations for two different robotic devices.
a) The configuration of the planar manipulator is defined by joint angles θ_1 and θ_2. b) The configuration of the car-like robot is specified by its position [X,Y] and its orientation θ_1.

2.2.2.2 *Collision Checking and Local Planning*

A critical component of any sampling-based motion planner is its collision processing algorithm, which uses geometric calculation to determine whether a particular robot configuration is in collision with any obstacles in its immediate vicinity. A collision processing algorithm is used to split C_{space} into two distinct subsets: C_{free} and C_{forbid}. C_{free} represents the free configuration space of the robot; the set of all robot configurations that are not in collision with any surrounding obstacles. In contrast, C_{forbid} represents the forbidden configuration space; the set of all robot configurations that are in collision with any obstacles in the robots workspace.

$$C_{Space} = C_{free} + C_{forbid}$$

Another particularly important component of a motion planning algorithm is *the local planner*, which utilises collision checking processes to determine whether a robot motion between two given configurations lies entirely in C_{free}. This is performed by interpolating a number of discreet robot configurations between the two given configurations, and then checking each interpolated pose for collision. If all generated configurations are found to be valid, then it is concluded that the resulting motion is C_{free}. Particular importance must be placed on the number of configurations to be interpolated. If this number is too small, it is possible that a certain collision may not be sampled. Conversely, if too many configurations are interpolated, the time taken to sample a given edge will be unreasonably high. Kavraki [45] presented an approach in which the line segment is "interpolated into m individual configurations $q_1, \dots, q_m$, such that for each pair of consecutive configurations (q_i, q_{i+1}) no point on the robot, when positioned at configuration q_i, lies further than some specified distance ε away from its position when the robot is at configuration q_{i+1}. The parameter ε can then be tuned for optimal performance, given the particular robotic set up. The type of interpolation performed can be carried out in a number of different ways. For example, if the robot travels with non-holonomic motion, then these constraints must be reflected in the interpolated path. In most applications, paths are interpolated in a simple linear fashion. Using the same convention as robot configurations, a motion that is found to be both kinematically feasible and collision-free is also referred to as *valid*.

2.2.2.3 *Network Graphs*

Using the aforementioned concepts, the motion planning problem is formally defined by Tsianos [46] as:

Given an initial and a goal configuration, $(\mathrm{q_{init}}, \mathrm{q_{goal}}) \in \mathrm{C_{free}}$*, find a continuous path*

$$\mathrm{p}: [0,1] \rightarrow \mathrm{C_{free}} \textit{ where } \mathrm{p}(0) = \mathrm{q_{init}} \textit{ and } \mathrm{p}(1) = \mathrm{q_{goal}}.$$

The general approach taken by sampling-based motion planning algorithms is to utilise collision checking processes to capture a set of valid robot configurations and motions, which are then used to build a simplified representation of C_{free}. If carried out to a suitable degree, this simplified model can be used to efficiently solve a given motion planning problem. This simplified approach is in contrast to early computational methods of motion planning, which attempt to build an explicit or complete representation of C_{free}. These approaches become problematic due to the difficulties associated with generating a complete representation of C_{free} in high dimensional environments.

In sampling-based algorithms, the simplified model of C_{free} is generally represented with a network graph, often referred to as a roadmap, consisting of nodes and edges $G = (N, E)$. Nodes of the graph are used to represent valid robot configurations, whilst edges are used to represent valid motions captured by the local planner, as shown in Figure 2-4 below. The use of this notation provides a simple and effective means to categorise, describe and analyse the operation of the many different sampling-based motion planning algorithms that exist.

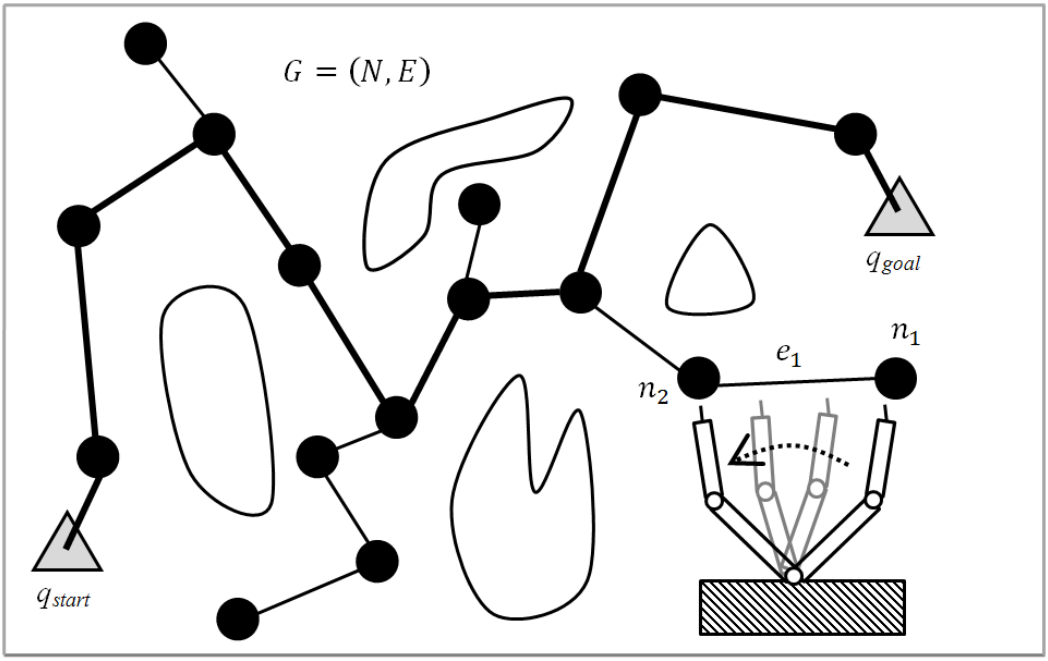


Figure 2-4 A typical roadmap graph $G = (N, E)$.
Valid robot configurations at n_1 and n_2 are represented in G as nodes: $(n_1, n_2) \in N$. The valid motion between n_1 and n_2 is added as an edge: $e_1 \in N$.

Using this network graph structure, a given motion planning problem is solved by generating a simplified representation of C_{free}, which links the given start and goal nodes together. This construction can be carried out in a number of different ways, which is typically done in an incremental fashion until the initial and goal nodes are found to be contained within the same connected network component. A graph searching algorithm, such as the Dijkstra [47] or A* algorithm [48], can then be used to return the minimum-cost path which joins them. Different robot types will have different modes of locomotion, so deciding the cost of a given motion represented by an edge of the graph is of considerable importance. For example, in [49] the edge cost for a mobile robot summed over two weighted components: The linear cost component C_L and the rotational cost component C_R:

$$C_{edge} = K_L \cdot C_L + K_R \cdot C_R$$

C_L is taken as the Cartesian distance travelled in traversing an edge, whilst C_R represents the amount of rotation made by the robot. The constants K_L and K_R are used to bias these terms to achieve a suitable overall edge cost for the given robotic set up. Other popular methods assigning a cost relative to the time taken to traverse the edge [50], as

well as methods which relate the cost to the swept volume made by the robot [51]. By doing this, the distance and rotation factors are both included in the cost evaluation.

For a robotic manipulator, a simple approach is to calculate the Euclidean distance travelled by the robots TCP. A more effective method of defining the cost of a given path is to evaluate the total displacement made by each joint. These displacements can be used to generate a weighted sum cost, which is biased towards the manipulators major joints (J_1, J_2, J_3) over the minor joints (J_4, J_5, J_6). Generally, this distribution works well as the optimisation goal is to minimize total Cartesian motion. Once the minimum cost path linking the start and goal nodes together has been returned, a post process algorithm is then able to translate it into a series of linked robot motions that can be uploaded into the real world robot system for use.

2.2.2.4 *Probabilistic Completeness*

Sampling-based planners have proven to be highly successful, which can be largely attributed to their effective use of randomised sampling to capture a simplified representation of C_{free}. This randomness ensures these planners have a quality known as probabilistic completeness. Svestka describes this property in [52]:

"A path planner is referred to as probabilistically complete if, given a problem that is solvable in free configuration space, the probability that the planner solves the problem approaches one as the running time approaches infinity. Hence, a probabilistically complete path planner is guaranteed to solve such a problem, provided that it is executed for a sufficient amount of time"

Probabilistic algorithms suffer from the fact that as C_{free} is only approximated with a simplified representation, they cannot explicitly recognise if a solution to a given problem is geometrically impossible. If some form of termination protocol is not implemented in these situations, the planner will effectively run forever.

The different variants of sampling-based motion planning algorithm share many of the same components, such as a local planner and network graph structure, to solve a given path planning problem. However, a major distinction between the different variants comes from the way in which the planner explores its surrounding C_{space}. The next two

sections of this Chapter will review the two most popular categories of sampling-based motion planning algorithms: Tree-based planners and roadmap-based planners.

2.2.3 Tree-Based Algorithms

Tree-based planners are a group of planners that construct their network graph as a tree like structure emanating from a start configuration towards its goal. Many different variants of this type of approach exist (see [39, 41, 53-56]), most making use of either one or two trees to solve a given path planning query.

Tree-based planners were the first planners to make effective use of a randomised, sampling-based, approach to motion planning. The Randomised Path Planner (RPP) [41], developed by Barraquand and Latombe, is an early example of particular note. This planner makes use of potential field functions to guide the robot towards a specified goal state. If the potential function becomes trapped by local minima, blocking the progress of the robot, a random walk algorithm is initiated. This is used to free the robot from the minima, so that the potential functions can be re-started and the motion planning process can continue. These procedures are repeated until the potential function is able to attain its goal state. This algorithm is widely credited with proving the effectiveness of probabilistically complete algorithms in solving complex motion planning problems.

The introduction of probabilistically complete algorithms, such as the RPP planner [41] or the Ariadnes' Clew algorithm [53], stimulated a flurry of research into tree-based motion planning. LaValle's Rapidly Exploring Random Tree (RRT) [56] algorithm provides another landmark approach to planning whose general concepts and method are widely used today. The RRT algorithm removed the use of potential functions in favour of completely randomised exploration, providing a simple means for solving complex problems. The algorithm also proved popular due to the ease in which non-holonomic motion constraints could be adapted into the planning process. The initial RRT algorithm was presented as an undirected search of C_{space}, as shown in Algorithm 2-1. An RRT is grown from an initial configuration, or root, which is incrementally expanded over a set number of iterations. With each iteration, a random configuration

qrand is generated and an attempt to grow the tree towards it is made with the EXTEND function. The EXTEND function, depicted in Figure 2-5 below, first finds the nearest node, q_{near}, already in the RRT. The local planner then attempts to sample a valid motion between these two nodes, which is capped by a fixed incremental distance ε. Three possible outcomes of this expansion can occur: *Reached*, where q_{rand} is added to the RRT as it lies within ε of a node in the RRT; *Advanced*, where a new node $q_{new} \neq q_{rand}$ is added to the RRT; or *Trapped*, where the new vertex is rejected as the motion between q_{near} and q_{rand} is found to be invalid by the local planner.

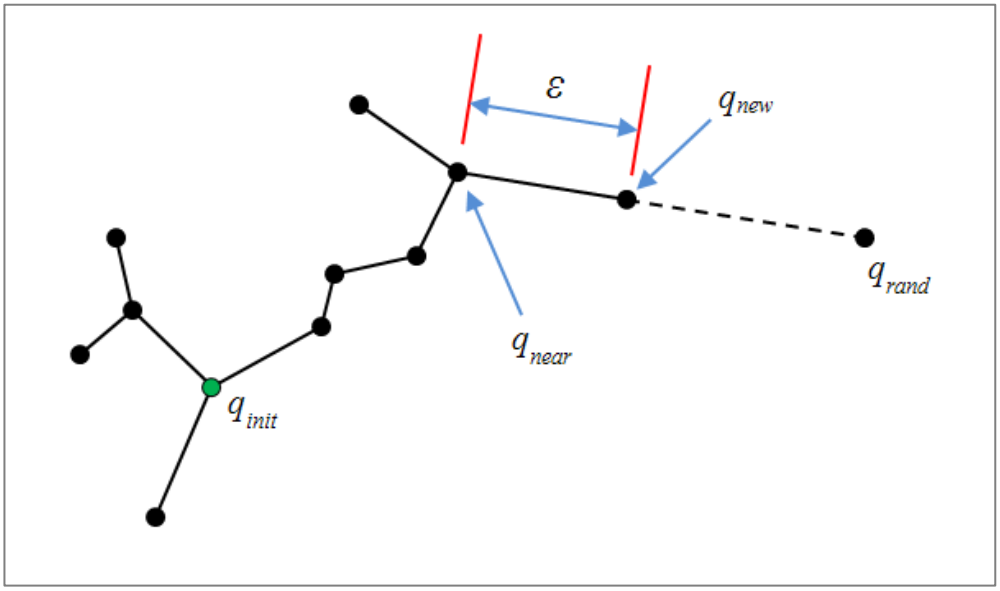


Figure 2-5 Graphical representation of EXTEND function. Adapted from [39].

Algorithm 2-1 The undirected RRT algorithm.

Build_RRT(q_{init})
1. T.init(q_{init})
2. **for** k = 1 **to** K
3. q_{rand} ← RANDOM_CONFIG()
4. EXTEND(T, q_{rand})
5. **end for**
5. **return** T

Algorithm 2-2 The RRT's EXTEND function.

EXTEND(T, q)

1. $q_{near} \leftarrow$ NEAREST_NEIGHBOR(q, T)
2. **if** distance$(q, q_{near}) > \varepsilon$ **then**
3. $q_{new} \leftarrow$ New_Config$(q, q_{near}, \varepsilon)$
4. result ← *Advanced*
5. **else**
6. $q_{new} \leftarrow q$
7. result ← *Reached*
8. **end if**
9. **if** LocalPath$(q_{new}, q_{near}) \in C_{free}$ **then**
10. *T.add_vertex(q_{new})*
11. *T.add_edge(q_{new}, q_{near})*
12. **return** result
13. **else**
14. result ← *Trapped*
15. **return** result
16. **end if**

The general RRT approach is probabilistically complete and simple in operation, making it a popular choice for many different practical applications [57-62]. One of the most beneficial qualities of the RRT is that its incremental growth is biased towards exploring the largest regions of unexplored space. This phenomenon is explained with an analysis of the Voronoi regions generated by the nodes of a RRT, as shown in Figure 2-6. Three aspects of these generated Voronoi regions can be used to assess the qualities of the explored space:

- Firstly, the relative sizes of the largest cells give us information about the dispersion of the sampled configurations belonging to the RRT.
- Secondly, the locations of these largest cells provide information about the regions of space that have not been explored well.
- Finally, the range of sizes of the Voronoi regions tells us about the coverage of the RRT. If all Voronoi regions have a similar size, it can be deduced that the RRT has covered the environment extensively. On the other hand, if a large range of sizes are present, it is clear that the RRT has explored some regions of the configuration space more thoroughly than others.

As new nodes are generated across the robots C_{space} uniformly at random, the probability that a given tree node will be selected for expansion is directly proportional to the volume of its corresponding Voronoi region. This means that an RRT's growth will be biased towards making its expansions from nodes located at the frontier of the RRT, into the unexplored space contained within the largest Voronoi regions.

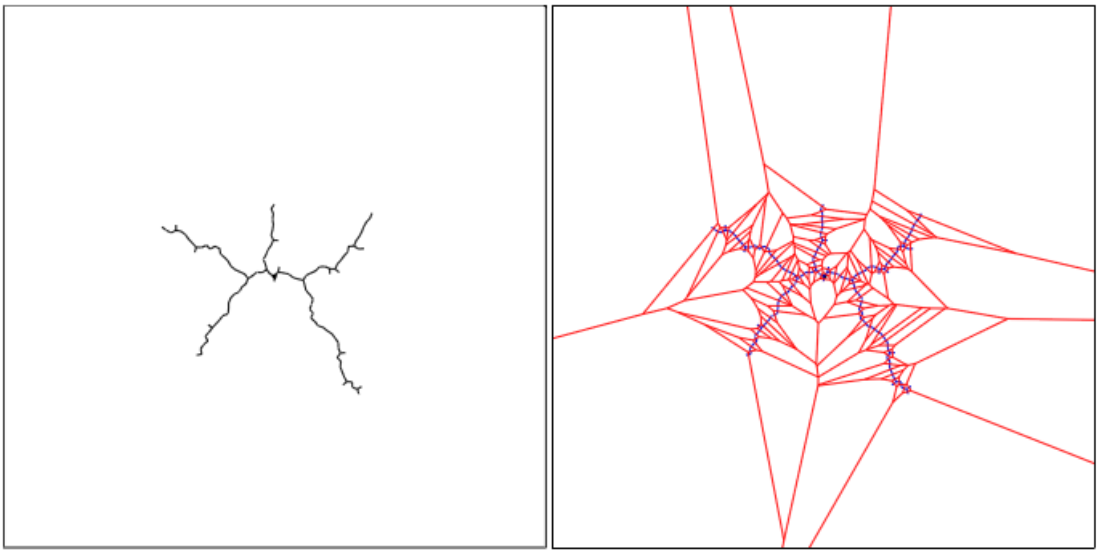


Figure 2-6 The Voronoi regions generated by an RRT.
These provide detailed information about the explored space, from [39].

The RRT algorithm presented in Algorithm 2-1 is used as a general exploration of C_{space}. However this algorithm can be easily adapted into a directed search suitable for solving typical motion planning problems. A simple approach is given in Algorithm 2-3, where Algorithm 2-1 is modified to include the function CHOOSE_TARGET described via pseudo code in Algorithm 2-4. This effectively changes the search from a purely random exploration, to exploration that is biased towards the goal configuration with a given goal sampling probability. The search will then continue until the RRT grows within a certain threshold distance ε of the goal configuration.

Algorithm 2-3 Directed RRT search algorithm.

```
Build_RRT(q_init)
1. T.init(q_init)
2. for k = 1 to K
3.     q_Target ← CHOOSE_TARGET()
4.     EXTEND(T, q_Target)
5.     if distance(T, q_Target) < ε then
6.         return path(q_init,q_goal)
7.     end if
8. end for
5. return T
```

Algorithm 2-4 The choose target function.

```
CHOOSE_TARGET()
1. ρ ← Random_Number([0,1.0]
2. if ρ < goal_sampling_prob then
3.     return q_goal
4. else
5.     return RANDOM_CONFIG()
6. end if
```

Algorithm 2-5 presents the RRT-Connect algorithm, which makes use of the function CONNECT shown in Algorithm 2-6. In this approach, two trees are grown from the initial and goal configurations with the objective of connecting them to solve the given problem. As both trees make effective use of Voronoi bias to explore C_{space}, this approach provides a drastic improvement to single tree RRT variants. With each iteration, the primary tree is grown in identical fashion to the original RRT algorithm. If the RRT expansion is possible, an attempt to join the two trees together with the CONNECT function is made. The CONNECT function operates in the same way as EXTEND, however there is no maximum distance that limits the expansion. The expansion continues until either collision is detected, or a successful connection is made. If the trees are not joined together, the primary tree is switched, and the motion planning process continues.

Algorithm 2-5 The RRT-Connect algorithm [39].

```
RRT_Connect_Planner(q_init, q_goal)
1.  T_a.init(q_init); T_b.init(q_goal)
2.  for k = 1 to K
3.      q_rand ← RANDOM_CONFIG()
4.      if not (EXTEND(T_a , q_rand) = Trapped) then
5.          if (CONNECT(T_b , q_new) = Reached) then
6.              Return PATH(T_a , T_b)
7.          end if
8.      end if
9.      SWAP(T_a , T_b)
10. end for
11. return Failure
```

Algorithm 2-6 The connect function.

```
CONNECT(T , q)
1. repeat
2.     S ← EXTEND(T , q)
3. until (S ≠ Advanced)
4. return S
```

Several different variants of the bi-directional RRT algorithm have been proposed, one of the most efficient being the RRTConCon approach [39]. In this variant a more aggressive search is performed by replacing all occurrences of EXTEND with the CONNECT function. In this way, a greedy search is often able to more rapidly find a solution, which comes at the expense of the quality of the returned path.

Hsu presented an RRT-based single-query motion planning algorithm for use in expansive configuration spaces [54]. Instead of relying on the inherent Voronoi bias that RRT methods possess, nodes to be expanded are selected from the RRT on the notion of 'expansiveness'. Each node in the tree is given a weight relating to the density of other nodes surrounding it, within a certain radius. With each iteration, the probability a node is selected for expansion is taken as the inverse of its assigned weight. A number of new configurations are sampled about this selected node, and the tree is expanded from the selected node towards a portion of these newly generated configurations. The RRT and Expansive Spaces planners both attempt to expand into unexplored space, the critical

difference between the two is in the techniques used to identify these regions. The Single Query Bi-Directional Lazy (SBL) [40] planner also uses node densities as a weighing method biasing the RRT's expansion. In this approach, however, 'lazy' methods of collision checking are carried out in order to reduce the overall number of collision checks required to solve a given problem. This lazy approach distributes collision-checking processes across the entire path, rather than a piece-wise collection of local paths sampled between nodes. This results in a scheme that first tests an entire path for collision at very rough resolutions, which is then refined as paths are found to be collision-free. This means the planner will only collision check a path at the finest resolution when it is fairly certain that the path will be collision-free.

RRT methods rely heavily on the use of a suitable distance metric to solve motion planning problems. A poorly tuned distance metric can result in drawn-out planning times, or in worst-case scenarios can lead to the tree growing in the wrong directions. RRT algorithms also experience difficulty in navigating thin obstacles and narrow passages. The incremental nature of the RRT's growth often makes it very hard for them to find the entrance to a passage. Compounding this, if a collision-free sample is somehow generated inside the region, but cannot be connected to an RRT, it is discarded. This effectively wastes a valuable opportunity for progressing the growth of the RRT. The planner presented by Strandberg [63], attempts to address this by keeping these useful samples and sprouting new RRT's from them. These local trees are then added into the growth cycles of the start and goal RRT's. This approach has been found to be useful provided a suitable upper bound on the allowable number of local trees is specified.

2.2.4 Probabilistic Roadmap Methods

The Probabilistic Roadmap Method (PRM) is another sampling-based motion planning algorithm that utilises both a network graph structure and the principles of probabilistic completeness. Since the development of the PRM concept, the approach has become widespread in industry due to its relative simplicity coupled with its effectiveness in more difficult applications. PRM planners are also easily adapted for almost any robot type, and have documented uses in a wide array of different applications [40, 42, 44, 64,

65]. As opposed to the RRT approach which samples between two given configurations, the general PRM approach first samples the entire configuration space of the robot, before plotting a suitable path to solve a given query. Whilst this approach may be significantly slower than the RRT method for simple queries, it allows for a more detailed sampling of C_{space} which produces better results when more difficult motion planning problems need be solved.

The work that laid the foundation for the PRM planner was developed concurrently at several different locations, before being assembled into a set methodology by Kavraki, Svedska, Latombe and Overmars [45, 64, 66]. The PRM planner differs from RRT type planners in a few fundamental aspects. Firstly, the PRM planner was primarily developed for use over multiple queries, as opposed to the single-shot approach of RRT planners. The PRM planner is typically utilised to exhaustively capture a good representation of C_{free}, before using this model to answer multiple motion planning queries. The way in which C_{space} is explored by PRM planners is also fundamentally different to RRT's. The PRM approach to sampling encompasses the entire expanse of the robot's C_{space}, as opposed to only sampling the space between two given configurations. As well, PRM planners often make use of heuristic techniques to identify difficult regions of C_{space}, allowing subsequent sampling strategies to be directed accordingly. A summary of many of these heuristic approaches is given in Section 2.2.5. It is generally accepted that PRM planners are often more effective at solving the more challenging motion planning problems.

The PRM approach is carried out over two distinct steps referred to as the construction and query phases, respectively. The construction phase, shown in Algorithm 2-7, is solely dedicated to building the roadmap graph G, which is used to solve the given motion planning query. This is done by sampling a set number of collision-free robot configurations in C_{space}, which are represented as nodes in G. Once the predefined number of free configurations has been generated, a local planner is then used to sample valid robot motions between nearby nodes. Collision-free motions are added to G as edges that link the respective nodes together. The number of C_{free} configurations to initially sample (N_{init}), and the number neighbours to attempt to connect to (N_{neighb}) have great effect on the overall effectiveness of the PRM approach.

Once the construction phase is complete, the query phase is initiated. In this phase, the initial configuration, q_{init}, and the goal configuration, q_{goal}, are linked into G using the local planner. If this succeeds, the motion planning problem has effectively been reduced to a graph search: The shortest path through G which links the start and goal node together can be found, solving the given query. If no solution can be found through G, additional sampling is done in an attempt to connect the disjoint components together. A graphic overview of the general PRM methodology is shown in Figure 2-7.

Algorithm 2-7 PRM construction phase.

PRM_Construction( );

1. *initialise* $G = (N, E)$
2. $N \leftarrow \emptyset$; $E \leftarrow \emptyset$
2. **loop**
3. $q_{rand} \leftarrow$ RANDOM_CONFIG( )
4. $N_c \leftarrow$ Set of nearest neighbours of q_{rand} chosen from N
5. $N \leftarrow N \cup \{q_{rand}\}$; *add* q_{rand} *to* N
6. **for all** $n \in N_c$ *in order of increasing* $D(q_{near}, n)$ **do**
7. **if not** same_connected_component(q_{near}, n) **then**
8. $E \leftarrow E \cup \{(q_{near} \leftrightarrow n)\}$; *add edge to E*
9. *update connected components of G*
10. **end if**
11. **end for**
12. **end loop**

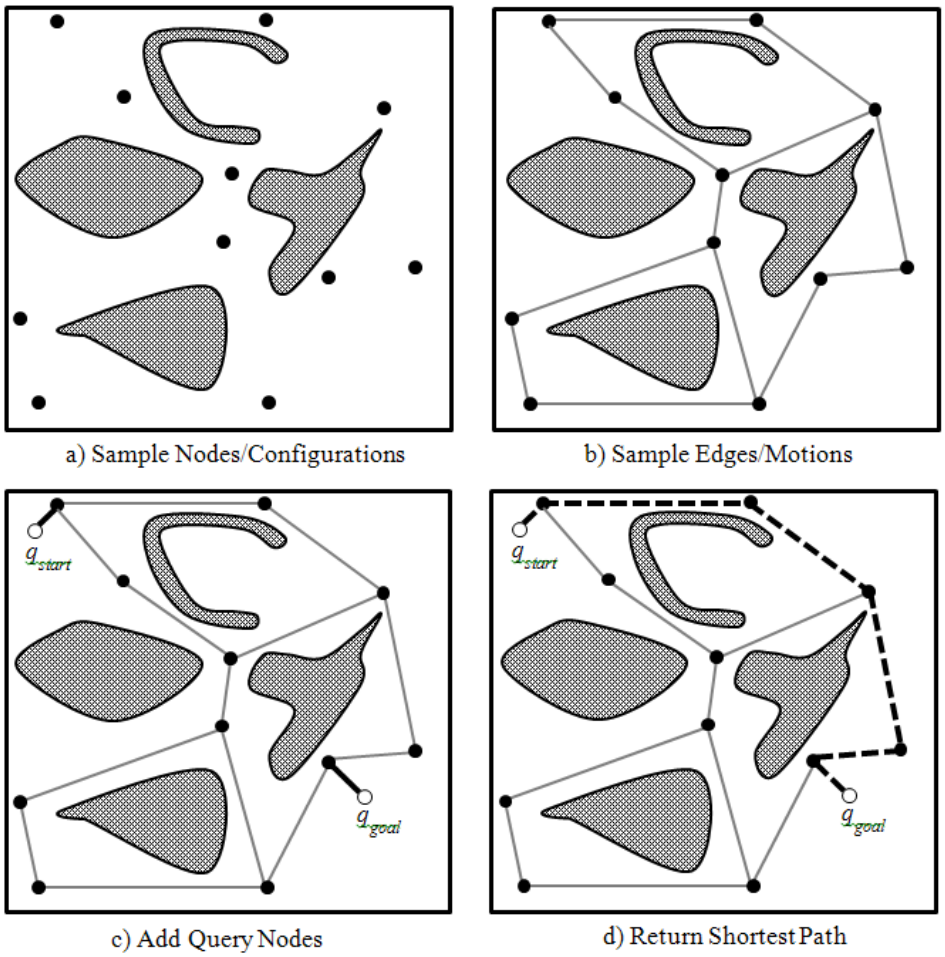


Figure 2-7 The PRM process.

The vast majority of the PRM planner's computation time is spent processing collision of nodes and edges during the construction phase, meaning that the planner is far more suited for use in a multiple query problems. A large amount of time often spent sampling C_{free} to a high resolution and, providing that the surrounding environment remains static, the resulting roadmap can then be used to rapidly solve multiple motion planning queries. The amount of time spent sampling C_{space} will have great effect on the quality of the paths generated, meaning that in some systems, highly optimal paths can be generated.

Extra work on the original PRM algorithm, carried out by Bohlin and Kavraki [50], resulted in a modified PRM planner of particular note. This work addressed two observed drawbacks to the general PRM planner. Firstly, collision-checking processes are by far the most computationally expensive component of the PRM algorithm. For

typical industrial problems, these processes can take up over 90% of the computation time. Secondly, it was observed that only a small fraction of these collision checks are typically carried out on the resulting solution path. Bohlin and Kavraki proposed that by delaying collision processing until the query phase, a PRM variant that utilises far fewer collision checks to solve a given problem would be possible. A new planner based on this approach was developed and termed the Lazy-PRM planner, which can process problems fast enough for use in single query applications. From a high level standpoint, as shown in Figure 2-8, the Lazy-PRM algorithm operates in a similar fashion to its predecessor. The algorithm begins by constructing an initial roadmap, however at this stage no collision processing of nodes and edges is carried out. Rather, all nodes and edges are marked as 'unchecked', allowing the roadmap to be constructed at a rapid pace. Then, repeatedly, the shortest available path through the roadmap queried and sent to the collision checking algorithms for evaluation. If clash is detected on the path, the offending element (node/edge) is immediately removed from the roadmap. If sufficient elements are removed, causing the start and goal configurations to become disconnected, an expansion phase is initiated.

The expansion phase generates a set of additional nodes, which are added to the roadmap in order to reconnect the start and goal configurations. This process continues until a collision-free path is found, or, after a set amount of time it is deemed that no solution is available. By delaying the clash checks until the query phase of the PRM process, redundant sampling of regions that could never provide a solution are removed, drastically saving computation time. This planner proved effective enough for use in single query problems, however it is also easily adapted for multiple queries by storing and augmenting roadmaps from previous problems.

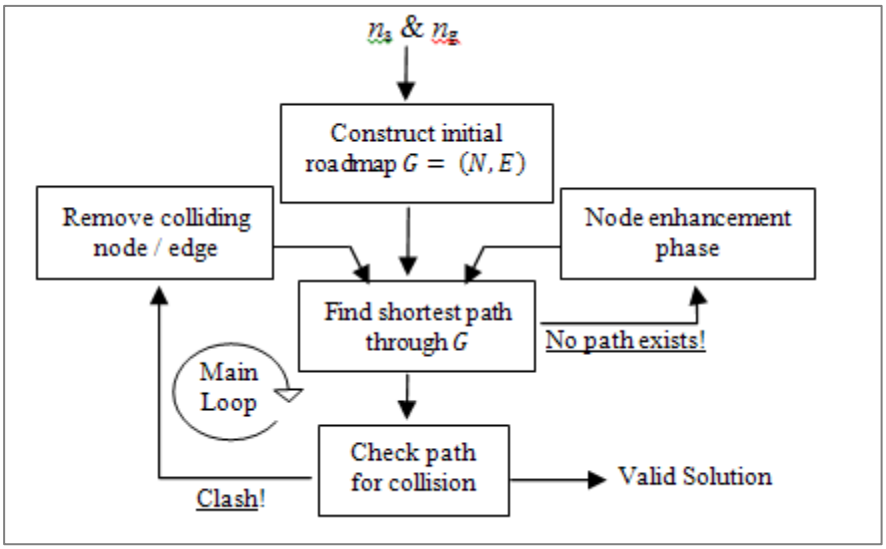


Figure 2-8 Upper level view of LPRM algorithm [50].

2.2.5 Heuristic Sampling for PRM planners

In recent years, research targeting entirely new motion planning algorithms has stagnated somewhat. The focus has now shifted in two different directions: Modifying PRM planners for use in specific applications (see [43, 57, 67]), and developing heuristic techniques that can be used to improve the performance of existing PRM planners. This Section aims to review notable work carried out in the latter category.

Whilst improved heuristic methods have been proposed for many different components of motion planning algorithms (see [48, 68, 69]), a main focal point appears to be on the development of heuristic, non-uniform, sampling methodologies. A particular need for these non-uniform sampling methods can be seen by considering, for example, the 2D motion planning problem presented in Figure 2-9 below. In order to solve the given query, a narrow passage of space must be navigated. A probabilistically complete algorithm will, indeed, eventually be able to reach a solution. However it must be considered that with fully randomised sampling, the probability that any one configuration will be generated inside this narrow passage will be relatively low. In this situation, a lot of time will be expended in order to completely capture the connectivity of this difficult region of C_{space}.

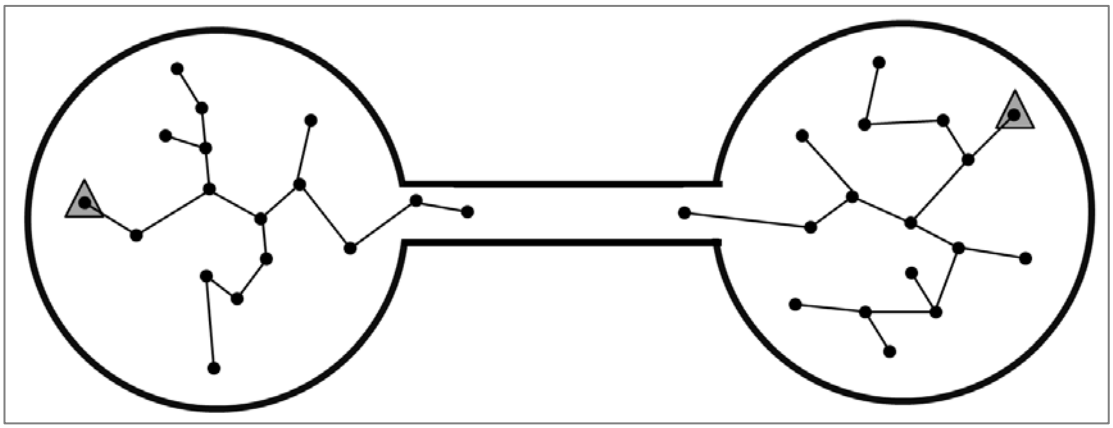


Figure 2-9 An example of the narrow passage problem.

Several methods for the improved sampling of narrow regions of C_{space} have been developed. The general approach is to utilise heuristic methods to gather information about the locations of these difficult regions, allowing configurations to be sampled nearby. The Gaussian sampling method [70], attempts this by generating valid configurations nearby obstacles in the robots C_{space}. Configurations are generated in pairs, within a certain distance of each other, randomly about the robots C_{space}. If one of these configurations is found to be in collision whilst the other is valid, then the valid configuration must be close to an obstacle and is subsequently added to the roadmap. The Bridge Sampling method [71], shown in Figure 2-10, extends this approach further by heuristically generating configurations that lie specifically within narrow passages of C_{space}. Two configurations, say q_1 and q_2, are again generated within some small distance of each other. If both q_1 and q_2 are found to lie in C_{forbid}, the midpoint between them, q_3, is then tested for clash. If q_3 is found to be valid, then it must lie within a narrow region of space and is subsequently added to the roadmap. These two sampling methodologies are relatively simple in operation, which allows them to be easily incorporated into an existing PRM planner. As well, due to the fact that configurations are still generated randomly, the probabilistic completeness of the overall motion planning algorithm is maintained.

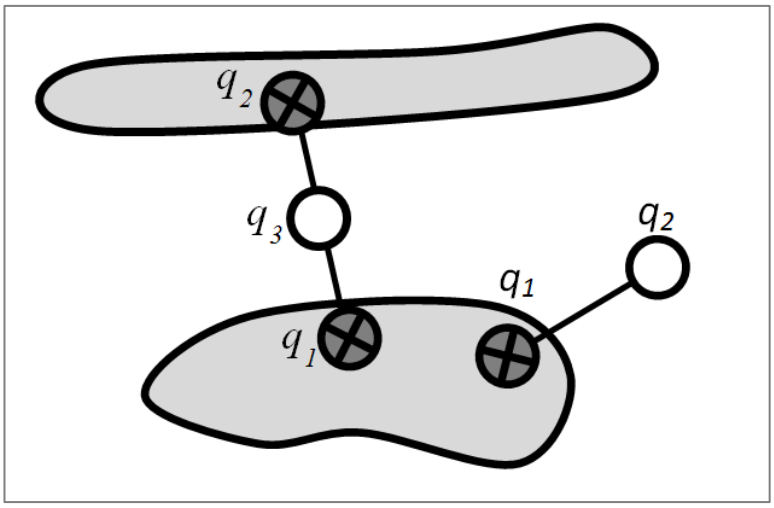


Figure 2-10 Bridge and Gaussian samplers.
The bridge sampler (left) generates configurations in narrow passages. The Gaussian sampler (right) generates configurations close to obstacles.

Obstacle-based PRM (OBPRM) [72] is another approach which attempts to sample robot configurations nearby obstacles. This heuristic approach is incorporated into a modified PRM planner, and operates by pushing invalid robot configurations in random directions until they become valid. This general methodology is extended with the development of the Medial Axis PRM planner [65]. In this approach, both valid and invalid robot configurations are retracted towards the medial axis of free space. This was found to be a practical approach for rigid-body style robots, however much difficulty was encountered when performing the necessary calculations for application in higher degree problems.

Even the original PRM algorithm, outlined in [45], utilises a simple form of heuristic non-uniform sampling. During construction of the initial roadmap, the number of attempts made by the local planner to connect nearby nodes to each other is recorded. The nodes with the highest number of failed connections are then deemed to be in difficult regions, and additional configurations are actively sampled around these nodes. The Lazy PRM planner [50] utilises non-uniform sampling during its enhancement phase, which is called when the roadmap graph becomes disconnected between its start and goal nodes. A set of new nodes are generated and linked into the roadmap in order to re-connect it. A portion of these nodes are generated randomly about C_{space}, so as to maintain the probabilistic completeness of the overall algorithm. The remainder of these nodes are generated about edges that have been invalidated by the local planner, but

also have at least one end point that lies in C_{free}. Edges that satisfy this criterion will surely be within close proximity of an obstacle, so it makes sense to sample nearby.

Another common problem experienced by PRM planners is that as the number of nodes in the roadmap grows, the time taken to process nearest neighbour queries increases significantly. With this in mind, it would be beneficial to keep the number of nodes required to solve a problem as small as possible. One novel approach to achieving this is presented in [73]. It was observed that the nodes sampled in the expansive regions of C_{space} generally do not enhance the connectivity of the roadmap, rendering them redundant to the given problem. In this method, nodes are only added to the roadmap if they cannot be connected to any other node, or if it can be connected to at least two other nodes from different roadmap components. This ensures that nodes are only added to the roadmap when they can offer some form of improvement, or if they can plot a new path that would previously have been impossible. Experiments showed that this approach could cover a given C_{space} using fewer sampled nodes and in a reduced amount of time.

Heuristic sampling has also been incorporated into some tree-based motion planning algorithms. For example, in [60], a cache of valid configurations sampled from previous queries is used to bias the growth of an RRT algorithm. In its standard format, the growth of the RRT alternates between expanding in a random direction, or towards the goal configuration. In this new approach, the alternation is expanded to include a third option of growing the tree towards a node found to be valid in a previous motion planning query. By doing this, the growth of the tree becomes biased towards expanding into regions previously explored. Another approach applied to an RRT algorithm is presented by Burns in [74]. The concepts of weighted utility were used to bias the use of a number of different sampling heuristics related to node selection, expansion direction, expansion distance and tree connection. In this method, utilities relating to node selection, expansion direction, expansion distance and connection attempts are evaluated to control the growth of the expanding tree. As the motion planning process continues, successful aspects are progressively rewarded and become more dominant.

Approaches that combine the strengths of different heuristic sampling methods into one hybrid scheme have also been proposed. Hsu [75] outlined an approach that adaptively

selects which sampling scheme to use, based on the progress in which C_{space} is sampled. Another approach proposed using multiple samplers in series, where the output of one sampler acts as an input to the other sampling schemes used [76].

2.2.6 Summary

Whilst the field of research of motion planning is active and has produced a wide array of different planning algorithms; no single motion planning algorithm has proven to be fundamentally better than any of the others. That is, there is no widely accepted consensus as to whether there exists an algorithm that can act as a 'silver bullet' for the wide array of different motion planning problems in general.

Indeed, certain methodologies have proven fundamentally more successful than others, such as the development of sampling-based approaches which utilise the concepts of probabilistic completeness to guarantee their success. However, even within this category of motion planning algorithm there still exists many different approaches, each tailored for a specific purpose with their own particular strengths and weaknesses. This is due to the fact that these sampling-based algorithms do not operate as one singular entity. Rather, they operate as a synergy of many different algorithmic processes, all working together to solve the motion planning problem. The separate components of a motion planning algorithm, such as graph searching algorithms, neighbour querying, local planning, collision-processing and non-uniform sampling strategies, are under continual development and modification. With each new development, a series of new planning algorithms can be spawned. These new algorithms may not necessarily perform better overall, however they will be well suited for the specific application it was developed for.

The recent flurry in research on non-uniform sampling techniques is further evidence of this. In the majority of cases, these new heuristic methods are developed to improve sampling for a particular scenario. However a 'black-box' approach, which combines all these effective techniques into one algorithm, is yet to be developed.

In summary, it is evident that the variety of motion planning algorithms in existence is almost as wide as the many applications that require a motion planning solution. If an

effective motion planning algorithm is to be developed for a particular robotic system, a deep understanding of the many fundamental approaches to motion planning is equally as important as having the programming and engineering skills to design, develop and implement these algorithms. In addition, the engineer must fully understand their target application before proceeding with the development of a motion planning algorithm. Many factors must be evaluated and understood. For example, does the environments configuration change, and how frequently? Does the robot shape change? Are there specific motion constraints or control factors of the robot that must be considered? Do we require optimal paths to be generated, or simply a path found very rapidly? Can we afford to tune the algorithm to maximise its performance? Or do we want an algorithm with minimal factors that can be tuned? etc.

Chapter 4 of this thesis presents the development of a new, sampling-based, motion planning algorithm designed specifically for the case study presented in Chapter 5. During the development of this algorithm, many of the considerations made above were evaluated; a summary of this process is presented in the introductory sections of Chapter 4.

2.3 OPTIMISATION OF ROBOT PATHS

A significant consideration when approaching any motion planning problem is the quality of the generated path. Sampling-based algorithms generally suffer in this regard due to their reliance on randomised sampling of C_{space}. This randomness often results in a path that skews far away from any local optimum, awkwardly navigating about obstacles with no particular intent. In the case of a robotic manipulator these zig-zag motions can also induce rapid changes in joint trajectories between successive motions, generating large forces in the manipulators structure causing vibration and excessive wear. It is clear that these raw paths require some form of optimisation before they are suitable to use with real world robotic hardware.

Before considering the problem of generating an optimal path, it is important to consider what exactly constitutes optimality with regards to robot motion, which varies considerably depending on the robotic system being used. Typically, an optimisation algorithm will attempt to improve path quality by reducing overall path distance. However in certain applications, other factors must also be considered. Examples include maximising clearance between a given robot and its surrounding environment [77], reducing overshoot [78], controlling orientations [57] or improving the dynamic qualities of non-holonomic motion [79].

The process of optimising robot paths can generally be classified into two approaches: Optimisation embedded in the motion planning process, and optimisation as a post-process to motion planning.

2.3.1 Embedded Optimisation

A popular approach to embedded optimisation involves generating modified roadmap graphs that store extra information about the environment during construction. This is demonstrated by Kim et al. [80], where the definition of an edge cost is expanded to include information relating to robot clearance, visibility and non-holonomic motion constraints. These additional constraints do not affect how the roadmap graph is constructed. Instead, they are used by the graph searching algorithm, during the query

stage of the PRM process, to evaluate and return a more optimal path through the constructed graph.

With roadmap planning algorithms, such as the PRM algorithm, the quality of a returned path is heavily reliant on the number of connections between each node in the roadmap. Consider two roadmap graphs $G = (N, E)$ and $G' = (N', E')$, each consisting of an identical set of nodes, i.e. $N = N'$. If more connections exist in G than G', i.e. $|E| > |E'|$, it can be said that G will have a higher probability of returning a shorter path than G'. However, the added computation required to sample these additional edges comes at a significant cost to the time taken to complete the construction of G. The work done by Nieuwenhuisen and Overmars [81] addresses this trade-off between optimality and computation time by first constructing a roadmap as a tree like structure (with no cycles present), and then selectively adding edges which generate cycles in specific areas. Using their notation, an edge causing a cycle is only added when: $K \cdot d(q, q') < G(q, q')$, Where q and q' are two nodes in the roadmap G; $d(q, q')$ is the corresponding distance between q and q'; and $G(q, q)$ is the shortest distance through G between q and q'. The arbitrary parameter K is used to control the number of cycles added to the resulting graph. A small value of K will add more cycles, whilst a large K will add less. An additional benefit of this method is that it is quite general, and could hence be easily combined with the concepts presented by Kim et al. [80].

Another well documented approach to optimisation involves generating an initial solution, analysing it, and then using this information to refine the result. A simple approach to this is presented by Klasing [82], where the quality of a returned path is evaluated each time a solution is found. If it is not deemed satisfactory, the construction phase of the cell-based PRM planner continues until a certain quality is met. This work, however, does not specifically detail how quality is measured, or the level of quality needed to pass the query. A similar approach is presented by Guernane [83], in which a modified PRM planner is used to generate an initial solution which acts as a guide to restrict the robots C_{space}. The motion planning process is then re-started in this reduced space with the aim of generating a more optimal solution, as depicted graphically in Figure 2-11 below.

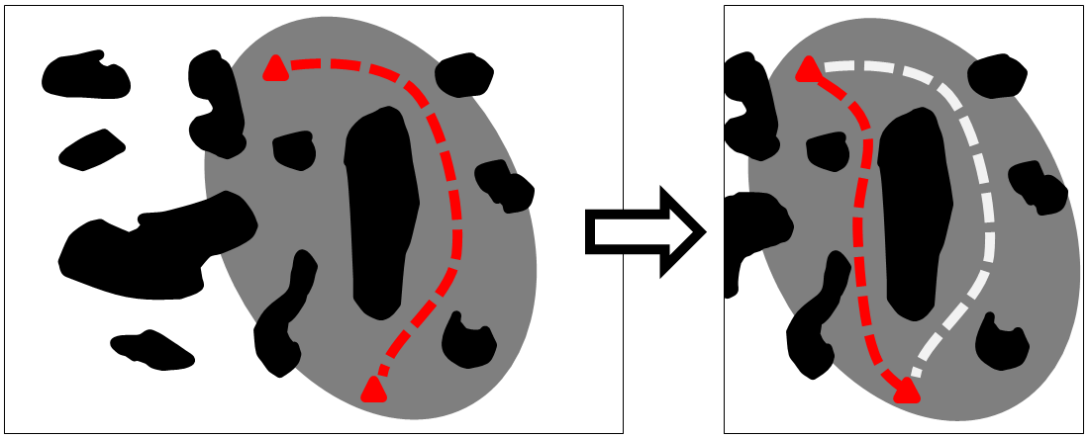


Figure 2-11 Path optimisation by via refinement.
An initial path is used to define reduced region (grey) to refine the motion planning process (figure adapted from [83]).

A similar concept was used for optimisation of paths by Sekhavat [79], where the initial path is used to define a 'tube' of space surrounding it. Inside this tube, the planner is then able to better plan motions with non-holonomic motion constraints to produce a smoother, more optimal solution. Complex mathematical approaches to generating optimal paths have also been a topic of research for embedded optimisation. In separate research, conducted by Ratliff [84] and Lin [85], gradient decent techniques were combined with probabilistic sampling-based methods to guide the growth of a roadmap to produce near-optimal solutions.

2.3.2 Post Process Optimisation

Post process optimisation involves the additional conditioning of a path that has been returned from a motion planning algorithm. A common trend in many documented approaches is to utilise the nodes of the returned path as landmarks for the local planner, which incrementally searches for any valid shortcuts. These approaches are popular due to their simplicity and effectiveness, making them easily adapted into different planners and robotic systems. Consider a collision-free path returned from a planner, $\bar{\pi}$, consisting of π_n nodes: $\bar{\pi} = \{\pi_1, \dots, \pi_n\}$, where each sequential pair of nodes in this path is joined by a valid linear motion. Isto [86] and Kallmann [87] performed optimisation by searching for and removing any redundant nodes from the path $\bar{\pi}$. This is carried out by incrementally checking whether a valid linear motion can be sampled

between any $\pi_i \rightarrow \pi_{i+2}$, for $\pi_i \rightarrow \pi_{i+2}$. If any successful connection between π_i and π_{i+2} is made, then π_{i+1} is deemed redundant and is then removed from the path $\bar{\pi}$, as shown in Algorithm 2-8 below.

Algorithm 2-8 Remove Redundant Nodes Optimisation.

Function: RemoveRedundantNodes (path $\bar{\pi}$)

1: $n \leftarrow$ number of nodes in $\bar{\pi}$
2: **for** i = 1 **to** $(n-2)$
3: **for** j = (i + 2) **to** n
4: **if** $\pi_i \rightarrow \pi_j$ is collision-free **then**
5: $\bar{\pi} = \bar{\pi} \backslash \pi_{i+1}$: *remove* π_{i+1} *from path* $\bar{\pi}$
6: **end if**
7: **end for**
8: end for

Algorithm 2-9 Path segment optimisation.

Function: SegmentShortcuts (path $\bar{\pi}$)

1: **loop**
2: $n \leftarrow$ number of nodes in $\bar{\pi}$
3: i = random integer $1 \leq i \leq n-2$
4: j = random integer $i+2 \leq j \leq n$
5: **if** $\pi_i \rightarrow \pi_j$ is collision-free **then**
6: $\bar{\pi} = \bar{\pi} \backslash (\pi_{i+1} \rightarrow \pi_{j-1})$: *remove bypassed segment from path* $\bar{\pi}$
7: **end if**
8: **continue loop**

This general scheme of incrementally searching a given path for redundant nodes, or shortcuts, is widely utilised in various formats. For example, in the approach outlined in Algorithm 2-9 [40], the local planner attempts to bypass path segments consisting of multiple nodes, rather than bypassing single nodes. In this approach the optimisation algorithm performs a set number of iterations, and path segments are selected at random. Guernane [88] used the mid points of edges, rather than nodes, as milestones to attempt shortcuts with the local planner, and Hsu [89] extended this approach by oversampling a number of additional configurations into the path, creating more potential for valid shortcuts. A graphical summary of some of these generic path shortcut techniques is presented in Figure 2-12.

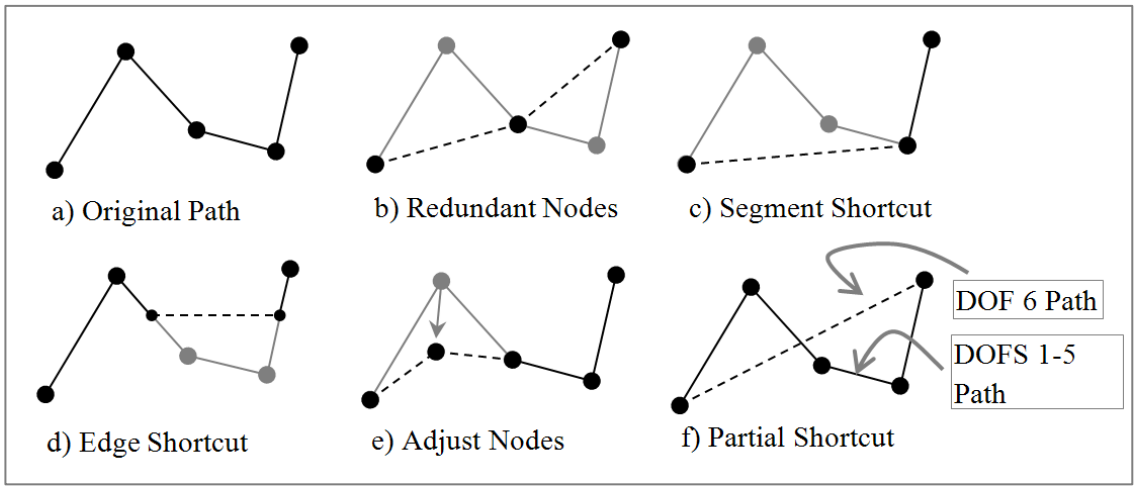


Figure 2-12 Summary of generic path shortcut techniques.

Smoothing the transition between successive motions is another motion parameter that is frequently addressed during post process optimisation. This is done to reduce jerk when transitioning from one motion and another, and is usually required when non-holonomic motion constraints must be considered. In work presented by various authors [88, 90, 91], this is achieved by adjusting the robot path at the transition between robot motions with cubic polynomial splines, an example of which is shown in Figure 2-13. Much like the generic methods described in Figure 2-12, a new path is first proposed, and then tested for validity by a local planning algorithm. If collision occurs on the modified paths, the offending arc's radius is reduced, and testing continues iteratively until the path is found to be in a collision-free state.

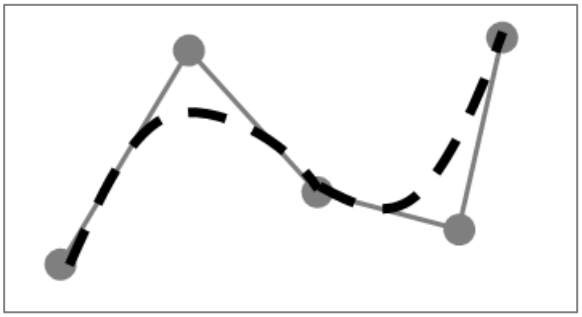


Figure 2-13 Smoothing path segments with cubic polynomial.

Other approaches to optimisation involve improving the clearance a robot shares with its surrounding environment. One approach involves the calculation of the geometric

medial axis of the robots environment. This axis can then serve as a guide for the robot, so that it can navigate various obstacles with the maximum possible clearance.

Both Geraerts [78] and Holleman [92] calculated the medial axis of free space first, and then use it as a guide for retracting a generated path towards, as shown in Figure 2-14. In high dimensional systems, however, computational times required to evaluate the medial axis are reportedly very high [78]. There is also difficulty involved in getting all parts of a higher-DOF robot to align and follow the medial axis, particularly when kinematic and non-holonomic motion constraints must be considered, as documented in [93].

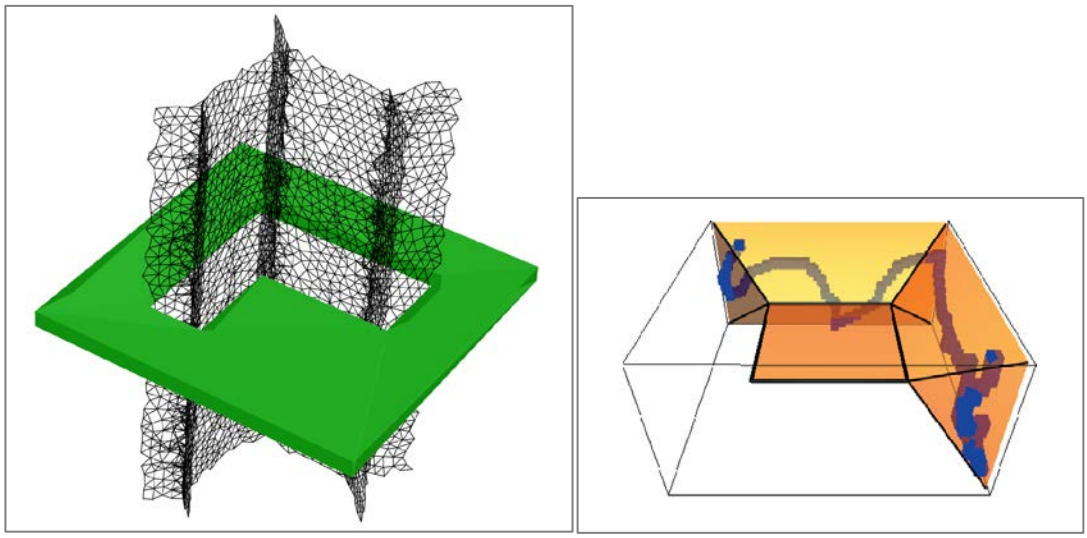


Figure 2-14 Examples of medial axis calculation for generating optimised robot paths. From [92] and [93].

A novel adaptation to many of previously presented generic shortcut methods, called the *partial shortcut* (P-Sc) algorithm, was developed by Geraerts [78]. This approach is similar to the multiple segment shortcut, however it only interpolates a shortcut to one of the robots movable degrees of freedom at a time. A formal description of the method is found in [78], where $\pi[0\ldots1]$ is used to represent the (continuous) path between two randomly selected configurations q and q'. $\pi_i[n]$ denotes the value of the i^{th} DOF at position $0 \leq n \leq 1$. In this method, the path π is replaced by π', which is a new path where all DOFs behave in the same way as π, except for one randomly selected DOF, f, which follows a linear trajectory interpolated between q and q'. This partial shortcut

algorithm is described with pseudocode in Algorithm 2-10, and is represented diagrammatically in Figure 2-12 detail f).

Algorithm 2-10 Partial shortcut pseudocode.

Function: PartialShortcut (Path π)

1: **Loop**
2: q, q' ← two random configurations on the path π
3: $\pi[0\ldots1]$ → the path between q and q'
4: f ← a random degree of freedom
5: **for all** $n \in [0,1]$ **do**
6: **for all** $i \neq f$ **do**
7: $\pi_i'[n] \leftarrow \pi_i[n]$
8: **end for**
9: $\pi_f'[n] \leftarrow (1-n)\pi_f[0] + n\pi_f[1]$
10: **end for**
11: **if** π_f is collision-free **then**
12: $\pi \leftarrow \pi'$
13: **end if**
14: **continue loop**

2.3.3 Summary

The motion planning and optimisation work carried out in this thesis deals exclusively with manipulator style robots. With this in mind, it is important to consider what is required when optimising a path for these types of robots. In literature, the majority of work carried out on optimising manipulator paths utilised one of the generic shortcut techniques depicted in Figure 2-12 (page 59). However it must be noted that a robotic manipulator will typically only need to move a small portion of its joints in order to navigate an arbitrary obstacle. Motion of any of the other manipulator joints is redundant, as they are not required for this navigation. Whilst generic path pruning and shortcut approaches are effective at shortening the overall length of a manipulator's path, a portion of the robots joints will still exhibit redundant motion, even after the optimisation process has been carried out. This is because the shortcuts used are interpolated over all of the manipulators joints simultaneously. The Partial Shortcut method, on the other hand, applies its shortcuts to single manipulator joints at a time.

This approach provides an effective means to reduce or completely eliminate redundant motion of all of the manipulators joints. In doing this, harsh changes in joint trajectories between successive motions are also reduced. The Partial Shortcut algorithms' probabilistic approach and use of roadmap style paths also simplifies its operational details, making it easy to understand and implement into an existing OLP system.

Optimising single robot DOFs at a time, however, requires significant computational effort to carry out. The original P-Sc algorithm requires a large amount of collision checks in order to to reduce the length of a path to a suitable level. This disadvantage becomes more prominent in industrial OLP applications where the large number of obstacles leads to a slower collision processing time, reducing overall usefulness of the partial shortcut approach. To address this particular weakness, the development of a new adaptive partial shortcut algorithm is presented in this body of work. This new algorithmic approach, presented in Section 4.2, is designed to operate more efficiently by reducing the number of collision checks required to optimise a given path.

3 AUTOMATED OFFLINE PROGRAMMING: COMPONENTS, CONCEPTS AND METHODOLOGY.

In this Chapter, the concepts and methodology behind an automated approach to offline programming are presented. This programming methodology is targeted specifically at programming robotic manipulators for industrial applications. Forming the outline of this Chapter is the block diagram presented in Figure 3-1 below. This diagram represents the workflow for the proposed AOLP process, which is adapted from the conventional OLP process block diagram previously presented in Figure 2-2 (page 24). In this Chapter, each process block in Figure 3-1is given its own sub-section, in which details of how automation can be applied and other developmental considerations are presented. The AOLP approach outlined in this Chapter was initially developed for robotic arc welding applications, so as a result many of the examples given are presented in the context of this manufacturing process. However examples of how these methodologies can be extended for use in other manufacturing processes are also presented where necessary.

Before the individual tasks involved in the AOLP process are presented in more detail, we will briefly review the overall approach of the proposed AOLP system.

3.1 AUTOMATED OFFLINE PROGRAMMING OUTLINE

The block diagram in figure 3-1 illustrates that the proposed AOLP approach consists of two key operational phases: The setup (preparation) phase and the robot programming phase. In addition, there is a testing and implementation phase, which involves testing a generated code file on the physical robotic system before it is implemented into full production. Whilst this testing phase is critical to the effective use of robot codes generated by AOLP methods, it is not an active part of the AOLP process itself. The techniques used to perform this phase are standard practice for a robotic operator when testing robot code files, regardless of the method used to generate them. As a result, details about this phase are kept to a minimum in this Chapter. Nevertheless, an example of how this phase is carried out is presented during the case study in Section 5.7.

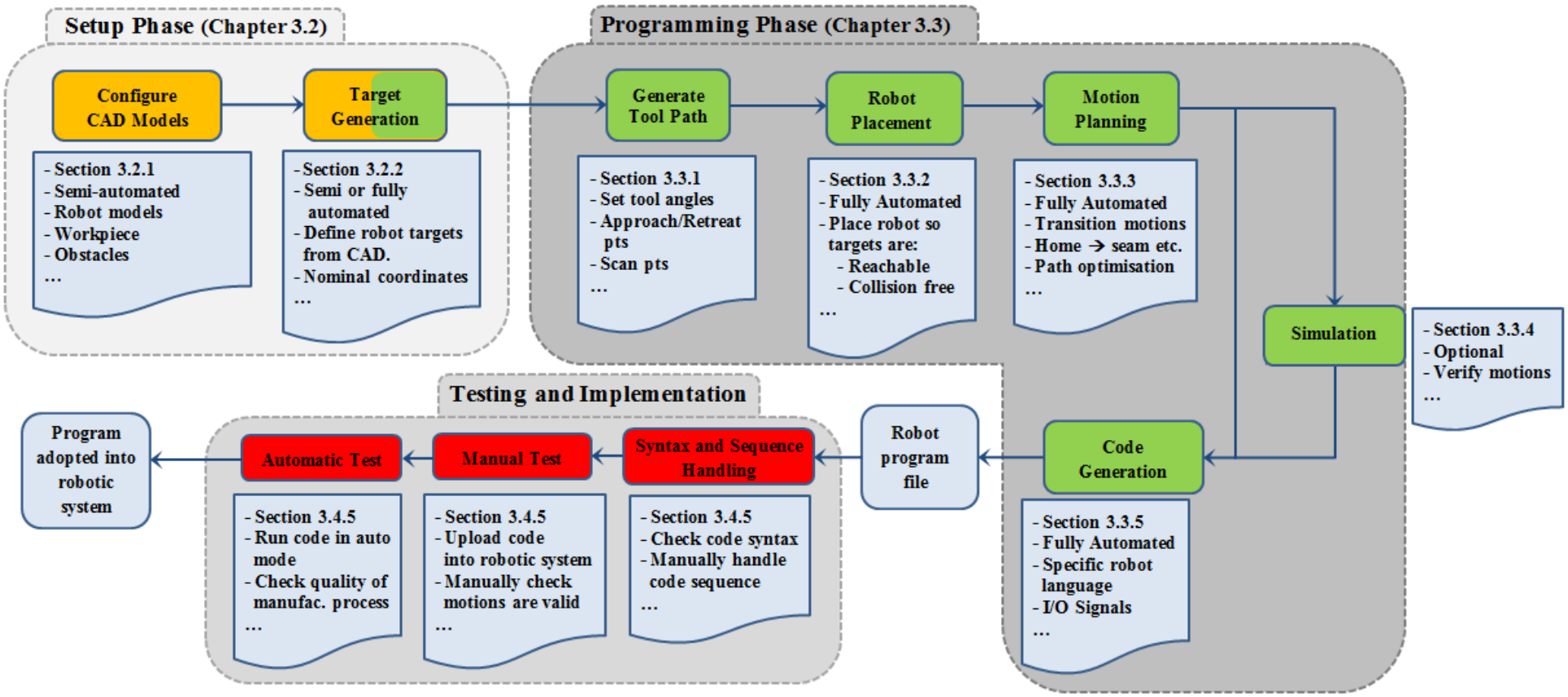


Figure 3-1 The proposed AOLP process.

Green blocks are automated. Orange blocks are semi-automated. Red blocks have no automation (manual process)

The setup phase is carried out over the first two blocks in the AOLP process diagram in Figure 3-1. The first task involves configuring and generating the CAD models used to represent the robotic system in a virtual environment (Section 3.2.1). The second part of the setup phase involves the use of the generated CAD data to define robot targets for the specific manufacturing process (Section 3.2.2).

The data generated during the setup phase provides a platform to carry out the programming phase. The programming phase makes use of a number of automated steps to generate a robot code file used to command the automation system to carry out a given manufacturing process. From a high-level perspective, the programming phase consists of five key steps:

1. Configuring and generating tool paths for the given manufacturing process (Section 3.3.1). This involves converting nominal tool placements, generated during the setup phase, into a continuous tool path that the robot tool needs to follow in order to carry out the manufacturing process.
2. Searching for a location to place the robotic manipulator so that it can successfully carry the tool along the path specified in the previous step (Section 3.3.2).
3. Generating a series of additional transition motions that guide the robot from a home position to the location where the manufacturing process takes place (Section 3.3.3).
4. (Optional) Verifying the generated motions via simulation (Section 3.3.4).
5. Generating a text file that translates the simulated robot motions into a series of commands in a robot programming language. This code file can then be uploaded into the physical robotic automation system for use (Section 3.3.5).

3.2 SETUP PHASE

As shown in Figure 3-1, the setup phase must be completed before the programming phase can commence. The setup phase takes longer to carry out than the programming phase, however the effect of this is minimised as this phase only needs to be performed

once. In this Section 3.2, the concepts and approach to automating the two key tasks of the setup phase are presented in detail.

3.2.1 Configure CAD models

The ability to represent physical components in a virtual-reality environment makes CAD a vital component of OLP. In OLP applications, CAD is used primarily for:

- *Visualisation*. A CAD representation of the automation system allows the user to inspect the systems physical set up and view simulated robot motions.
- *Collision processing*. The geometric CAD data of a robot and its surrounding environment is used in conjunction with collision processing algorithms to plan and generate collision-free robot motions.

Today, almost all products are designed with the use of CAD software packages. This drastically simplifies the procurement of CAD models to represent the physical components of an industrial automation system. For example, on the ABB website one can freely download the CAD models of their entire robotic product range. The same goes for CAD models of items such as robotic tooling. CAD models of Fronius weld torches are available from their website, and Destaco provides a range of different CAD models of gripping end effectors used in pick and place operations. Even generic workshop components such as machine-workbenches and PLC cabinets can be downloaded from the Demmeler and Mitsubishi websites respectively. Items that are not readily available for download in the online domain can typically be procured via:

- Direct request to the item's manufacturer or industry expert.
- Represented with a similar model from a competitor's CAD library.
- A mock-up representation of the item can be manually modelled with CAD design software such as Auto-CAD, DELMIA or Pro-Engineer.

Due to many end-uses and often complex nature of CAD, a vast array different CAD formats exist, such as the popular IGES, STL and STEP variants.

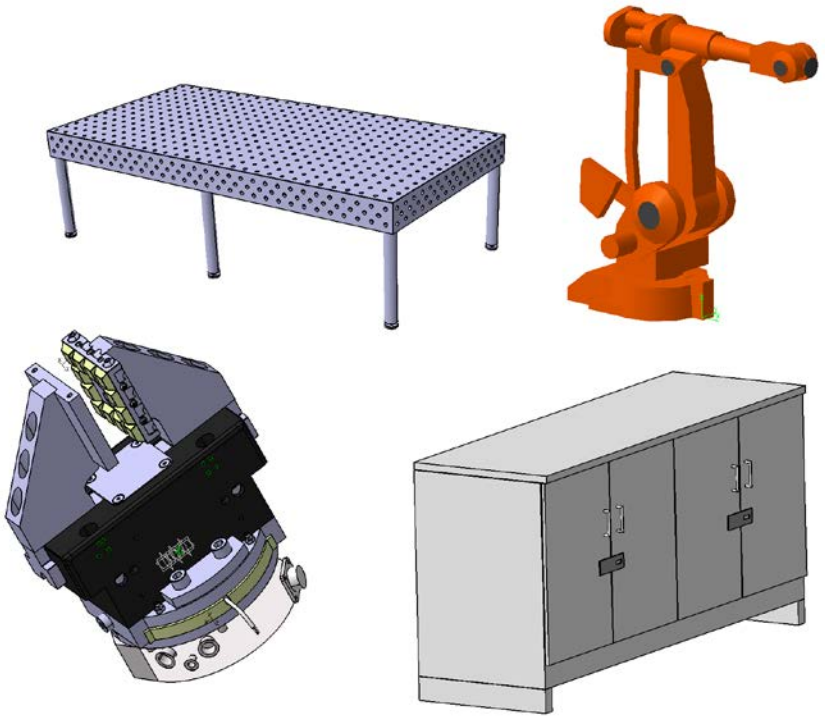


Figure 3-2 Example CAD models of typical workcell items.
Individual items not to scale

For OLP applications, CAD models are also used for collision processing purposes. The proposed AOLP systems collision processing algorithms will use the geometric CAD data of a robot, it's tooling and the surrounding environment to determine whether any of their boundaries are intersecting. These algorithms are used to ensure that motions generated in the virtual environment will not cause collision when used in the corresponding real world robotic system. In order to carry this out, the various objects and devices that make up the robotic workcell must be modelled in some format that can be processed by geometry-based algorithms.

Collision processing between the different elements of the robotic system is, by far, the most time consuming algorithmic component of the AOLP process. Typically, these calculations are performed hundreds of thousands of times for a single run of the programming phase. In our application, a rapid calculation speed is deemed far more important than spatial exactness and graphical visualisation (a similar approach to the work presented in [94]), so simplified bounding volume CAD representations of the robot and its surrounding environment are used. These bounding volume CAD

representations feature far fewer geometric elements than their full detail counterparts, which allows for the rapid processing of collision.

Over the next two subsections the use of this simplified bounding volume approach will be further justified. In addition, the methods used to generate these simplified CAD models for the robots (Section 3.2.1.1) and the surrounding environment (Section 3.2.1.2) are also presented.

3.2.1.1 *Robot and Tooling Representation*

To represent the robot and its associated tooling, sphere-bounding models are used, as shown in Figure 3-3 and Figure 3-4. To generate these models, spheres of varying diameters are arranged in a manner that captures the full geometric detail of the robot and tool as best as possible. This method of CAD representation allows for rapid distance computation due to the resultant simplification in overall geometry, and the reduced number of individual graphical elements processed in each collision test. An example of this is shown in Figure 3-4, the sphere-bound model of the Kawasaki ZX3000 manipulator is represented with 16 spheres, whereas the full detail STEP model is comprised thousands of individual triangle shaped elements.

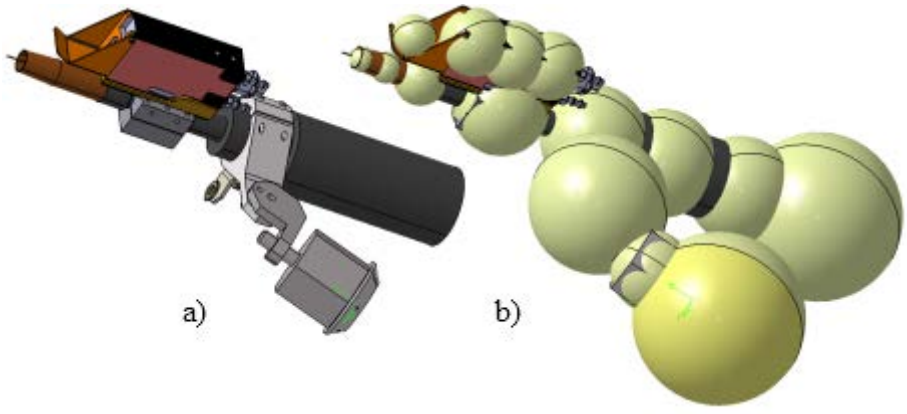
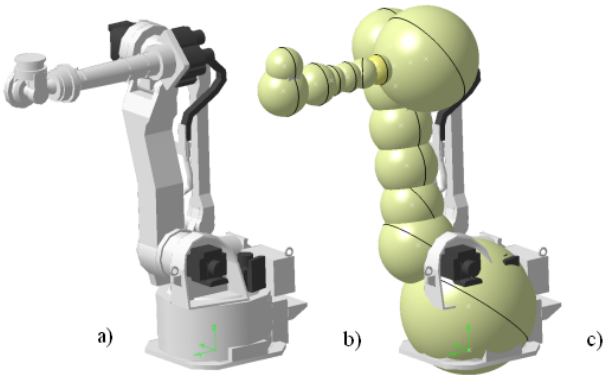


Figure 3-3 a) Full CAD representation of weld torch. b) The corresponding bounding-sphere representation.



	A	B	C	D	E	F
1	Spheres attached on the robot for clash check					
2	# Spheres:	18	X	Y	Z	Radius
3	1	Joint1	0	0	300	450
4	1	Joint2	150	288	740	260
5	1	Joint3	400	288	2040	250
6	1	Joint4	0	288	2040	350
7	1	Joint5	0	1188	2040	140
8	1	Joint6	21	1438	2040	170
9	1	T0	0	1438	2212	125
10	0	Link1	interpolated			0
11	4	Link2	interpolated			210
12	0	Link3	interpolated			0
13	6	Link4	interpolated			100
14	0	Link5	interpolated			
15	0	Link6	interpolated			
16	1	wirefeed	interpolated			200

Figure 3-4 a) Full CAD representation of Kawasaki ZX300. b) The corresponding bounding-sphere interpretation. c) Sphere locations defined in Excel

What is required for the AOLP system proposed in this work is an efficient means to generate these sphere-bounding CAD representations, and also a method to check whether they are configured in a way that suitably represents their full detail CAD counterpart. This is achieved through a combination of an Excel spreadsheet and the DELMIA CAD software suite as follows:

- The location and radius of the spheres used to define the robot/tool model are entered into an excel spreadsheet by the user, as shown in Figure 3-4 above.
- Once done, a Visual Basic script is called from the DELMIA CAD design software suite that reads this information from Excel and then renders the sphere data accordingly. At the same time, a full detail STEP model is superimposed with the generated sphere-bound model, as shown in Figure 3-3 and Figure 3-4, so that the user is able to check the suitability of the sphere model. If modifications are required, the user can then update the Excel spreadsheet and rapidly re-load this data to re-check its suitability.

The visual basic scripts used to automate this procedure interface directly with drawing and rendering commands of the DELMIA software suite, and are called from the graphical user interface (GUI) presented in Section 5.4.1. Further details of *how* a user generates these sphere-bounding robot models are presented in Section 5.5.1.

The drawback to this sphere bounding approach is that these models do not represent the real world component with a high degree of exactness. However the fact that the boundaries of the sphere model typically extend beyond the boundary of the full detail

version ensures that if the sphere model passes a certain collision test, then the full detail model will pass the same test as well. This avoids the situation where a series of collision-free motions generated with sphere models will result in collision when the same motions are used in the real world environment.

3.2.1.2 *Environment and Workpiece Models*

Bounding volumes are also used to represent objects in the robots surrounding environment. However, the variety of volumes used to define these objects is extended to include spheres, cylinders, rectangular prisms and plates. This approach is used for the same reasons as the sphere models are used for representing the robot and tooling: By simplifying the geometry that makes up the workcell environment, collision processing can be performed far more rapidly.

Whilst the generation of these models is also automated via the use of Excel, Visual Basic and DELMIA software, there are fundamental differences to how this data is generated. For the robot model the data is generated manually, and DELMIA is then used to render the resultant model for visual inspection. For environmental objects, on the other hand, full detail models are loaded into the DELMIA interface and a Visual Basic script is used to automate the extraction of the required simplified geometry data.

A Visual Basic script was developed that allows the user to interact with DELMIA's CAD environment in order to efficiently gather this data in an interactive, semi-automatic manner. The full detail CAD representation of the item is loaded into DELMIA, and a Visual Basic script is initiated from the GUI presented in Section 5.4.1. The user is then prompted to click on the plate they wish to define, and once done the plate is separated from the rest of the model. The user is then prompted to click on and select the vertexes that make up one of the place's faces, as well as to click on an edge that represents the plate's thickness. This data is collated by the Visual Basic script and is then stored in the excel spreadsheet for later use by the AOLP system.

Whilst this general approach has negative aspects - the inability to model complex curves, for example - this method achieves significant benefit, as a drastically reduced dataset is used to explicitly define the plate's structure, which allows for the rapid algorithmic processing of collision.

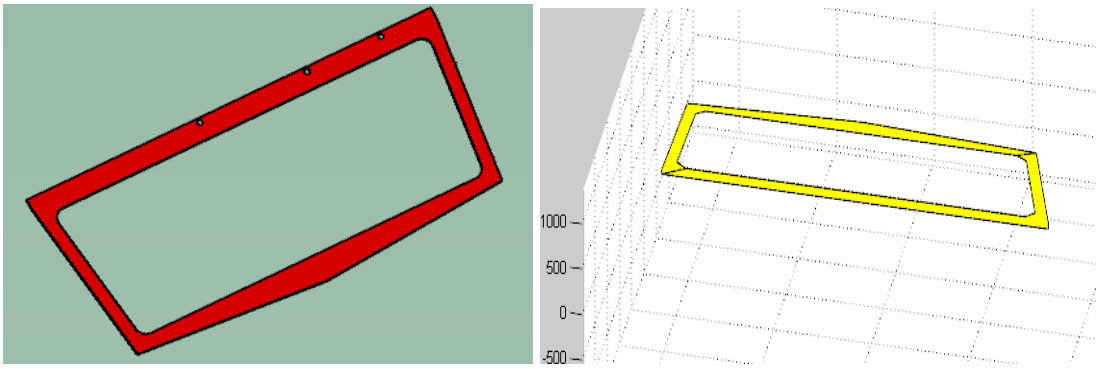


Figure 3-5 The full detail CAD representation of a plate (left), and its corresponding bounding volume representation (right).

3.2.2 Target Generation

In an OLP environment, robot targets are used to define the location of key process points for the manufacturing task that is being carried out. These targets are then used to position the robot and align its tooling so that the given manufacturing task can be performed in a suitable manner.

The method of defining robot targets in the proposed AOLP system is shown in Figure 3-6 below. The robot target itself is comprised of a local coordinate frame featuring three local $\bar{x}$, $\bar{y}$ and $\bar{z}$ axes. The location of the target frame is specified relative to the global coordinate frame with 3D Cartesian coordinates, and the targets orientation is denoted by Euler rotation angles about each of its three local axes. The robot can then use this information to align its own tool centre point (TCP) coordinate frame with the robot target frame, as depicted in Figure 3-6. Algorithmic details relating to the representation of robot targets is presented in further detail in Appendix Section 7.3.1 (page 188).

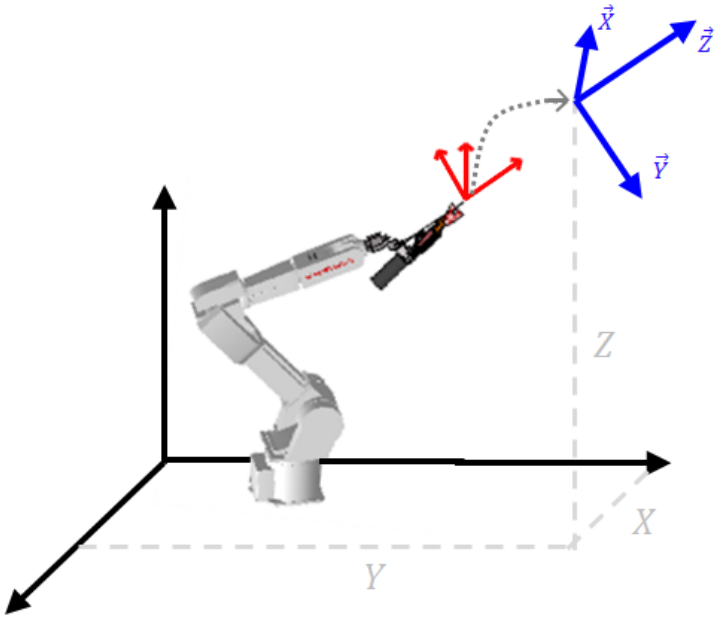


Figure 3-6 A manipulator using a robot target frame to align its tool frame.
Legend: Global reference frame (black). Robot's TCP frame (red). Robot target frame (blue)

The goal of this target generation step is to attach these robot targets onto the key manufacturing process points of the items that are being fabricated. These targets can then be used to properly position the robot and it's tooling in a way that allows the specific manufacturing task to be carried out in an acceptable manner, as shown in Figure 3-7 below.

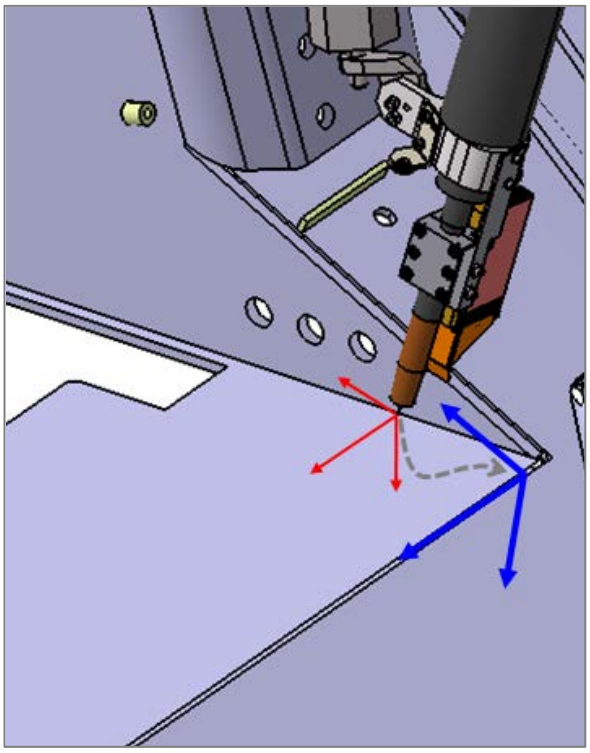


Figure 3-7 Robot aligning its weld torch with a target denoting the start of a seam
Legend: Robot TCP frame (red). Robot target frame (blue)

In currently available OLP software packages, such as DELMIA, targets are generated manually through a few operations in the software's graphic interface. Target generation workbenches assist in this operation, providing a variety of commands that assist the user to attach these tags onto CAD models and to orientate them to properly reflect the required tool positioning for the manufacturing process being performed.

This task, however, is time consuming and difficult to carry out, as optimal tool orientations for a given manufacturing process are difficult to properly determine in the software's user interface. For a complex component that features hundreds of manufacturing process points, this approach becomes problematic, as a significant amount of time is required to properly position and align the all of these targets. To improve the speed in which these targets can be generated, an automated approach to performing this task is proposed.

3.2.2.1 *Automated Target Definition for Weld Seams*

When manually generating and orientating robot targets in a typical OLP environment, the users aim is to attach, or 'snap', these targets onto geometric features of the CAD model that represent manufacturing process points. For the AOLP system, this task is automated with the use of geometric-based algorithms that analyse the CAD model of a workpiece in order to locate where these process points are. This information is then used to automatically generate robot targets for the given manufacturing process.

The target generation algorithms operate by exploiting the fact that a weld seam will always occur at the intersection of two plates. Each surface of the work object is analysed, and then robot targets representing weld seams are generated at each intersection. Algorithmic processes then utilise surface data of the intersecting plates to correctly orientate these targets, so that they represent suitable weld torch orientations for a typical welding process. The algorithmic details behind this approach are presented in Appendix Section 7.3.5.

Whilst this approach is fully automated, it was found not to be suitable for use in some applications where, for example, a large amount of plates in a complex arrangement are featured in the design of the workpiece. This high number of plates, combined with their complex arrangement results in a very large number individual weld seams being defined, the vast majority of which are not actual seams to be welded by the robotic system. An interactive, semi-automated, approach to the generation of robot targets was developed for use in these situations. The approach utilises a human operator's cognitive ability, in conjunction with automated extraction of geometric data, to define weld seams in an efficient and rapid manner.

This semi-automatic method of defining weld seams is similar in nature to the generation of surface data for clash models described above in Section 3.2.1.2. The user carries this task out from a Visual Basic GUI, which interfaces with DELMIA's CAD commands. The user is prompted to manually select an edge that represents the weld seam they wish to define, and to also select the faces that are incident to this edge. This geometric data is then used by the same algorithmic processes as the fully automated approach to generate and orientate robot tags representing the seam. The collated data is

then pasted into the Excel spreadsheet for later use with AOLP. Further details of how this process is carried out are presented in Section 5.5.3.

This semi-automated approach reduces the amount of data to analyse by prompting the user to select the edge they wish to define as a seam. This cuts out the need for geometric algorithms to search through all of the edges for potential seams. The approach also provides the programmer with a far more effective interface to generate targets than the standard methods available in DELMIA. This is due to the fact that the fully manual methods featured in DELMIA require that the user first add two targets to define the seam, and then they have to orientate them correctly in a separate process. The semi-automated approach, however, already assumes you are searching for an edge to define as a seam. Once the user selects the edge, tags are added and orientated automatically, an approach shown to significantly reduce the number of individual operations the user has to perform in order to define a particular seam. Tests indicate that the time taken to define each seam is reduced by up to five times when using the semi-automated approach over standard methods currently available in DELMIA.

The general methodology presented above can be easily expanded to incorporate other manufacturing processes. For a robotic pick-and-place operation, for example, these geometric algorithms could analyse the entire workcell and automatically identify and locate the items that are to me grasped by the robotic device. Then, the orientation of the item can be analysed in order to properly position a robot target above it in a manner that allows the gripper to properly grasp it.

3.3 PROGRAMMING PHASE

Once the setup phase presented in Section 3.2 is complete, the programming phase can commence. The data generated during the setup phase is used as a platform to carry out the programming phase. As presented in the introduction to this Chapter, the programming phase is carried out over 5 distinct steps, each presented individually over the remainder of this Chapter.

3.3.1 Tool Path Generation

The first step of the programming phase uses the robot targets generated in step two of the setup phase (Section 3.2.2) to generate a continuous tool path that the robot must use in order to carry out the given manufacturing task. There are two considerations that must be made when generating these paths:

- The specified path must feature tool orientations that are suitable for the given manufacturing process. For example, correct tool placement is critical to generating a high quality, defect-free, weld seam. The generated path must follow these optimal parameters as closely as possible
- The generated path must not introduce collision between the tool and the work piece. In instances where space is tight, the optimal path may in fact collide with other elements of the work piece. In these situations, a trade-off between optimal tool path geometry and collision must be made.

In currently available OLP software, such as DELMIA, this process is performed manually through various commands that allow the user to manipulate the location and orientation of robot targets. However, the generation of a suitable path hinges on the expertise of the user, who must evaluate several important criteria relating to the manufacturing process being performed. For example, in typical gas metal arc welding (GMAW) processes, the relative orientations between the weld torch and the plates that make up the weld seam are critical in ensuring that the weld will be of suitable quality. In an ideal situation, there is a preferred push angle for the torch, however this angle can be varied to a certain extent without significantly sacrificing the quality of the generated weld. Due to the symmetric shape of the torch, it is free to rotate about its own Z-axis during the weld process. Whilst this freedom to alter the torch orientation provides a means to allow the torch to access corner welds, it also complicates the torch placement problem, as it allows for a near infinite number of torch placement solutions for a given weld seam.

In the AOLP system, the generation of a valid tool path is done via algorithmic processes that automatically generate a large number of potential tool paths, and then removes the ones deemed unsuitable for use. The overall approach is presented via

pseudocode in Algorithm 3-1. During the initial setup, a range of allowable tool orientations that result in defect free welds is specified by the user. If $\hat{n}$ orientations for a weld seam's starting point, and $\hat{m}$ orientations for the seam's end point are specified, the tool path generation algorithms take this data and generate a total of $\hat{n} \times \hat{m}$ unique tool paths. These paths are passed to a collision processing algorithm, which tests whether the tool can traverse along each of these paths without causing collision, as shown in Figure 3-8. The paths that no not pass this test (due to collision between the torch and workpiece) are discarded, whilst the ones that pass are added to an array of valid tool paths $\bar{\pi}$. Once all paths have been tested, this array $\bar{\pi}$ is then exported for use in the next step of the AOLP process.

Algorithm 3-1 Tool path generation for a weld seam.

Function: Tool_Path_Generation $(\hat{s}, \hat{e})$

input: $(\hat{s}, \hat{e})$ – arrays of suitable tool orientations for start and end of seam, respectively.
output: $\bar{\pi}$ – array of valid tool paths

1: $\hat{n} \leftarrow$ number of tool orientations in $\hat{s}$
2: $\hat{m} \leftarrow$ number of tool orientations in $\hat{e}$
3: **for** i = 1 **to** $\hat{n}$
3: **for** j = 1 **to** $\hat{m}$
4: $\beta \leftarrow$ tool path interpolated between $\hat{n}(i)$ and $\hat{m}(j)$
5: **if** β is *collision-free* **then**
8: *add* β to $\bar{\pi}$
9: **end if**
10: **next for**
10: **next for**
11: *export* $\bar{\pi}$ to user

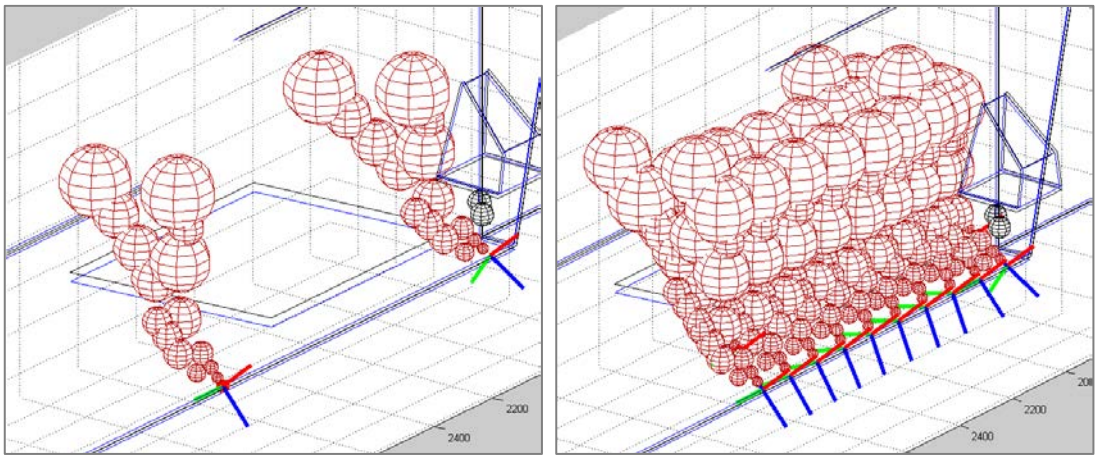


Figure 3-8 Start/end targets of a seam (left) are used to generate a valid, collision-free, tool path (right).

Whilst this step typically involves processing collision for hundreds of different tool paths, the overall processing time is low. This is due to the fact that collision processing carried out during this step is performed with the tool model only; the robot does not need to be considered at this stage. In addition, CAD models of items in the workcell located a significant distance from the seam are also disregarded during this processing of collision, which further eases the computational load.

For other manufacturing processes, such as a pick and place operation, a similar methodology for the generation of valid tool paths can be implemented. For this type of operation, the nominal target data provided from the setup phase can be used to identify the location of the item that is to be grasped. In this step, this same algorithmic approach could be used to generate a tool path that ensures the gripper can approach the item from a suitable distance without causing collision. A safe retreat motion with the tool and its grasped item can also be planned at this stage. The generated path is then passed to the next step of the AOLP process in the same manner as the tool path for a weld seam.

3.3.2 Robot Placement

The way in which the robot placement problem is addressed is dependant setup of the robotic system used. In some cases, the robotic manipulator that carries out the manufacturing process is mounted on some form of transport, such as a linear rail, that

gives the manipulator greater access to regions of the workcell. In other cases, the robot manipulator's base is fixed, and the workpiece is mounted on some kind of positioner, such as a linear conveyor, rotating table positioner, or a more complex positioning device featuring multiple DOFs.

Regardless of the robotic setup used, the robot placement problem relates to correctly positioning the robot and workpiece relative to one another, so that the manufacturing process can be performed.

3.3.2.1 *Positioning a robot with a movable base*

A robot setup with a movable base is commonly encountered in systems involved with small batch production. The robot base is attached to some form of transport, which can relocate the manipulator to different areas of the workcell. A number of different types of transport mechanism are commonly used, which can feature single or multiple DOFs. For example, a linear rail is able to re-position a manipulator along its one axis of motion, whereas a mobile platform will have the ability to reposition robots in 2-DOF. In large-scale shipbuilding operations, it is not uncommon to see welding robots mounted on overhead gantries that can position the robot in the 3-axes of Cartesian space.

Regardless of the number of DOFs featured in the transport mechanism, the general approach to solving the robot placement problem with this style of setup remains the same: We must find a location to position the robot so that it can carry its tooling along a specified path (generated during the previous step of the AOLP process) without infringing upon its kinematic parameters (joint limits, singularity) or causing collision between the robot and workpiece. The mobility provided by the robots transport mechanism simply defines the region of space to search when attempting to properly position the robot.

In a generic OLP software package, such as DELMIA, this task is typically performed manually through trial and error. The user positions the robot with an initial estimate of a suitable location, and then views a simulation of the robot attempting to perform the pre-defined tool path. If this initial position is deemed unsuitable due to the infringement of kinematic motion constraints, or the occurrence of collision between the

robot and workpiece, the user can then manually modify the robot's positioning accordingly. This process is then repeated until the user is satisfied with the robot's final position. This manual nature of this task is troublesome due to the fact that it is difficult to know *how* one should re-position a robot to avoid a particular collision or kinematic infringement. The difficulty is increased when the transport mechanism features multiple DOFs. However the trial and error approach used to solve this type of problem makes it an ideal candidate for automation.

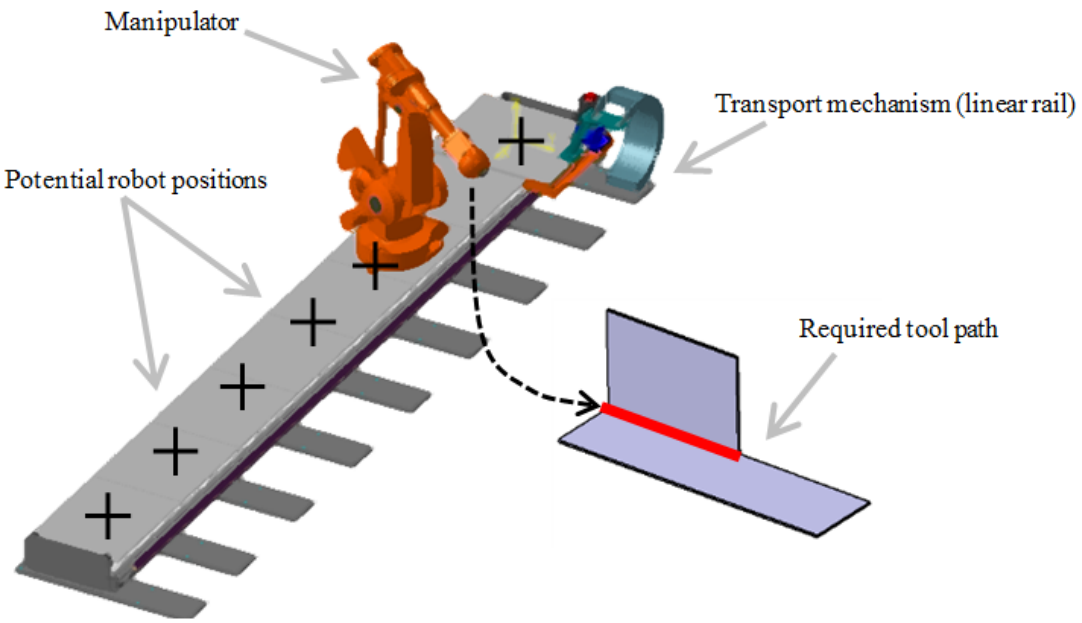


Figure 3-9 The robot placement problem.
Each potential robot location is tested to see if the robot can carry out the required tool path.

In the AOLP system, robot placement is performed with a combination of algorithms that handle robot positioning, kinematics and collision processing. The automated approach is described via pseudocode in Algorithm 3-2 below.

Algorithm 3-2 Robot placement process

Function: Robot_Placement (π, n)

input: π – input tool path. n – number of transport locations

output: β – array of suitable robot locations

```
1:  β ← generate array of n transport locations, distributed evenly.
2:  for i = 1 to n
3:      move robot base to β(i)
4:      if π is kinematically valid then
5:          if π is not collision-free then
8:              remove β(i) from β
9:          end if
10:     else
11:         remove β(i) from β
12:     end if
14: next for
15: export β to user
```

In line 1, an array of n predetermined base locations for the robotic manipulator, interpolated evenly over the transport mechanisms work envelope, is generated (as shown above in Figure 3-9). The manipulator's location is moved to each of these generated base positions and, at each position, two tests are performed. A kinematic evaluation is performed first[2]: From a given base position $\beta(i)$, inverse kinematic algorithms are used to test whether the manipulator can traverse the entire tool path π without exceeding its joint limits, or encountering singularity. If this test is passed, the path π is deemed *kinematically valid*, and the next test is initiated. The second test performed is a collision test: From the same base position $\beta(i)$, collision processing algorithms are used to monitor whether collision occurs between the robot and its surrounding environment as the manipulator traverses the path π. If either of these two tests fail, then the base position $\beta(i)$ is removed from the array β, and the testing procedure continues. After all n base positions have been tested, the base positions that remain in β will allow the manipulator to traverse the tool path π in a kinematically valid and collision-free manner. This array β is then transferred to the next step of the AOLP process.

[2] Kinematic testing is performed first due to its rapid computation speed. This avoids expensive collision processing being performed on tool paths that are not kinematically feasible.

This two-phase approach, where discreet base positions are first generated and then tested with the manipulator, dramatically reduces the complexity of the overall problem. By doing this we avoid the situation where one optimal result must be selected from a possibly infinite number of potential solutions.

In the inverse situation - where the robotic manipulator is the element that is held in one fixed location and it is the workpiece that is to be positioned - the same general methodology as the one described above can be used. To do this, the same kinematic and position algorithms can be used to automate the process. The only distinct difference is that instead of generating an array of potential base positions for the manipulator, we now generate potential locations for the workpiece to be positioned at. Care must be taken to ensure that the generated workpiece positions are kinematically feasible for the workpiece positioner.

3.3.3 Motion Planning and Path Optimisation

At this stage of the programming process, a tool path for the specific manufacturing process has been generated, and a location to position the robotic manipulator so that it can follow this path in a valid manner has also been found. The programming process now turns to planning a series of transition motions that will take the robotic manipulator from a pre-defined home position to the location where the tool path is situated.

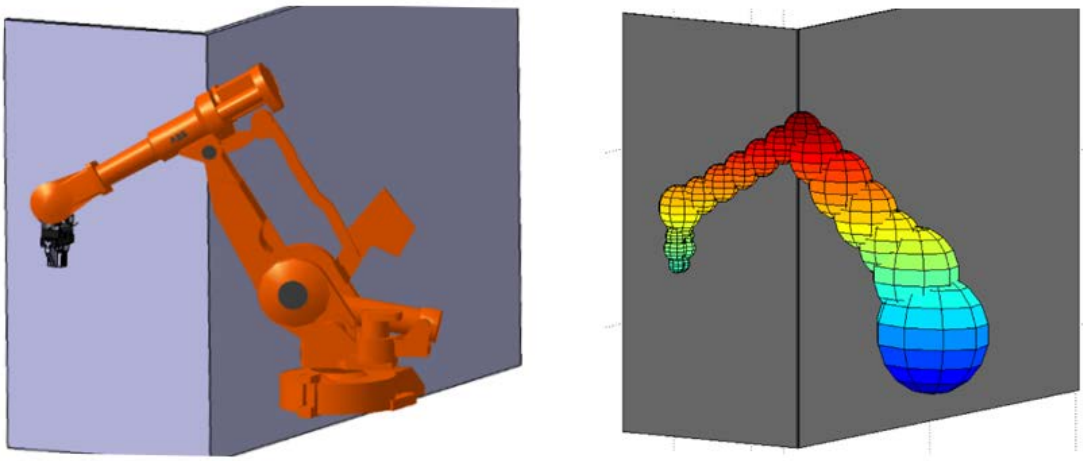
Performing this task in currently available OLP software, such as DELMIA, is relatively simple. DELMIA has well configured menus that allow a user to efficiently plan, generate and modify robot motions; providing an efficient means for a user to manually generate these transition motions. Since the AOLP approach does not have these faculties available, an automated motion planning algorithm was developed to carry out this task. However, developing an algorithmic approach to solving these motion planning problems is no trivial task, particularly for robotic manipulators which feature a high number of movable DOFs and typically operate in cluttered or complex 3D environments.

As discussed in the literature review (Section 2.2), sampling-based motion planning algorithms have proven effective at solving these more difficult motion planning problems. For use in the AOLP system, a new sampling-based motion planning algorithm, developed specifically for use in robotic manufacturing applications, was conceptualised and developed. The background research, development and testing of this motion planning algorithm is presented in Chapter 4. Also presented in Chapter 4 is the development process for an optimisation algorithm used to improve the quality if paths generated by the motion planning algorithm.

3.3.4 Simulation

For the AOLP approach presented in this work, two simulation platforms were developed. Initially, simulation was performed in the DELMIA software environment, which allows for simulation with the full detail CAD representation of all robots and objects. However, to avoid the reliance on third-party software packages, another simulation interface developed entirely in MATLAB was also developed for situations where DELMIA may not be readily available.

As shown in Figure 3-10, the benefit of using the DELMIA simulation interface is the superior rendering and visualisation control made available to the user. Development of this interface was relatively straight forward, as DELMIA is already well configured for simulation purposes. A Visual Basic script was created that can read a generated robot code file into the DELMIA environment. The robot motions contained in the robot code file are then configured with DELMIAs inbuilt simulation interface for use. The user is able to use DELMIA's inbuilt simulation commands to view, re-play and inspect the generated motions. The Visual Basic code that controls this simulation is called from the Visual Basic GUI shown in Figure 5-10 (page 130).



a) b)

Figure 3-10 Comparison of simulation methods
a) Full detail DELMIA simulation. b) MATLAB counterpart with simplified CAD

The design goal of the MATLAB simulation interface was to provide simulation capability that could be incorporated into the AOLP system without having to rely on any third-party software. The drawback of the MATLAB simulation, however, is that all rendering is done with the simplified bounding volume CAD models generated in Section 3.2.1. A distinct benefit of the MATLAB simulation, however, is that it is far more immersed in the programming process, allowing the user to either display the simulation during programming or after a program is generated. As robot paths are generated, they are uploaded automatically into the simulation interface for immediate use, which gives the user the capability to check the validity/suitability of certain robot motions before the entire robot code is generated. For example, simulation can be used to check that the positioning of the weld robot, generated during Section 3.3.2, is suitable before continuing with motion planning processes, which are performed afterwards. This can provide invaluable information for to user, as it helps convey what is happening and assists in adjusting parameters if the AOLP software experiences difficulties in programming a specific process.

3.3.5 Code Generation

The final step of the programming process involves generating a robot code file that can be uploaded into the real world robotic system for use. This involves converting

simulated motions from the OLP environment into the program language of the target robotic hardware manufacturer, as well as adding any necessary I/O control signals for additional equipment in the workcell.

The difficulties associated with generating these robot code files poses more of an issue for generic OLP software, as they often require compatibility with many different robot manufacturers. Currently, the AOLP system has the capability to generate robot code files in three different robot languages: Kawasaki Robotics AS language, ABB Robotics Rapid language and Kuka Robotics language. As more applications featuring different robot manufacturers are encountered by the AOLP system, this library of compatible robot code languages will increase.

The actual conversion of the robot motions generated with the AOLP system into the target robot language is relatively trivial. The main difficulty relates to correctly encoding and sequencing the I/O signals for additional equipment featured in the robotic system. For example, as robots programmed via offline methods require some form of local calibration procedure, the commands for the laser (or other) calibration system used by the robot will then need to be incorporated into the generated robot code file in the correct sequence. This is further complicated by the existence of many different calibration methods; each featuring their own unique signal communication protocols, and different manufacturers of calibration equipment that exist; most of whom utilise different programming syntaxes. The tooling performing the manufacturing process, a weld torch for example, will also feature its own set of signal processing protocols that need to be followed as well. All of these external I/O signals must be compiled and seamlessly incorporated into the algorithmic processes that generate the complete robot code file for the AOLP system. In this experience, debugging and testing the algorithms that generate these code files without causing syntax or command sequencing errors entails significant effort. Example codes generated for Kawasaki robots, including all I/O signals for calibration and welding processes, are presented in Appendix Section 7.2 (page 181).

3.4 DEVELOPMENT CONSIDERATIONS

In this Chapter's final section, some additional considerations that must be addressed in order to convert the conceptual AOLP approach presented so far into a functional programming system suitable for real world programming applications are presented.

3.4.1 Programming System

To develop a functional AOLP system, a programming platform that will allow us to develop, control and test the algorithms that carry out the programming process is required. For the specific application presented in this thesis, the MATLAB programming environment was used. MATLAB provides a useful interface for scripting and debugging matrix-based algorithms. MATLAB can also be used to create customisable GUI interfaces that allow the user to initiate and interact with the algorithmic processes involved in AOLP. MATLAB also has fully developed plotting and graphical simulation capabilities, which would have to be developed from scratch if more low-level programming systems, such as C++, were used.

3.4.1 Spreadsheet Interface

At several stages of the AOLP process, the user needs to interact with the system to input data that can be used by the algorithmic processes involved in AOLP. This interaction is facilitated by a Microsoft Excel spreadsheet, which acts as a database for this information. Excel is used primarily as is it provides an effective means to enter, view and modify data. In addition, it features an interface that can be easily understood and used by people with low-level, or no, programming skill.

3.4.1 Coordinate Systems

The use of coordinate systems is fundamental to defining the location and orientation of objects in a virtual environment. A number of different types of coordinate system exist, each featuring their own particular strengths and weaknesses. Some coordinate conventions, such as Cartesian and Euler coordinates, are easy for a human to visualise.

Other coordinate systems, such as quaternion representations, are devised for computational efficiency and exactness, rather than user friendliness. These considerations make the selection of appropriate coordinate system of significant importance, and if multiple coordinate systems are used, the ability to translate between them is necessary. A summary of the coordinate systems used in the developed AOLP system is given in Appendix Section 7.3.1 (page 188).

3.4.2 Robot Kinematics

Robot kinematics involves the algorithmic calculation of the motions a robot must make in order to achieve a specified configuration. The study of kinematics is divided into two distinct categories, forward and inverse kinematics. Forward kinematics involves the calculation of a robots TCP position, based on a set of joint angles. Conversely, inverse kinematics involve determining a set of joint variables that will position a robots TCP at a given location and orientation. For the calculation of both forward and inverse kinematic problems, the Denavit-Hartenberg (D-H) [95] convention is particularly effective. The general D-H approach assigns individual coordinate systems to each link of the robots chain, starting from the base. Homogenous coordinate transformation matrices are then used to relate the position of each link to each other in order to solve the given kinematic problem.

For the developed AOLP system, forward kinematics are calculated using Corke's robotics toolbox [96], developed specifically for MATLAB. In order to work on the developed AOLP system, robot specific data relating to link geometry, joint limits and tooling are encoded into a forward kinematics function. An array $[q_1,...,q_6]$, containing the robot configuration parameters acts as an input to the function, which then returns another array containing the corresponding position of the robots TCP. This function also specifies the position of each robot joint, which is used for updating collision models of the robot (covered in Section 3.4.3).

Inverse kinematic problems involve determining a set of joint variables that will position a manipulator's end effector at a desired position and orientation. The use of inverse kinematics is essential in the planning stages of OLP, as manipulation tasks are

typically formulated in terms of Cartesian positioning of the robots tool. Inverse kinematics problems are more difficult to solve than forward kinematics. Often multiple solutions are available, and if joint axes coincide, an infinite array solutions need to be handled. In this work, inverse kinematics functions are formulated using the D-H representation in conjunction with closed-form symbolic equations based on the works of Paul and Zhang [97]. This closed-form approach is found to be more efficient than iterative solutions. Unlike the iterative method, which can only find a solution closest to the given reference joint target, the closed form method is able to find all solutions for all possible robot configurations, making it much more useful in robot trajectory and motion planning. Several robot-specific kinematic functions have been developed for use in the AOLP system: Kawasaki FA10N, Kawasaki ZX300, ABB IRB4400 and KUKA KR30 Li16-2.

3.4.3 Collision Processing

In the AOLP system, collision processing is used to determine whether the boundaries of two given CAD models are intersecting with one another. In the AOLP system, collision processing algorithms are used to plan and generate collision-free robot motions and tool placements during three steps of the AOLP programming process:

- *Tool path generation*: Collision processing algorithms used to generate optimal tool paths for a manufacturing process that do not produce collision between the tool and workpiece.
- *Robot placement*: For a given tool path, the robot placement algorithm utilises collision processing to place the robot in a location that allows the robot to carry out the path without colliding with its surrounding environment.
- *Motion planning*: Collision processing algorithms used to generate collision-free transition motions that navigate the robot from its home position to the location where the manufacturing process is performed.

In the AOLP system, collision processing is performed between the dynamic elements of the robotic system (i.e. the robot and tool) and their surrounding environment. Each

time the location of a dynamic element is changed, a collision check is performed to ensure the updated positing has not induced a new collision.

Collision processing is, by far, the most computationally expensive algorithmic process involved in the AOLP process. This is partly due to complexity of the geometric algorithms used, and also to the fact that these collision processing algorithms are typically called hundreds of thousands of times during the programming process for a typical manufacturing task. The AOLP system's use of simplified collision models, presented in Section 3.2.1, was done as a direct means to reduce these computation time. The mathematical methods used to calculate collision between these simplified CAD geometry is presented in Appendix Section 7.3.4 (page 192).

4 DEVELOPMENT OF MOTION PLANNING AND PATH OPTIMISATION ALGORITHMS

In this Chapter, the development of motion planning and path optimisation algorithms for use in AOLP are presented. As the development of these algorithms represents a significant amount of research and testing, they are given their own Chapter in order to properly document this process.

4.1 MOTION PLANNING

4.1.1 Introduction

Motion planning is a key component of the AOLP process. Up until this stage of the offline programming process, tool paths and associated robot positioning to carry out the manufacturing process have been specified. The goal for a motion planning algorithm is to generate a series of motions that can guide the robot from its home position to the location where the manufacturing process is to be carried out, and back again.

For industrial applications, selection of a suitable motion planning algorithm is of critical importance. The search algorithm must produce motions that are smooth, collision-free, and well within the reach of the robot, whilst avoiding singularities and joint limits. It is further desired that the planning algorithm be fast when the planning problem is simple, but still able to find solutions for difficult problems when they arise [98].

The Lazy PRM planner [50] (LPRM) is one variant of particular interest as it was found to perform particularly well for the specific robotic application presented in this work. As discussed in Section 2.2.4, the LPRM algorithm was developed as an evolution of the popular Probabilistic Roadmap Planner (PRM) [45]. The LPRM planner is able to generate collision-free paths in high dimensional configuration space (C_{space}) using only a fraction of the collision checks of its predecessor, allowing it to be used in both single and multiple query style problems. This makes it particularly useful for a robotic manufacturing application. If the workcell or robotic setup is changed, the LPRM

algorithm's efficient approach allows it to rapidly generate a new solution from scratch. The resultant roadmap graph solving this initial query can then be stored as a global roadmap, which can be used as a basis to assist in solving other motion planning queries. Each time a motion planning query is solved, any new configurations or motions sampled are then added to this global roadmap, which is then used to solve subsequent queries later on. The result is a motion planning algorithm that becomes more effective as more motion planning queries are solved, until the environment or robot model changes and the process is restarted..

In Section 4.1.2, the original Lazy-PRM algorithm, including all relevant terminology and algorithmic details, is first presented. The remainder of Section 4.1 then presents the development of a new variant of this algorithm, called the Lazy Significant Edge Algorithm (LSEA). The LSEA algorithm is an evolution of the original LPRM algorithm, and utilises a novel non-uniform sampling heuristic to more effectively solve motion planning queries. The developed LSEA algorithm is presented in detail in Section 4.1.3. In Section 4.1.4, the effectiveness of the LSEA algorithm is evaluated alongside its predecessor, the LPRM algorithm, with a series of different motion planning problems designed to compare their respective performances. To conclude the presentation of the LSEA algorithm, a discussion of the method is conducted in Section 4.1.5.

4.1.2 The Lazy-PRM Algorithm

An overview of the LRPM planner is outlined in Figure 2-8 (page 49), and operational details are presented in presented in Algorithm 4-1 and Algorithm 4-2 below. The LPRM algorithm begins with the construction of an initial roadmap graph $G = (N, E)$ consisting of a set of nodes $N = \{n_1, n_2, \dots\}$, and interlinking edges $E = \{e_1, e_2, \dots\}$. During the construction phase of the LPRM algorithm, configurations are generated randomly from a uniform distribution, and are inserted into G as nodes. The queries' start and goal nodes, n_s & n_g, are also included in this set. Nearby configurations are then linked together with corresponding robot motions, which are inserted into G as edges. All nodes and edges generated at this stage are not checked for collision. Instead, their status is marked as 'unchecked', which is updated later in the planning process.

Algorithm 4-1 The Lazy PRM algorithm.

```
LAZY_PRM
1: Construct Initial Roadmap: G = (V, E)
2: loop
3.    Π = find shortest path through G
4.    if Π = Ø:   - no path exists
5.        call function: NODE_ENHANCEMENT(G) (Algorithm 4-2)
6.    else
7.       collision test Π
8.         if Π ∈ C_free
9.              Solution found. return Π
10.       else
11.             remove element from G
12.       end if
13.   end if
14. end loop
```

The query phase of the LPRM algorithm is initiated once the construction of the initial roadmap is complete. This phase uses an iterative process of searching for the shortest path through G, between n_s and n_g, and then checking the returned path for collision. Collision checking is performed by a local planning algorithm, which is able to deduce whether a particular robot configuration or motion lies in the free configuration space (C_{free}), or is intersecting with an obstacle at the boundary of the forbidden configuration space (C_{forbid}). If any node or edge on a returned path is invalidated by the local planner, i.e. it does not lie entirely in C_{free}, then it is immediately removed from the roadmap, and the next shortest path is queried. This process continues until it is terminated in one of two ways:

1. A C_{free} path between n_s and n_g is found, solving the given motion planning query
2. No path between n_s and n_g exists, a situation in which the enhancement phase is called.

The purpose of the enhancement phase is to link additional nodes into G, so that n_s and n_g are reconnected into the same component and the motion planning process can continue. The configurations of these new nodes are generated through a combination of

random and heuristic methods. The configurations generated randomly in this phase are key to the LPRMs overall effectiveness: Random sampling ensures that, given enough time, a solution will be found provided that one exists (see probabilistic completeness [52]) . The heuristic technique used in the LPRM's enhancement phase attempts to sample robot configurations at the boundary of obstacles. This is done to improve the efficiency of the motion planning process by placing new configurations in workspace regions that randomised sampling has a low probability of capturing efficiently.

The heuristic method of sampling in the LPRM algorithm is detailed in Algorithm 4-2 and Algorithm 4-3. This sampling is done by selecting a number of points in the roadmap, called seeds, and randomly distributing new configurations nearby each of them. All roadmap edges that have been previously invalidated and removed from the roadmap are collected into a new set E'. For each edge in this set, we check if at least one of its end points is C_{free}. On line 3 of Algorithm 4-3, a seed is generated at the midpoint of every edge in E' that satisfies this criterion. These edges must lie near the boundary of an obstacle, so generating a configuration nearby the midpoint of these edges increases the probability of generating a configuration in a useful region in C_{space}. To stop any cyclic behaviour of this heuristic sampling scheme, seeds are only generated on an edge that links together two randomly generated nodes. A detailed overview of the LPRM planner, including all operational details and terminology, can be found in [50].

Algorithm 4-2 The node enhancement function.

NODE_ENHANCEMENT(G)

input: G – disconnected roadmap graph.

output: $\acute{G}$ – modified roadmap.

1. S = set of roadmap seeds. Generated from function SELECT_SEEDS(G)
2. **for** i = 1 **to** $|S|$
3. $H\{i\}$ = configuration sampled nearby $s\{i\} \in S$
4. **end for**
5. R = set of randomly generated nodes
6. Connect all new nodes to roadmap: $\acute{G} = G \cup \{H, R\}$
7. **return** $\acute{G}$

Algorithm 4-3 Select seeds heuristic – original LPRM method

SELECT_SEEDS(G) – *Original LPRM method*

input: G – disconnected roadmap graph.
output: S – set of roadmap seeds.
1. E′ = set of all roadmap edges $e \in C_{forbid}$
2. **for** i = 1 **to** $|E'|$
3. **if** $e\{i\}$ has one end point $\in C_{free}$
3. $S\{end\} =$ configuration at midpoint of $e\{i\} \in E'$
5. **end if**
6. **end for**
7. **return** S

In this Chapter an improved heuristic sampling technique for the LPRM planner, called the significant edge heuristic, is presented. This method distinguishes itself by not only sampling at the boundary of obstacles, but by sampling about the boundary of obstacles that have not yet been navigated by the robot. This improves performance by eliminating redundant sampling about obstacles that have already been cleared. This Chapter first presents the significant edge heuristic and compares its operation to the sampling heuristic used in the original LPRM planner (Algorithm 4-3). The performance of this new approach is then demonstrated on a number of different motion planning problems, along with a discussion relating to the effectiveness of the significant approach.

4.1.3 The Lazy Significant Edge Algorithm

The goal of the significant edge heuristic is to sample configurations about obstacles in the robots C_{space} that have not yet been navigated. This heuristic operates with a similar methodology to the LPRM planner's sampling heuristic: A number of special robot configurations, referred to as seeds, are selected about C_{space}. Additional configurations are then sampled nearby these points and linked into the roadmap. The critical difference between these two heuristic methods is the way in which these seeds are selected.

The operation of the significant edge heuristic is outlined in Algorithm 4-4, which is called from the LPRM's enhancement phase (Algorithm 4-2). Initially, all roadmap

edges that have been invalidated and removed from G in previous iterations are collected into a set of candidate edges E'. Then, one-by-one, each edge in E' is inserted back into G, and a test is performed to check whether the query nodes n_s and n_g have been rejoined into the same connected component. That is, given the graph $G = (N, E)$, let $\sim_G$ be the equivalence relation on vertices of G generated by $n_1 \sim_G n_2$ if there is a valid path between vertexes n_1 and n_2, where $\{n_1, n_2\} \in N$. For each element e' in the set of invalid edges E', we ask whether $n_s \sim_{G \cup \{e\prime\}} n_g$. Edges that satisfy this condition are deemed significant and are kept in E', otherwise they are discarded from the set. On lines 9-11 of Algorithm 4-4, seeds are then defined as the midpoint of each of these significant edges. In Algorithm 4-2, new roadmap configurations are then generated randomly within a certain distance of each of these seeds. A number of randomly generated configurations are also sampled during the enhancement phase, and all new configurations generated are then linked into the roadmap graph. In order to reduce cyclic behaviour in consecutive roadmap expansions, significant edges are not used multiple times.

A useful property of significant edges is that they always occur about an obstacle that has not yet been navigated by the robot. If an invalid edge is placed back into G, but does not reduce the component count (refer Figure 4-1 detail (e)), the clash object obstructing it has effectively already been navigated by the robot. Another complimentary feature of the significant edge heuristic is that no calculations rely on explicit representations of C_{space}. Rather, they are calculated from roadmap information alone, providing an efficient method of calculation, regardless of the complexity of the robot, or its surrounding C_{space}.

Algorithm 4-4 Select seeds heuristic – significant edge method.

SELECT_SEEDS(G) – *Significant Edge method*

input: G – disconnected roadmap graph

output: S – set of roadmap seeds

1. E' = set of all roadmap edges $e \in C_{forbid}$
2. **for** $i = 1$ **to** $|E'|$
3. $\acute{G} = G \cup \{e\{i\}\}$: insert $e\{i\} \in E'$ into G:
4. **if** $component\{n_s\} = component\{n_g\}$: roadmap reconnected
5. $e\{i\} \in E'$: keep $e\{i\}$ in E'
6. **else**
7. $E' = E' - e\{i\}$: discard $e\{i\}$ from E'
8. **end if**
9. **end for**
10. **for** $i = 1$ **to** $|E'|$
11. $S\{i\} =$ configuration at midpoint of $e\{i\} \in E'$
12. **end for**
13. **return** S

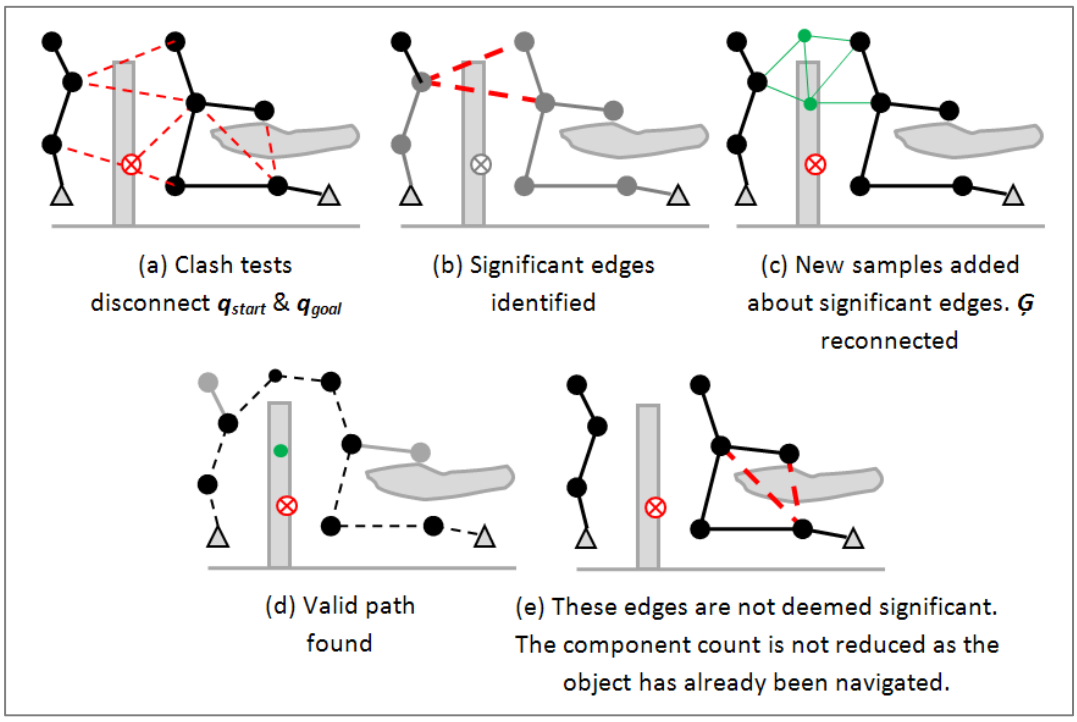


Figure 4-1 Overview of the significant edge heuristic.

4.1.4 Experiments

A series of motion planning problems were developed to test the general effectiveness of the significant edge heuristic when applied to the LPRM planner. For benchmarking purposes, the original variant of the LPRM planner is tested alongside.

Three path-planning problems, shown in Figure 4-2, Figure 4-3 and Figure 4-4, were developed to test both algorithms. Problem 1 is carried out on a 2D plane, which features a densely scattered array of circular obstacles. The robot used in this environment, which has two translational and one DOF, is tasked with finding a path from one corner of the environment to the other. Problem two features a more complex 3D environment, and a 6-DOF free flying robot. The robot must navigate a range of spheres and a large planar wall. The last problem features a 6-DOF articulated manipulator, which is mounted on a linear rail to form a 7-DOF robotic mechanism. This manipulator structure is tasked with navigating a 3D industrial environment, from one end of the rail to the other. This problems difficulty is increased by introducing joint limits on the manipulators' base joint, forcing its motion into the midst of a series of obstacles to solve the given query. Both motion planning variants featured in these tests use the same variables to control the overall motion planning algorithm, which are presented in Table 4-4.

All results are presented in Table 4-1, Table 4-2 and Table 4-3. These results are given in terms of the number of collision checks required to solve the given query, and the time taken. To reduce error, all results presented are averaged over 100 individual tests. All of the motion planning algorithms and environments were created in the MATLAB software package.

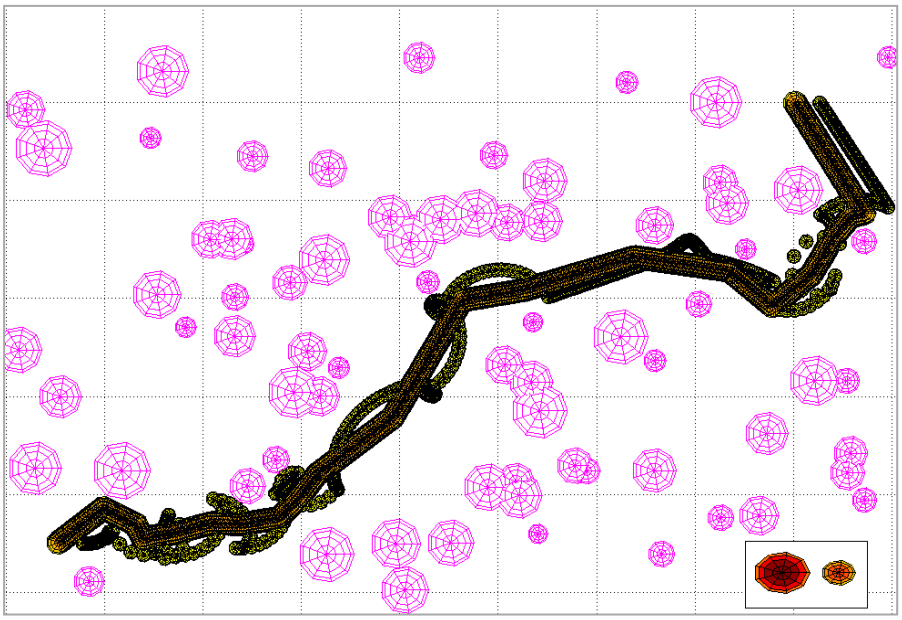


Figure 4-2 Problem 1: 2D environment and sample solution
Robot model shown bottom right.

Table 4-1 Results from Problem 1

Heuristic Used	# Clash Tests	Time (sec)
Original	645.9	23.4
Significant Edge	398.7 (38.3%)	13.2 (43.5%)

(% improvement shown in brackets).

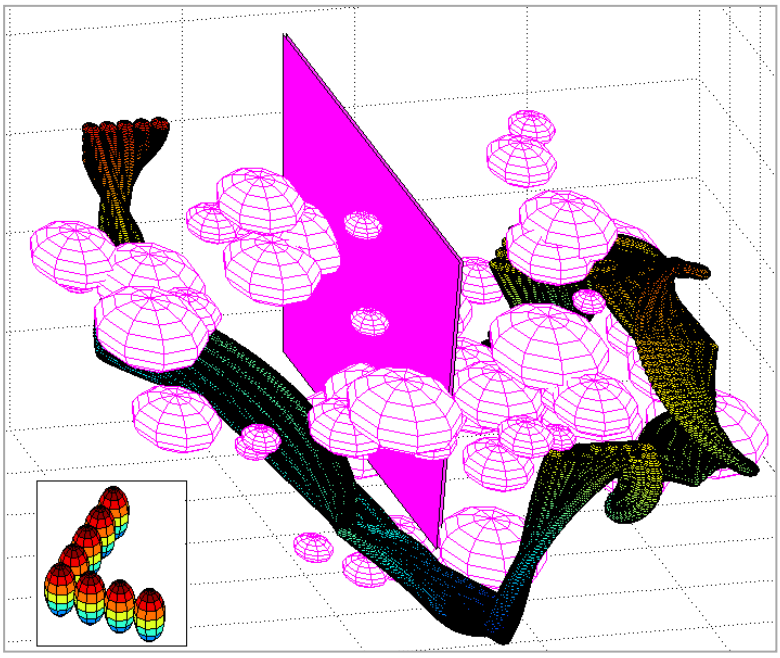


Figure 4-3 Problem 2: 3D environment and sample solution.
Robot model shown bottom left.

Table 4-2 Results from Problem 2.

Heuristic Used	# Clash Tests	Time (sec)
Original	841.3	42.4
Significant Edge	503.9 (40.1%)	27.1 (36.1%)

(% improvement shown in brackets).

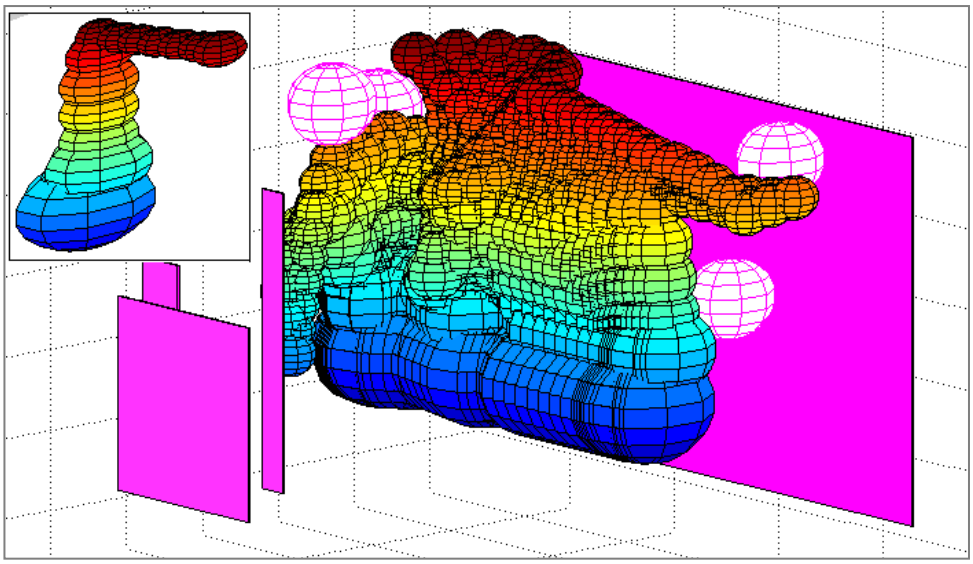


Figure 4-4 Problem 3: 7-DOF environment and sample solution.
Robot model shown top left.

Table 4-3 Results from Problem 3.

Heuristic Used	# Clash Tests	Time (sec)
Original	709.7	52.4
Significant Edge	452.8 (36.2%)	35.6 (32%)

(% improvement shown in brackets).

Table 4-4 LPRM parameters used

Test	Init G	nNeighb	nSeed	nSamp	Rad	nRand
2D	200	5	15	2	300	10
3D	250	5	15	2	500	10
7-DOF	300	5	20	2	500	15

- **Init G:** Number of Nodes in initial Roadmap.
- **nNeighb:** Number of neighbours each node is connected too.
- **nSeed:** Number of seeds to use in each enhancement phase.
- **nSamp:** Number of configurations to sample about each seed.
- **Rad:** Max radius a sample is generated from a seed (mm).
- **nRand:** Number of configurations generated randomly in the enhancement phase.

4.1.5 Significant Edge Discussion

In each of the tests conducted, the LPRM algorithm performed considerably better with the significant edge heuristic incorporated into its operation. Problem 1 was solved an average 43% faster with the significant edge heuristic in place. Similar levels of performance were observed in problem 2, with a 36% improvement in time and 40% reduction in collision checks used. The 7-DOF manipulator setup received the smallest performance improvements from the significant edge heuristic, the time and collision checks required to solve the problem were reduced by approximately a third.

Whilst both sampling heuristics featured in these tests are similar in that they generate additional samples about seeds in the roadmap, an important distinction between the two methods is in the way that these seeds are selected. In the original LPRM's heuristic, seeds are generated from any edge that is invalidated by collision checking processes. By sampling configurations randomly about these seeds, it is possible to generate them at the boundary of obstacles. However if the robot has already navigated the obstacle in question, the computational effort expended in this process will have been for little reward. A beneficial property of a significant edge is that the obstacle that has invalidated it will always be an obstacle that has not yet been navigated by the robot. It therefore makes sense to add additional samples in the vicinity of these edges. Using a significant edge heuristic in a PRM-based planner ensures that as the motion planning process continues, the sampling bias will tend to focus exclusively about particularly difficult regions (a narrow passage for example) as the other, more open, areas are progressively navigated.

As reported in [50], the size of the initial roadmap has significant influence on the overall effectiveness of the LPRM algorithm. The general approach used in this initial work was to use very large initial roadmaps - in the vicinity of 10,000 nodes - in order to avoid the use of the node enhancement phase in solving a given problem. However with the significant edge heuristic incorporated into the node enhancement phase of the LPRM algorithm, using smaller initial roadmaps was observed to be effective. This can be primarily attributed to two main factors: Firstly, by using a small initial roadmap we are essentially forcing the enhancement phase at a much earlier stage of the motion planning process. This allows the LPRM algorithm to utilise its sampling heuristic more

effectively. Another benefit of using smaller initial roadmap sizes is that many of the graph-based algorithms, such as graph searching or neighbour queries, can be performed far more rapidly. Analysis of the effectiveness of the significant edge heuristic relative to the initial roadmap size presents an intriguing topic for future research.

Another beneficial property of the significant edge heuristic is that it requires only roadmap information to perform its calculations. This makes the algorithm generic enough to be incorporated into another type of PRM planner, such as an RRT or cell-based algorithm, or even be combined with other heuristic-based sampling algorithms, such as the Gaussian [70] or Bridge [71] samplers.

4.2 PATH OPTIMISATION

Whilst probabilistic motion planning algorithms are an effective approach to solving complex motion planning problems, a significant drawback of the method is the quality of the returned path. Whilst a raw path from the motion planning algorithm can potentially be used as-is, the fact that the path could be used repeatedly over hundreds or thousands of runs makes the path's quality of significant importance. To ensure paths are of suitable quality, a post process optimisation algorithm is used to condition paths before they are uploaded and used in the real world robotic system.

As discussed in the Section 2.3.2, the Partial Shortcut (P-Sc) algorithm, developed by [78], was found to be particularly effective at smoothing PRM paths generated for robotic manipulators. Paths are optimised one DOF at a time, resulting in reduced path length and positional overshoot. Its reliance on probabilistic methods also removes any need for complex mathematical representation of the surrounding environment. The disadvantage of the P-Sc algorithm is the increased computational effort required to optimise each robot DOF individually. In this section, a new partial shortcut algorithm, which adaptively selects robot DOFs to optimise, is presented. This new algorithm improves upon the original approach by reducing the number of collision checks required to optimise a given path.

This Section first presents a summary of notation relevant to describing these optimisation techniques. A formal description of the partial shortcut method is then presented, along with the new Adaptive Partial Shortcut algorithm. The effectiveness of this new approach is then evaluated with a series of tests on an industrial robotic manufacturing system.

4.2.1 Relevant Notation

Sampling-based motion planning and optimisation algorithms operate on the premise of a robot moving about its surrounding workspace *W*, without colliding with any obstacles. To properly characterise this foundation, some relevant notation is described.

A robot configuration π_i is defined by the set of numbers representing values of the individual robot DOFs needed to specify the robot position in *W* i.e. $\pi_i = [q_1, \dots, q_{q_n}]$ where q_n is the number of DOFs of the robot. The f^{th} value of configuration π_i can be referenced as $\pi_i[f] = q_f$.

A local path Π_i is a single motion from one individual robot configuration; π_i, to another; π_{i+1}, where $\Pi_i \in C_{free}$.

$$\Pi_i \in C_{free} = \{\pi_i, \pi_{i+1}\}$$

The motion of Π_i is defined using a local path planner, usually a linear interpolation in either Joint or Cartesian space. A local path that is kinematically feasible and resides completely within C_{free} is said to be valid.

A path $\overline{\Pi}$ is an ordered series of *n* local paths, i.e. $\overline{\Pi} = \{\Pi_1, \dots, \Pi_n\}$. A path cannot feature discontinuities i.e. a robotic device must be able to move from the initial configuration π_1 of Π_1 to the final configuration π_{n+1} of Π_n.

For an industrial manipulator, the length of a given path is described using a joint cost function $c_J(\dots)$, which produces an array containing the change in displacement used by each robot DOF to travel the path. This array is referred to as the joint cost array. For a local path Π_i, which features only one direct motion:

$$c_J(\Pi_i) = [|\pi_i[1] - \pi_{i+1}[1]|, \dots, |\pi_i[q_n] - \pi_{i+1}[q_n]|] \qquad (1)$$

The resultant cost array contains the change in displacement of each robot DOF used to travel from π_i to π_{i+1}.

For a path $\overline{\Pi}$, we sum the joint costs across the *n* local paths contained within it i.e.

$$c_J(\overline{\Pi}) = \left[\sum_{i=1}^{n} |\pi_i[1] - \pi_{i+1}[1]|, \dots, \sum_{i=1}^{n} |\pi_i[q_n] - \pi_{i+1}[q_n]|\right] \quad (2)$$

The resultant cost array produces the change of displacement of each robot DOF used to travel the entire path, from π_1 of Π_1 to π_{n+1} of Π_n.

In the tests conducted in Section 4.2.3**,** the P-Sc algorithm is terminated using a total cost function $c_T(\dots)$. The function produces one single value and is a summation of each element in the joint cost array produced from $c_J(\dots)$:

$$c_T(\overline{\Pi}) = \sum c_j(\overline{\Pi}) = \sum \left[\sum_{i=1}^{n} |\pi_i[1] - \pi_{i+1}[1]|, \dots, \sum_{i=1}^{n} |\pi_i[q_n] - \pi_{i+1}[q_n]|\right] \quad (3)$$

4.2.2 The Partial Shortcut Algorithm

Path optimisation algorithms such as the path pruning and shortcut methods presented in Section 2.3 reduce the overall path cost by bypassing robot configurations wherever possible. The P-Sc method, outlined in Algorithm 4-5, operates instead by decoupling the robot DOFs, and optimizing path configurations one DOF at a time.

The P-Sc algorithm starts by selecting a robot DOF, *f*, randomly from predefined weighted distribution. This distribution is spread across all available robot DOFs, and is used to skew this random selection towards particular DOFs of interest. The input path $\overline{\Pi}$ is then split into three segments denoted $\overline{\Pi}', \overline{\Pi}'', \text{and } \overline{\Pi}'''$, using the randomly generated integers *a* and *b*, where *n* is the number of local paths belonging to $\overline{\Pi}$:

$$1 < a + 1 < b \leq n \text{ and}$$

$$\overline{\Pi}' = \{\Pi_1, \dots, \Pi_a\}, \overline{\Pi}'' = \{\Pi_{a+1}, \dots, \Pi_b\}, \overline{\Pi}''' = \{\Pi_{b+1}, \dots, \Pi_n\}$$

For the random DOF f, all configurations $\pi_i''|i = 1 \dots m+1$ in $\overline{\Pi}''$ are replaced by the equivalent value linearly interpolated between 1 and $m+1$, where: $m = b - a$ is the number of local paths in $\overline{\Pi}''$. If the f^{th} value of configuration π_i'' is $\pi_i''[f]$, then

$$\pi_i''[f] = {}^{\text{int}}\pi_i''[f] \mid i = 1 \dots m+1 \quad (4)$$

Where ${}^{\text{int}}\pi_i''[f]$ is the i^{th} interpolated value of the f^{th} DOF between $\pi''_1[f]$ and $\pi''_{m+1}[f]$. If the modified segment is found to be valid, i.e.

$$\overline{\Pi}'' \in C_{\text{free}}$$

then the path Section $\overline{\Pi}''$ is inserted back to the original path.

$$\overline{\Pi} = \overline{\Pi}' \cup \overline{\Pi}'' \cup \overline{\Pi}'''$$

Otherwise, $\overline{\Pi}''$ generated from this iteration is discarded. This process is then repeated iteratively until a termination criterion is met.

The P-Sc algorithm does not add any additional samples to, or around, the input path; nor are any removed. P-Sc relies solely on modifying the configuration of path nodes. Since the algorithm only applies shortcuts to a single robot DOFs at a time, it proves less efficient at reducing overall path length than other shortcut methods. Isolating the interpolation to a single joint, however, provides an effective means to remove excessive joint overshoot.

Algorithm 4-5 The partial shortcut algorithm.

PARTIAL SHORTCUT($\overline{\Pi}$)

input $\overline{\Pi}$ – Path for optimisation
output $\overline{\Pi}$ – The optimised path
$n \leftarrow length(\overline{\Pi})$
loop
  $f \leftarrow$ a random degree of freedom
  $a, b \leftarrow$ two random indices: $1 < a + 1 < b \leq n$
  $\overline{\Pi}' \leftarrow \overline{\Pi}\{\Pi_1, \dots, \Pi_a\}$
  $\overline{\Pi}'' \leftarrow \overline{\Pi}\{\Pi_{a+1}, \dots, \Pi_b\}$
  $\overline{\Pi}''' \leftarrow \overline{\Pi}\{\Pi_{b+1}, \dots, \Pi_n\}$
  $m \leftarrow (b - a)$
  for $i = 1$ **to** m
    $\pi_i''[f] \leftarrow \text{INTERPOLATE}\left(\pi_i''[f], \pi_{m+1}''[f], i/(m+1)\right)$
  end
  if $\overline{\Pi}'' \in C_{free}$ **then**
    $\overline{\Pi}'' \leftarrow \overline{\Pi}' \cup \overline{\Pi}'' \cup \overline{\Pi}'''$
  end
end loop

4.2.3 Adaptive Partial Shortcuts

It was found that when operating on an industrial robotic system, the P-Sc algorithm spends the majority of its computation time, up to 90%, processing collision checks. Therefore, reducing the number of collision checks required to smooth a path provides an effective means to improving the overall efficiency of the algorithm. This reduction will be more effective in an industrial manufacturing environment, where collision checks are more computationally expensive to process. In this Section a new adaptive partial shortcut algorithm is presented. The new approach is based on the original P-Sc algorithm described in Section 4.2.2, and features a dynamic method of selecting partial shortcuts. The pseudo-code for the approach is given in Algorithm 4-6, and its operational details are presented below.

4.2.3.1 *Adaptive Weighting Distribution*

On line 3 of Algorithm 4-5, a single robot DOF f is selected from a weighted distribution. For example [78] used a weighting of [6,6,6,2,2,2] to bias the selection of f for an articulated industrial robot with 6 DOF towards the robot's major joints (J_1, J_2, J_3) over the minor joints (J_4, J_5, J_6). Generally, this distribution works well as the optimisation goal is to minimise total Cartesian motion. However there are certain situations where this fixed distribution may have a negative impact on the overall performance of the P-Sc algorithm; for instance when trying to optimise a path with excessive joint motion on joints J_4, J_5 and J_6. An adaptive approach to bias the selection of f is proposed to address this issue.

The Adaptive Weighting Distribution (AWD), shown in Algorithm 4-7, is a distribution that is updated at each iteration of the P-Sc algorithms main loop. The AWD method aims to generate a distribution that will skew the selection of f towards the joints that deviate furthest from the optimal solution, Π_{ref}. For the input path $\overline{\Pi}$, the reference path Π_{ref} is a local path travelling directly from the start configuration π_1 to the goal π_{n+1}, it assumes that no obstacles are present in the problem:

$$\Pi_{ref} = \{\pi_1, \pi_{n+1}\} \text{ where } C_{space} = C_{free}$$

Used as a reference, Π_{ref} is the most optimal path possible for a particular motion planning problem, as the local planner directly joins the start and goal configurations without any of the excess motion needed to avoid obstacles about C_{space}.

In Algorithm 4-7, joint cost arrays for the local path Π_{ref} and the current path being optimised, $\overline{\Pi}$, are calculated using the joint cost functions (1) and (2). The AWD is then produced from the difference between resultant joint cost arrays:

$$AWD = \left[\left|\left[c_J(\overline{\Pi})\right] - \left[c_J(\Pi_{ref})\right]\right|\right] \quad (5)$$

When used to select *f*, the resultant AWD will bias the selection towards the DOFs that deviate the furthest from the optimal solution. As the optimisation process continues, the outliers in the AWD are incrementally reduced, and the bias becomes more evenly spread to the other remaining robot DOFs. In Section 4.2.4.2, the effectiveness of the proposed AWD method is tested alongside several static weighting distributions.

4.2.3.2 *Optimal Number of Joints to Interpolate*

The original P-Sc approach carries out shortcuts on single robot DOFs. If the partial shortcut is instead carried out simultaneously on a number of DOFs, say two or three, the efficiency of the algorithm may be improved.

To test this concept, and to find an optimal number of DOFs to shortcut, the Adaptive P-Sc algorithm includes the function selectDOF. This function, called on line 5 of Algorithm 4-6 and shown in Algorithm 4-8, facilitates the selection of manipulator joints to be partially interpolated. Two variables are passed into this function: nDOF, which specifies how many robot DOFs are to be selected, and AWD; the current adaptive weighted distribution. nDOF robot DOFs are then selected from the AWD, in the same manner as in the original P-Sc algorithm, and the partial shortcut is carried out simultaneously on this group. If a random selection is used for nDOF, the random number of joints to select is evaluated each time the function selectDOF is called.

A series of tests were proposed to find an optimal value for nDOF, the candidates shown in Table 4-5. These tests are carried out and summarised in Section 4.2.4.

Table 4-5 Inputs for SelectDOF tests.

nDOF	DOFs to Select Per Iteration
1	One DOF
2	Two DOFs
3	Three DOFs
Rand(1/2)	Randomly select either one or two
Rand(2/3)	Randomly select either two or three
Rand(1-3)	Randomly select between one and three

4.2.3.3 *Pre-test Potential Improvement*

Due to the random nature in which path segments and robot DOFs are selected by the P-Sc algorithm, there exists a probability that a particular partial shortcut will be duplicated at a later stage in the optimisation process. This probability is higher in shorter paths, and was also observed to increase as the P-Sc algorithm nears completion. If a particular partial shortcut is duplicated, a significant amount of computational effort will be wasted as collision checks will be carried out to validate a path segment that provides no benefit to the cost of the overall path. A method to avoid this duplication of partial shortcuts is to evaluate the potential benefit a particular shortcut could possibly provide, before carrying out computationally expensive collision checking procedures. This is done on line 14 of Algorithm 4-6, immediately before the collision checking of the modified path $\overline{\Pi}''$ is done. Only if the modified path $\overline{\Pi}''$offers a reduction in total cost over the original segments, is the path then checked for collision. Otherwise the algorithm immediately jumps to the next iteration, eliminating the chance that a particular partial shortcut is carried out more than once. This feature of the Adaptive P-Sc algorithm is evaluated in Section 4.2.4, below.

Algorithm 4-6 The adaptive partial shortcut algorithm.

ADAPTIVE-PARTIALSHORTCUT($\overline{\Pi}$)

input $\overline{\Pi}$ – Path for optimisation

output $\overline{\Pi}$ – The optimised path

1: $n \leftarrow length(\overline{\Pi})$
2: $\Pi_{ref} = \overline{\Pi}\{\pi_1, \pi_n\}$
3: **loop**
4: $AWD \leftarrow$ genAWD$(\overline{\Pi}, \Pi_{ref})$
5: $f \leftarrow$ selectDOF$(AWD, nDoF)$
6: $a, b \leftarrow$ two random indices: $1 < a + 1 < b \leq n$
7: $\overline{\Pi}' \leftarrow \overline{\Pi}\{\Pi_1, \dots, \Pi_a\}$
8: $\overline{\Pi}'' \leftarrow \overline{\Pi}\{\Pi_a, \dots, \Pi_b\}$
9: $\overline{\Pi}''' \leftarrow \overline{\Pi}\{\Pi_b, \dots, \Pi_n\}$
10: $m \leftarrow (b - a)$
11: **for** $i = 1$ **to** m
12: $\pi_i''[f] \leftarrow$ INTERPOLATE$(\pi_i''[f], \pi_m''[f], i/(m))$
13: **end**
14: **if** $c_T(\overline{\Pi}'') < c_T(\{\Pi_a, \dots, \Pi_b\})$
15: VALIDATE PATH($\overline{\Pi}''$)
16: **if** $\Pi'' \epsilon\, C_{free}$ **then**
17: $\overline{\Pi}'' \leftarrow \overline{\Pi}' \cup \overline{\Pi}'' \cup \overline{\Pi}'''$
18: **end**
19: **end**
20. **end loop**

Algorithm 4-7 Generate adaptive weight distribution function.

genAWD$(\overline{\Pi}, \Pi_{ref})$

input $\overline{\Pi}$ – Current Path. Π_{ref} – Reference Path.

output AWD – The adaptive weight distribution

1: $AWD = \left[\left|\left[c_J(\overline{\Pi})\right] - \left[c_J(\Pi_{ref})\right]\right|\right]$

Algorithm 4-8 Adaptive select DOF function.

selectDOF(AWD, $nDoF$)

input AWD – Current Weight Distribution. $nDoF$ – number of DOFs to select

output f – DOFs to partially interpolate.

1: **if** *nDOF* **is** *a random selection* **then**
2: n = random integer, specified by *nDOF*
3: **else**
4: $n = nDOF$
5: **end if**
6: $f = \{f_1, \ldots, f_n\}$
7: **for** $i = 1$ **to** n
8: f_i = a random selection from AWD
9: **end**

4.2.4 Experimental Results

4.2.4.1 *Setup*

A series of tests were conducted to assess the performance of the Adaptive P-Sc algorithm. To do this four sample paths were generated. These paths feature large variances in length and complexity, and are indicative of paths typically encountered by the industrial manipulator system.

Test path 1 consists of four individual robot motions travelling from a home position to a tool rack, where a gripping end effector is loaded. The required distance for the robot to travel is short and no tool is mounted. The resultant path returned from the PRM planner only features a small degree redundant motion. Path 2 simulated the robot moving from the tool rack, with a gripper now attached, to a handling bay on the other side of its configuration space. This path traced a similar region of space to the first, but was extended somewhat. Some additional obstacles were present, and the attached end effector increases the difficulty. Path 3 simulates the robot picking up a metal plate from a rack, and moving to position the plate on a workbench. The path length is longer than previous tests and intricate motions are required to both pick up and position the metal plate. The large size of the plate being transported also complicates the path planning

problem. The resultant path is long, with many more nodes present. The path also features a lot of redundant joint motions, which are required to navigate the obstacles present in the problem. The fourth and final test path was similar to the third, however the path length was extended, and extra obstacles were scattered about the workspace to increase the difficulty of the problem. The purpose of this path is to test the performance of the adaptive P-Sc algorithm in a difficult scenario. Details of each of these test paths are given in Table 4-6 and Figure 4-5. To determine the effectiveness of the adaptive P-Sc algorithm, the various functions that distinguish it from the original P-Sc algorithm were tested independently of one another. These tests consisted of smoothing the four paths until the optimal path cost, shown in Table 4-6, was achieved. All results presented are averaged over 70 individual tests, and are given in terms of the number of calls to the collision checking algorithm. The results are presented graphically in the results Chapter below. The optimal path costs, used to terminate the optimisation process, were determined in a separate series of tests by exhaustively running the P-Sc algorithm on each path until it was optimised to a level deemed to be suitable for use in a real world robotic system. The OLP system and subsequent motion planning and optimisation algorithms used in these experiments were all developed and run in the MATLAB programming environment.

Table 4-6 Test path data.

Path	# Nodes	C_J	C_T	Optimal C_T
1	5	[130,50,104,312,119,85]	681	330
2	8	[200,167,120,288,222,333]	1330	550
3	15	[280,264,210,712,447,608]	2521	720
4	19	[305,255,284,784,538,868]	3034	1010

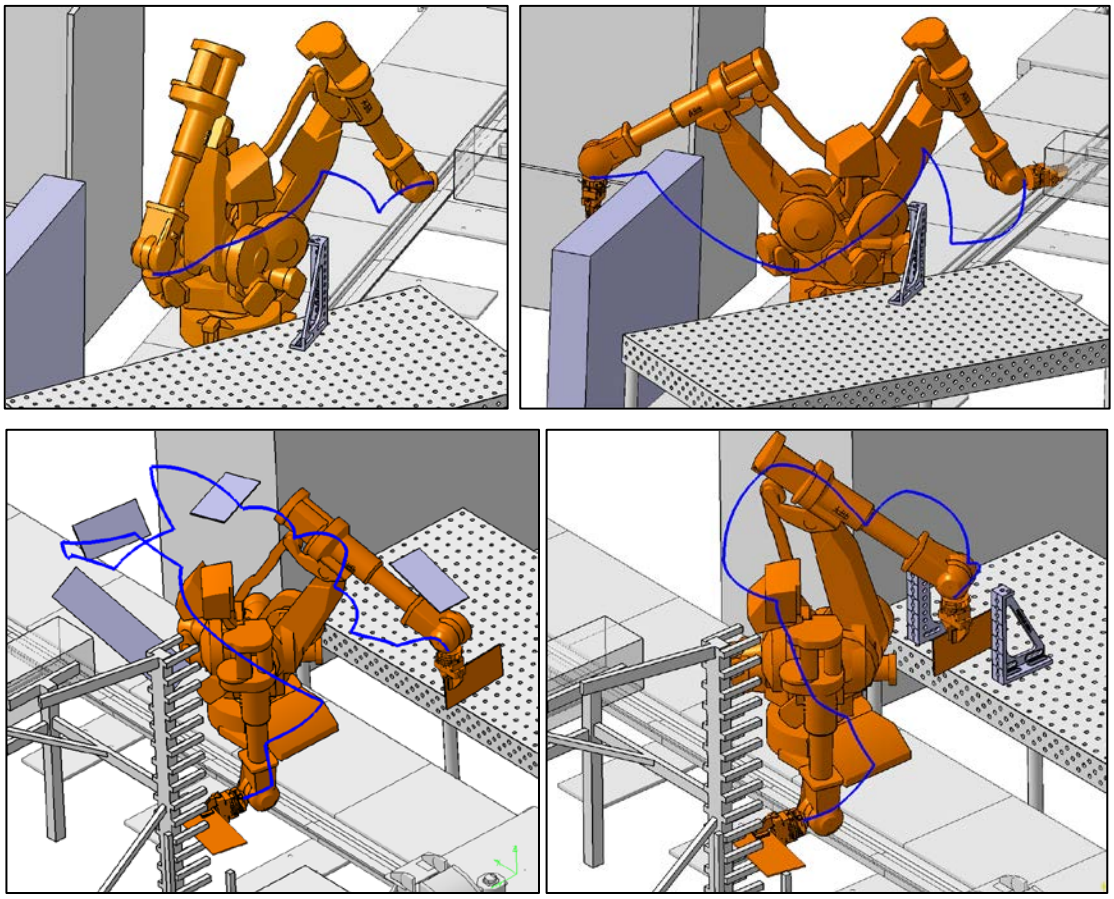


Figure 4-5 Robot test paths 1- 4.
Clockwise from top left, respectively.

4.2.4.2 *Results*

The first test aimed to find an optimal probability distribution to use when randomly selecting robot DOFs to carry out partial shortcuts with. The AWD approach was evaluated alongside several static probability distributions. The original distribution used in [78] was included, along with some other static distributions that were found to perform well over the four test paths.

The results of these tests are shown in Figure 4-6. It was found that the adaptive method performed the best in each of the four paths. Using paths 1 and 2, the average number of collision checks required were reduced by 15% over the best static distribution, and by 32% over the distribution specified in [78]. The AWD approach was found to perform better on the more complex paths 3 and 4, where it used 30% fewer collision checks than its closest competitor and 38% less than the distribution specified in [7878]. When

averaged over the four test paths, the number of collision checks used by the AWD approach lies 1.94 standard deviations below the sample average, further highlighting the general effectiveness of the approach.

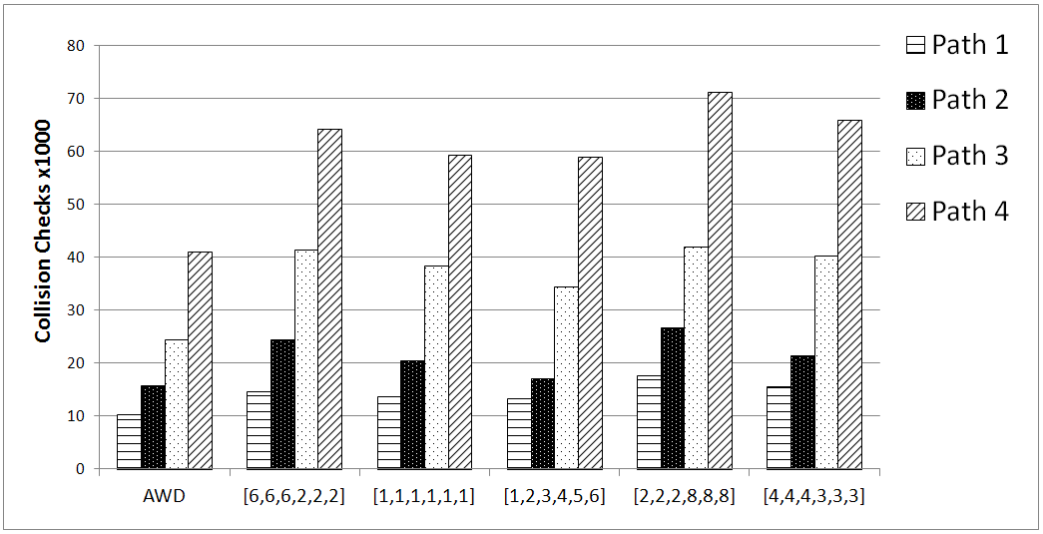


Figure 4-6 Results from weight distribution tests.

The next test was designed to assess the effectiveness of the selectDOF algorithm introduced in Section 4.2.3.2, and also to find the optimal value to use for the input: nDOF. In these tests, the AWD method was used. The results, summarised in Figure 4-7, indicate that randomly selecting either two or three joints per iteration was the most effective. When averaged over the four test paths, selecting Rand(2/3) joints per iteration was found to reduce the number of required collision checks by 24%, 11%, 38%, 15% and 10% over the 1-DOF, 2-DOF, 3-DOF, Rand(1/2) and Rand(1-3) variants. In the simpler problems, which were easier to navigate, the 3-DOF option performed well. However on paths 3 and 4, the 3-DOF approach was outperformed considerably by the random selection variants. This can be attributed to the high rates of failure when attempting to partially shortcut three robot DOFs simultaneously in more complex regions of space.

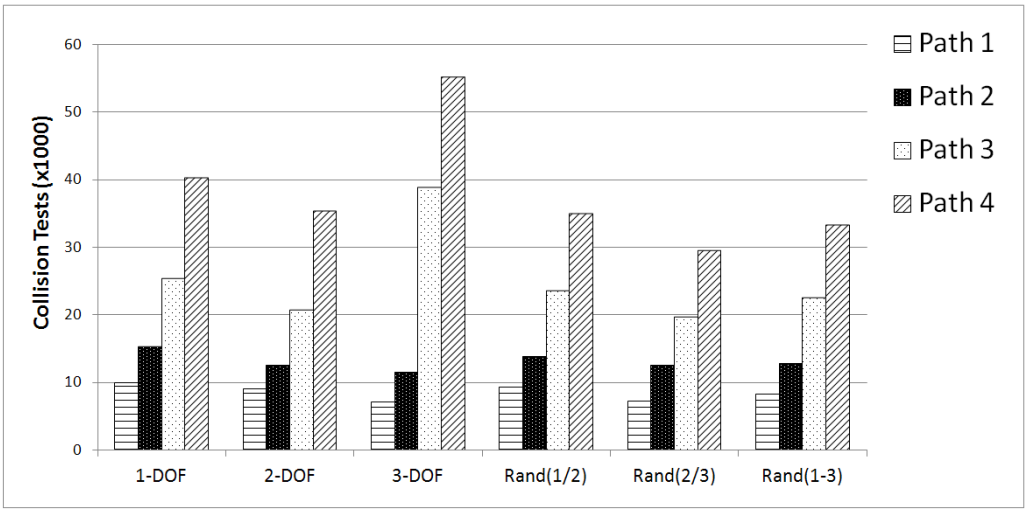


Figure 4-7 Results from the nDOF tests.

The next series of tests assessed the Adaptive P-Sc algorithms use of a check that ensures partial shortcuts are not carried out multiple times, as presented in Section 4.2.3.3. As expected, this check provides an overall benefit to the speed of the P-Sc algorithm. By checking for potential improvement, the number of collision tests used to reach an optimal solution was reduced by 7%, 11%, 23% and 29% for each of the four respective problems. On average, the total number of required collision tests was reduced by 18%. Both P-Sc variants used in this phase of testing utilised AWD, and were also modified to randomly select either two or three joints per iteration. The data from these tests is presented in Figure 4-8.

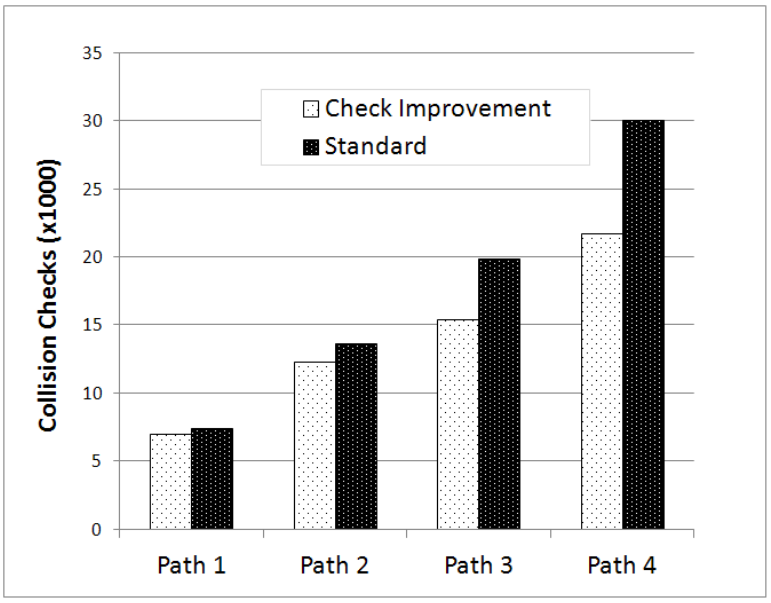


Figure 4-8 Results from checking potential improvement

To determine the overall effectiveness of the Adaptive P-Sc algorithm, its performance is compared to the original P-Sc variant presented in [78]. The configurations of both algorithms and results of the subsequent tests are presented in Table 4-7. Overall the adaptive algorithm performed significantly better than the original variant. On average, it reduced the number of collision checks required to generate an optimal path by 61%. The adaptive algorithm showed improved performance across all tests, and is particularly suited to longer and more complex test paths. The advantage of the adaptive P-Sc approach is shown in Figure 4-9, which details the optimisation process of both variants during one of the optimisation tests carried out on path 3. The figure shows the adaptive approach is more successful at finding valid partial shortcuts than the original variant; far fewer main loop iterations are required to reach the optimal solution.

Table 4-7 Comparison of Adaptive and Original P-Sc Algorithms

	Original P-Sc	Adaptive P-Sc	
Configuration			
Weight	[6,6,6,2,2,2]	AWD	
nDoF/iteration	1	Rand(2/3)	
Check improve?	No	Yes	
Tests			% Reduction
Path 1	14,193	7,127	49.8
Path 2	23,120	12,443	46.2
Path 3	45,961	17,061	63
Path 4	60,527	19,877	67.2
Average	**35,950**	**14,116**	**60.7**

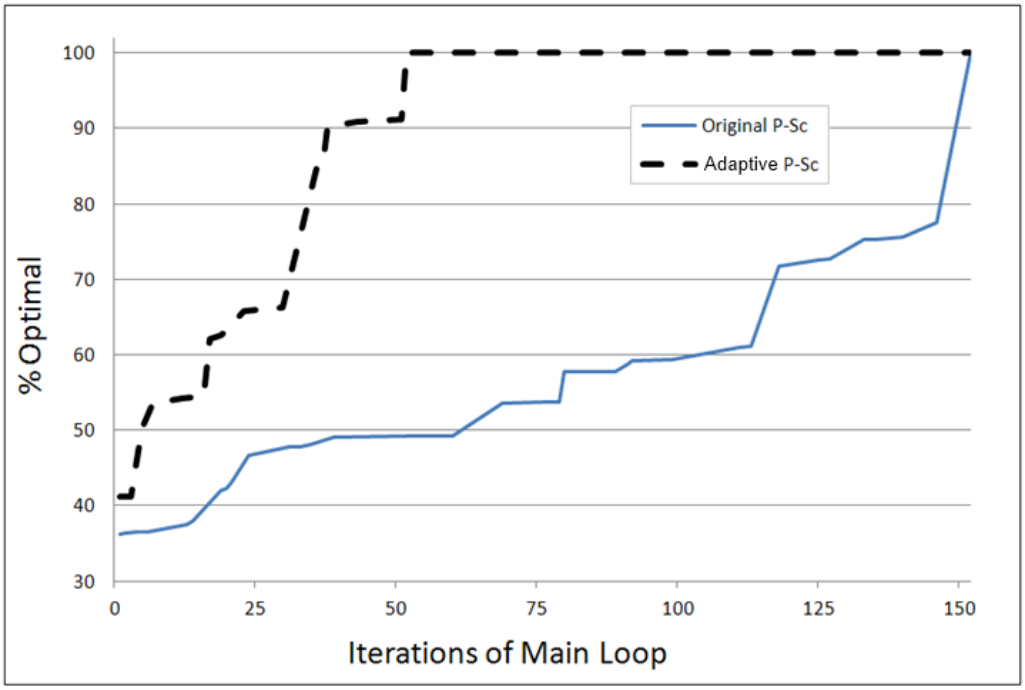
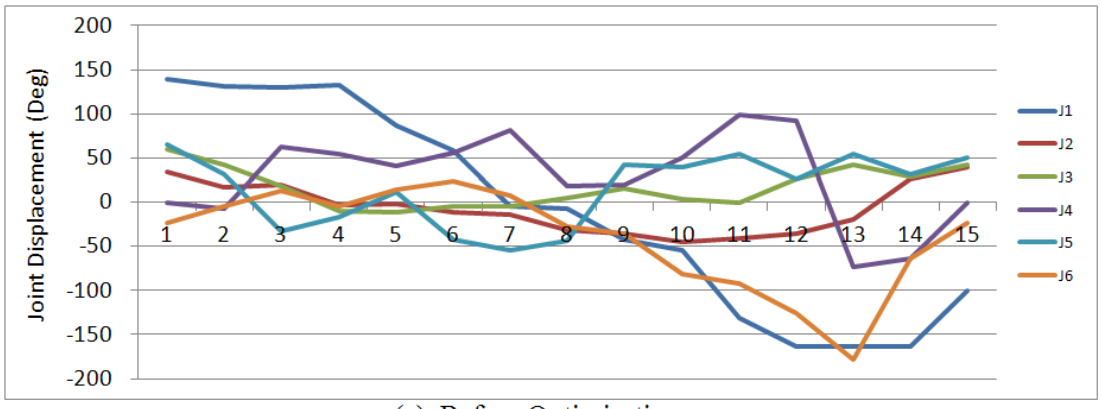


Figure 4-9 Comparison of adaptive and original P-Sc algorithms.

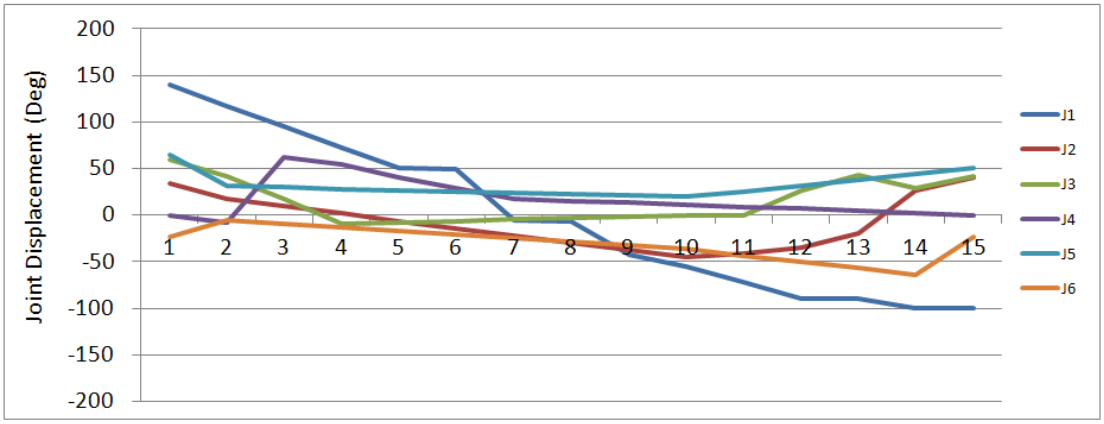
4.2.5 Optimisation Discussion

As a by-product to the way in which it reduces path length, the P-Sc algorithm also reduces redundant joint motion and overshoot. This phenomenon is represented graphically in Figure 4-10, which plots the trajectories of the six individual joints of the

manipulator as it travels path 4, both before and after optimisation is carried out. The optimisation process reduced the number of individual changes in joint trajectory from 90 down to 25, indicating that redundant motion has been removed. Whilst changes in joint trajectory still exist in the optimised path, the degree in the severity of these changes has been significantly reduced, producing a smoother and more aesthetic series of manipulator motions.



(a) Before Optimisation



(b) After Optimisation

Figure 4-10 Joint trajectories for path 4, before and after optimisation.

Another area of importance relates to an effective means of terminating the P-Sc algorithm. Until this stage, all tests have been terminated once the P-Sc algorithm has optimised the sample path to a pre-determined value. It is important to note that in a real world application, this optimal value will not be known, introducing the need for an effective termination protocol for the algorithm. The simplest methods of terminating the optimiser are to end the process after a set number of iterations, or an amount of

time. The drawback to these methods is that they do not take into account the complexity of the path, which can vary widely. Paths that are highly complex may not be optimised enough within the set timeframe. Conversely, simple paths may be completely optimised within a few iterations and the remainder of the P-Sc iterations will be redundant. To address this issue, a specialised termination protocol was devised.

The termination protocol operates by counting consecutive failures of partially interpolated paths. The general approach is to terminate the algorithm if η successive partial shortcuts return a negative result from the collision test. To determine a suitable value of η, consecutive failures of the collision testing algorithm were recorded over 100 tests on each of the four queries. Over the 400 tests carried out, the longest run of consecutive failures encountered before an optimal solution was achieved was nine. By setting η to a conservative value of 12, we are ensuring that the P-Sc algorithm will terminate suitably. Naturally, if significant changes to the robotic system, or its surrounding environment were to occur, this value of η would need to be recalculated.

5 CASE STUDY: IMPLEMENTATION OF AOLP TO A ROBOTIC WELDING SYSTEM

The plight of an industry partner involved in this research provided an opportunity to develop and test the effectiveness of AOLP programming in a real world application.

In order to keep up with increasing demand, our industry partner implemented a robotic welding cell to replace manual welding processes performed during the fabrication of an armoured vehicle's hull. However great difficulty was experienced programming this cell due to the high number of robots involved and the complex nature of the workcell's setup. Using conventional online programming methods, presented in Section 1.1.3, a team of operators and engineers spent over 6000 man-hours programming the system for the entire welding sequence. Compared to the workcell's production cycle time of 16 hours, programming this cell represents a time that is well over 350 times that of the production cycle.

Factors contributing to the overall difficulty associated with the online programming of this robotic cell included:

1. The robotic devices used in the cell are configured in a 13-DOF robot-on-robot-on-rail setup (see Section 5.1.3 below). Jogging the overall robot with this setup is unintuitive as the direction of the joystick is almost always misaligned with the world coordinate system.
2. Programming internal welding seams inside the hull is difficult due to its many confined spaces.
3. The high DOFs of the welding system make it impossible to find an optimal solution manually or through general search algorithm.
4. Programming the welding sequence is made more difficult as three separate tools need to be considered in the programming process (temperature sensor, laser scanner and welding torch).

Due to the difficulties encountered the first time the cell was programmed, the industry partner was hesitant to utilise this robotic welding cell in future manufacturing operations. Even though all robotic hardware has been purchased and set up for use, the programming of the cell has become a major hurdle in its effective utilisation. As a

result, when minor alterations have been made to the hulls design, the subsequent manufacture of these modified sections has not been carried out by the robotic cell. They are, instead, performed manually (i.e. by human) due to the difficulty and high costs associated with modifying or creating new robot programs for the existing robotic system.

Our industry partner is anticipating orders for new variants of the vehicle, which feature significant updates to its design. If the current practice of online programming is used to reprogram the workcell to manufacture these new variants, the industry partner will not be able to afford the significant cost and system downtime incurred the first time the cell was programmed. It is clear that in this situation an alternative and more effective method of programming is needed.

5.1 ROBOTIC WORKCELL SETUP

5.1.1 General Workcell Configuration

Due to the large number of seams and complex geometry (large size and substantially enclosed structure) featured in the design of the vehicle's hull, a specific robotic cell was created to maximise the number of external and internal seams that can be welded. A CAD representation of these robotic welding cells overall layout is shown in Figure 5-1 below. To satisfy cycle time requirements, the final design of the cell includes two identical welding systems. Each welding system consists of a robot-on-robot-on-rail setup, which produces a combined 13-DOF mechanism, as shown in Figure 5-2.

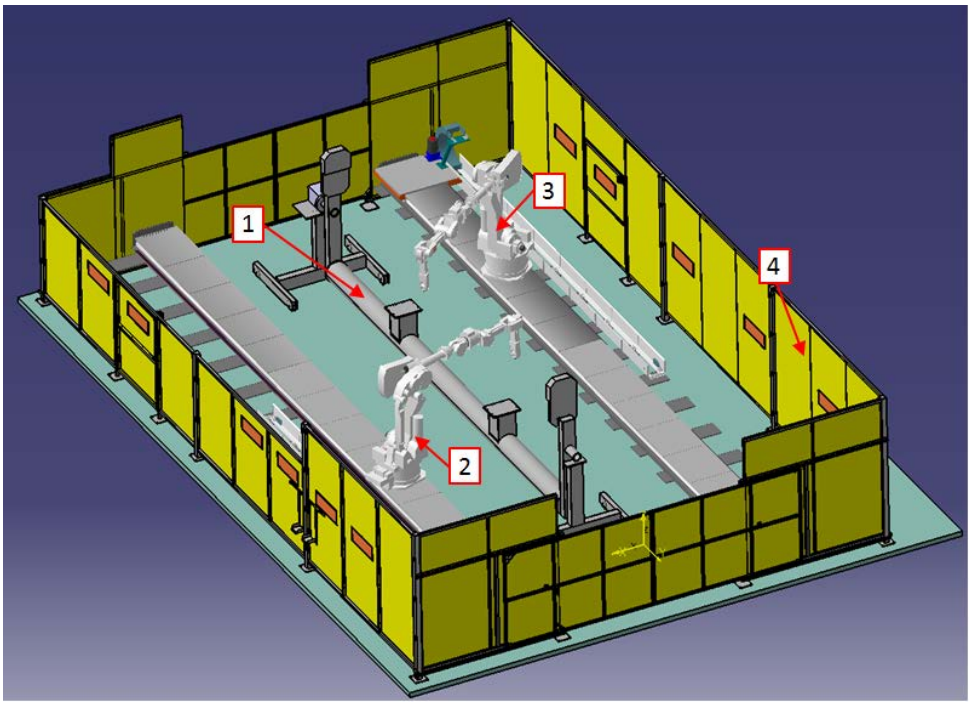


Figure 5-1 CAD model of the Robotic Welding Cell

Detail: (1) Rotating trunnion (hull omitted). (2) Left side13-DOF welding system. (3) Right side 13-DOF welding system. (4) Safety fencing.

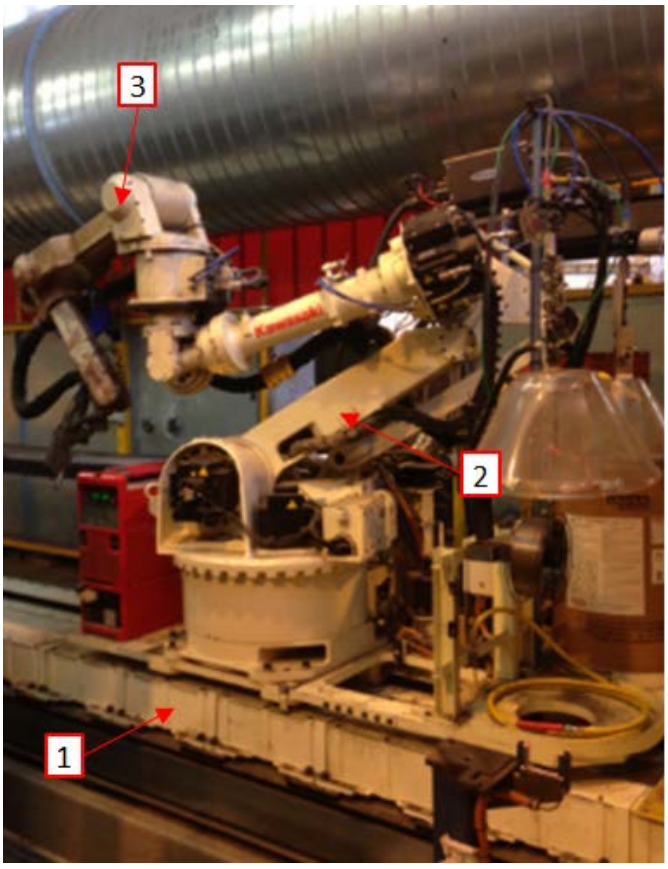


Figure 5-2 One of the combined 13-DOF welding systems

Detail: (1) Linear rail. (2) Auxiliary robot. (3) Weld robot.

5.1.2 Work-Piece and Trunnion

The vehicle hull is mounted on a rotating trunnion, shown in Figure 5-3. This allows the welding robot to maintain a down-hand position when welding, and also assists in providing internal access for the robotic welding system through openings in the hulls structure such as the windscreen or bonnet.

Before the hull is mounted on the trunnion, its structure is manually tack welded together, as shown in Figure 5-4, with the aid of large-scale jigging. These tack welds are also used as reference points by the laser scanning calibration system to regenerate accurate coordinates for a seam before welding processes are carried out (refer to Section 5.2.2 below). In all, there are 260 individual seams that are welded by the robotic system, and 16 different trunnion orientations are used during this process.

The vehicle hull has weld seams located both internally and externally. There are five principal points of access to the inside of the hull:

- A rectangular rear door opening
- A circular turret port on the roof
- Rectangular windscreen access
- Rectangular bonnet opening
- Square front grill opening

Figure 5-3 Rotating trunnion used to position the vehicle hull.
Trunnion rotates clockwise to position vehicle hull for the welding process (Vehicle hull edited out).

Figure 5-4 Weld tacks used to pre-assemble hull.

5.1.3 Robotic Devices

As mentioned in Section 5.1.1, there are two 13-DOF welding systems in the manufacturing cell. Each of these systems consists of a 6-DOF Kawasaki FA10N welding robot which is mounted directly to a 6-DOF Kawasaki ZX-300S auxiliary robot. This combined robot-on-robot set up is then mounted on a 12-meter long linear

rail in order to allow it to reach both the front and rear sections of the vehicle hull. To carry out a weld seam, the rail and auxiliary robot are used to position the weld robot nearby. These two robots are then held still whilst the weld robot carries out the weld process, as shown in Figure 5-5 below.

Figure 5-5 The robotic mechanism performing a welding operation on the vehicle hull
The auxiliary robot holds the weld robot in place whilst it performs the weld process (Vehicle hull edited out).

5.1.4 Weld and Calibration Equipment

The welding systems used in this robotic system consist of a Fronius weld torch and power supply, along with their associated wire feeding mechanisms. There are two welding systems used, one mounted on each of the welding robots. Mounted on the weld torch is the laser scanner used for calibration purposes (Section 5.2.2), and an infra-red temperature sensor used to check that pre-heat temperatures are suitable before carrying out a weld. The combined torch and scanner is shown in Figure 5-6.

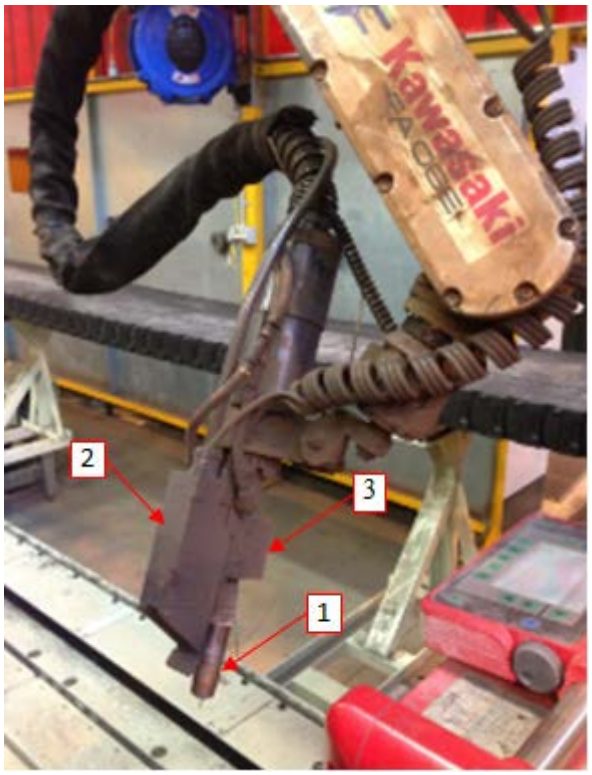


Figure 5-6 Weld torch and scanner mounted on the weld robot.
Detail: (1) Weld torch. (2) Laser scanner. (3) Temperature sensor.

5.1.5 Miscellaneous Components

The robot-on-robot set up used in this workcell requires a state of the art communication system to ensure each robot is interfaced and can communicate their current positions and activities to one another. The PLC and computer system which manages this communication is housed in an external cabinet, shown in Figure 5-7 below. The cabinet is equipped with a computer monitor which is mounted on its front. This serves as an interface for users to monitor and control data relating to weld sequence, welding processes, signal communications, trunnion orientation etc. The interface is also used for checking the quality of a laser scans, as shown in Figure 5-8, which is particularly useful during the testing phase outlined in Section 5.7.2.

Figure 5-7 PLC cabinet

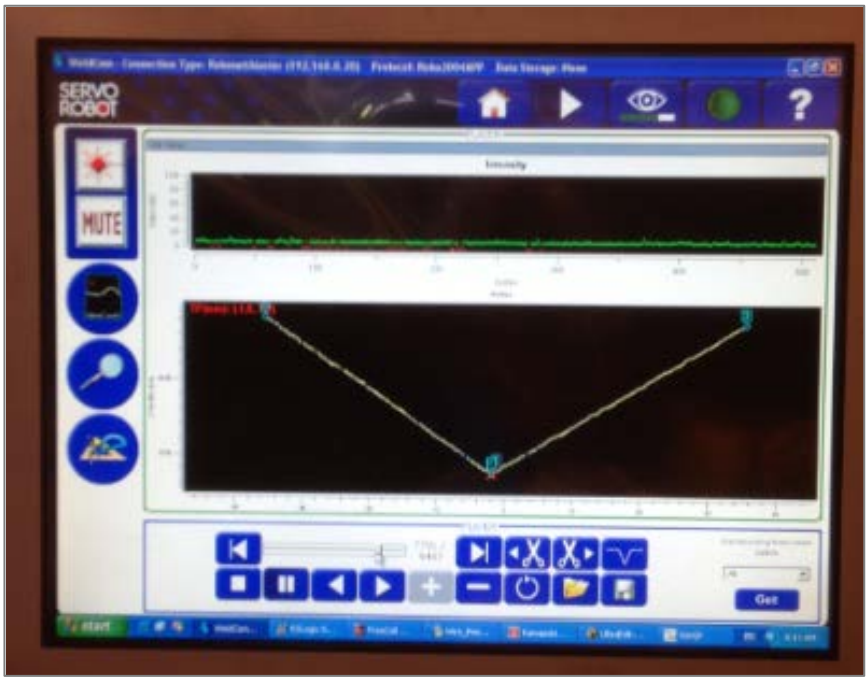


Figure 5-8 WeldCom laser scanning interface

5.2 PROCESS REQUIREMENTS

5.2.1 Robotic Welding Process

In order to program the robotic system to carry out a weld seam, one must understand all the operations that need to be performed by the robots, and the sequence in which they are to be carried out. In this application, this process was simplified somewhat as the robotic welding cell already had weld seams programmed into it in the past. Although the method used to program the cell in the past is fundamentally different to the AOLP approach to programming (the robotic cell was previously programmed with online methods, AOLP utilises an offline approach), the set and sequence of operations that the robotic welding cell will perform essentially remains the same. A brief summary of the operations the robotic cell performs to carry out a weld seam can be summarised as:

- *Orientate hull*. The first action is performed by the workcell's trunnion. It rotates to position the hull so that the seam that is to be welded is orientated in a 'down-hand' configuration that is optimal for welding.
- *Preheat*. The preheating system ignites its blow-torch and preheats the plates making up the seam that is to be welded.
- *Auxiliary robot motion*. After preheating, the auxiliary robot then moves to position the weld robot at a suitable location so it can perform all of its welding operations. If the seam is located in the hulls internal structure, the auxiliary robot will have to carry the weld robot through one of the hulls entry points (Section 5.1.2). Due to the low accuracy of the auxiliary robot/rail, during the welding of most seams (>99%) the auxiliary robot/rail is held stationary. Only the welding robot provides the motions for the welding.
- *Temperature check.* Before any welding is performed, the weld robot uses a temperature sensor mounted on the weld torch to ensure that the temperature of the plates is sufficient enough to begin welding.
- *Scanning and calibration.* The robot utilises the laser scanner mounted on the weld torch to scan the local geometry that makes up the weld seam (refer to

Section 5.2.2 below). This information is then used to re-generate a calibrated set of coordinates for the seam.

- *Welding*. After calibrated coordinates for the have been generated, the welding robot can then move to carry out its welding processes. A large portion of the seams that make up the hull involve multiple passes.
- *Auxiliary robot motion*. Once the weld seam has been completed, the auxiliary robot then performs a series of motions to remove the welding robot from the vehicle hull, back to its home position. If another seam is to be welded nearby the auxiliary robot will move the weld robot directly to a position so it can carry it out instead.

5.2.2 Laser Calibration Procedure

In Section 5.1.4 the existing robotic manufacturing cell's laser calibration system is presented. This calibration system is necessary, as it is used to make up for the geometric variances between each vehicle hull input into the manufacturing system. These variances are a result of the inaccuracies that occur from the manual tacking of the hulls structure, and the poor levels of repeatability that the trunnion has when positioning the hull at its pre-set orientations. Before the weld robot carries out a particular weld, it scans the plates in the locally vicinity of the seam in order to re-generate a it's calibrated set of coordinates. This ensures that the robot can accurately carry out the welding process, regardless of variances in the hulls geometric structure or positioning.

Local calibration is also necessary to make up for additional deviation between the actual geometry of elements in the work cell, such as the workpiece, and the nominal geometry of the CAD environment. Preliminary tests showed that, for this specific application, the deviation can be up to 20mm. The existing laser scanner used in this system, however, is more than suitable to locally regenerate the coordinates of the seam, allowing the robot to accurately carry out its welding processes.

It must be noted that the orientation of the weld torch when it is carrying out the weld process is not modified by this calibration process. Only the weld seams start, end and

mid-points are updated by the laser scanning system. With the goal of generating the real world [X,Y,Z] coordinates of the seam's start and end positions, the way in which the scanning process is carried out depends on the configuration of the plates that surround the seam, as shown in Figure 5-9 below.

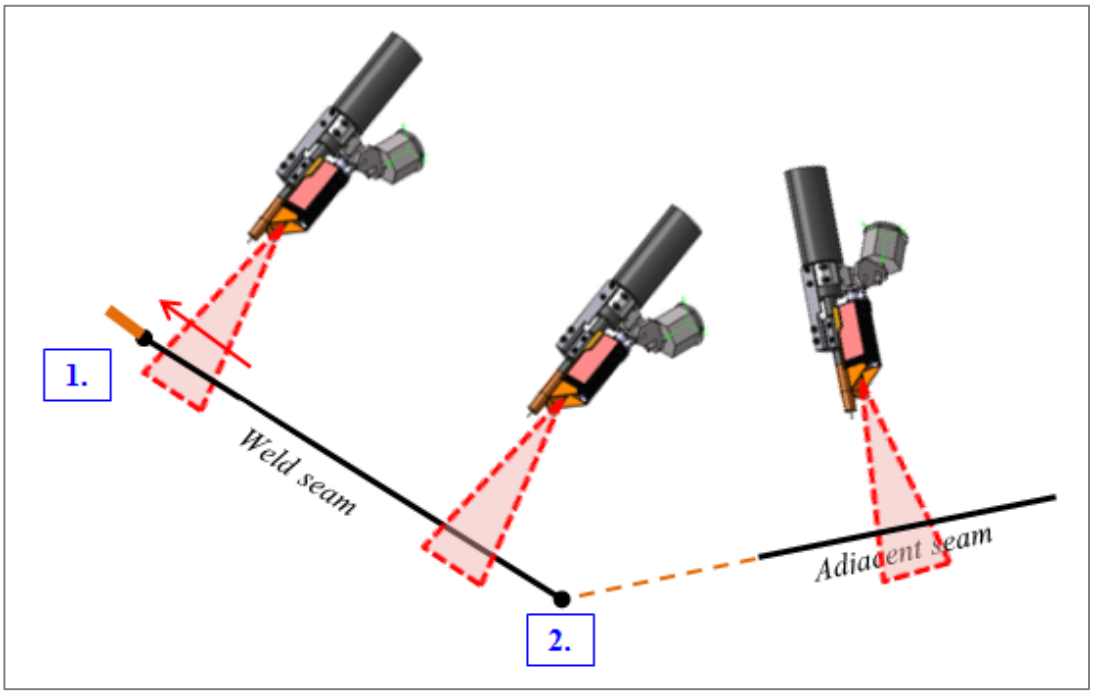


Figure 5-9. Two different scanning processes to regenerate calibrated seam start/end locations.

In situation 1 of Figure 5-9, the end of the weld seam is marked by a physical weld tack, which is deposited during the initial tack-up assembly of the hull. To capture the position of this tack, and generate an accurate position for the end of the seam, the robot positions the scanner ~80mm from the tack. The scanner then sweeps towards the tack, and the robot accurately records it's coordinate once it is reached. The other method of scanning, depicted in situation 2 of Figure 5-9, is required if the weld seam starts or ends in a corner. In these situations, the close proximity of the plates often restricts the scanners access, preventing a method similar to the one in situation 1 from being used. To overcome this, another edge (formed by two plates) that intersects with the weld seam is scanned, and a vector is generated. The coordinate of the start/end point of the seam is then generated from the intersection of this vector and the weld seam.

5.3 EXCEL INTERFACE

An Excel spreadsheet is used to provide an interface for the user to enter various setup information used by the AOLP algorithms. The spreadsheet is divided into six tabs, a brief summary of each tab is provided below. A more detailed description of each worksheet, including examples of layout, can be found in Appendix Section 7.1.

- **Trunnion.** In this tab the user inputs data relating to the positioning and rotation capability of the trunnion that the vehicle hull is mounted upon.
- **Sphere Definition**. Provides information about the bounding sphere models used to represent the robots in the workcell. Sphere models are read from this data, and are then used by the AOLP algorithms (Section 5.5.1 below).
- **Clash Objects**. Provides information about the simplified bounding box CAD models used to represent the environment and workpiece in the robotic workcell (Section 5.5.2 below).
- **Seams**. This tab provides the nominal coordinates of each seam that is to be welded by the robotic system. This data is input into the spreadsheet via the use of a Visual Basic script, described in detail in Section 5.5.3 below.
- **Zone**. Specifies the regions of space used to simplify and solve the robot placement problem (Section 5.6.3)

5.4 GRAPHICAL USER INTERFACES

Two separate graphical user interfaces (GUIs) where developed to allow a user to control the operation of the AOLP algorithms. The first GUI, developed in the Visual Basic programming language, is used entirely for the Setup phase of the AOLP process. The other GUI, developed in MATLAB, is used exclusively for the programming phase.

5.4.1 Visual Basic Graphical User Interface

The Visual Basic GUI, depicted in Figure 5-10 below, is primarily used during the setup phase of the AOLP process. The Visual Basic programs called from this GUI

communicate directly with DELMIA simulation software, and with the Microsoft Excel spreadsheet presented in Section 5.3. The GUI is used for four separate tasks:

- Defining the sphere-bound CAD models of the robotic devices and tooling (Section 5.5.1).
- Defining the bounding box CAD models of the workcell environment (Section 5.5.2).
- Generating robot targets that represent the seams that are to be welded (Section 5.5.3).
- Uploading robot code files generated by the AOLP system into DELMIA for simulation (Section 5.6.8).

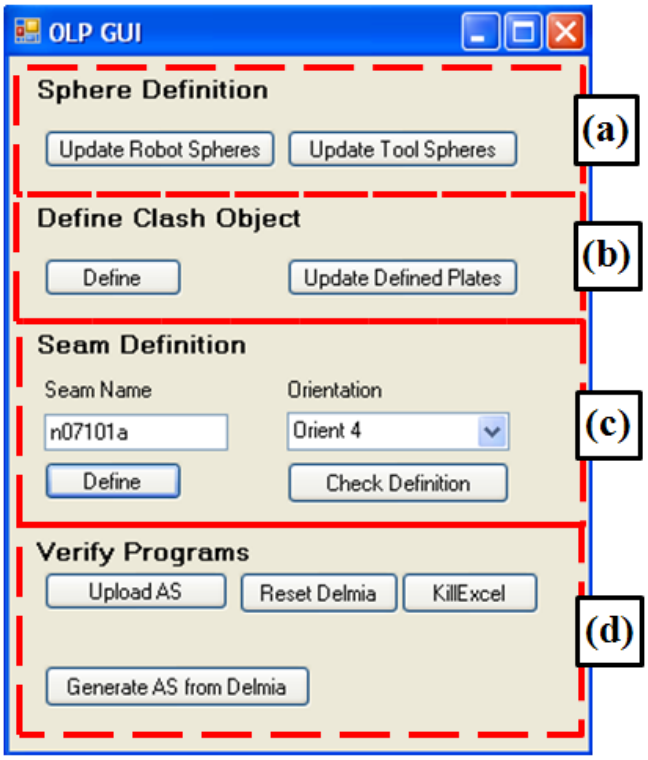


Figure 5-10 Visual Basic GUI:
(a) Robot Sphere Visualisation. (b) Clash Object Definition. (c) Seam Generation. (d) Simulation for verification.

5.4.2 MATLAB Graphical User Interface

The MATLAB GUI, shown in Figure 5-11 below, is used to control the AOLP algorithms that carry out the programming process for the robotic cell. It is configured with several panels, each one dedicated to a particular aspect of the OLP process. The main programming work is carried out over the four panels shown in Figure 5-11, below, whilst simulation and code generation processes are carried out in supplementary panels shown in Figure 5-12.

- Panel (a) of the GUI depicted in Figure 5-11 is dedicated to the generation of tool placements. The user selects the seam to be programmed, and is able to generate a suitable tool path for the weld torch to follow in order to carry out the weld.
- Panel (b) is used for configuring additional calibration scans (Section 5.6.1) and also for compiling all generated tool paths into a complete path for the robot to follow.
- Panel (c) is used for the robot placement problem (Section 5.6.3), and also as a display for many of the results obtained up until that stage of the programming process.
- Panel (d) is used to control robot motion planning algorithms (Sections 5.6.5 and 5.6.6) which link the individual elements of the weld process together.

Two additional supplementary panels are included in the MATLAB GUI, as shown in Figure 5-12 below.

- Panel (a) is used to convert all generated motions into a text file that can be uploaded into the real world robotic system for use (Section 5.6.7). Code variables used to designate job numbers and weld settings can be altered in this panel, and they are inserted into the generated code accordingly.
- Panel (b) is used to simulate various motions generated by the AOLP algorithms (Section 5.6.8).

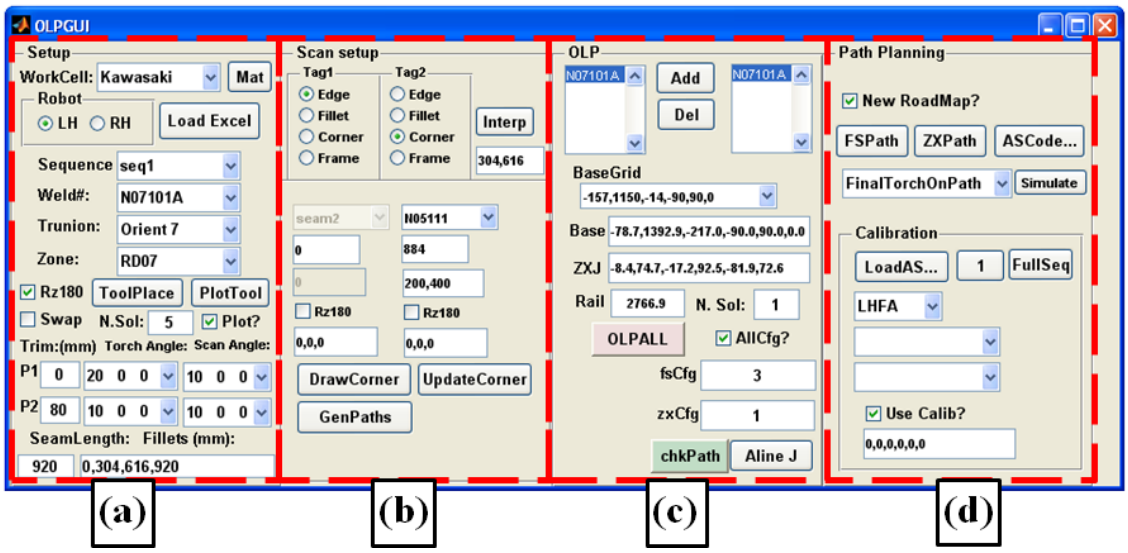


Figure 5-11 MATLAB GUI

(a) Tool Placements. (b) Scan setup and path generation. (c) Robot placement. (d) Motion planning

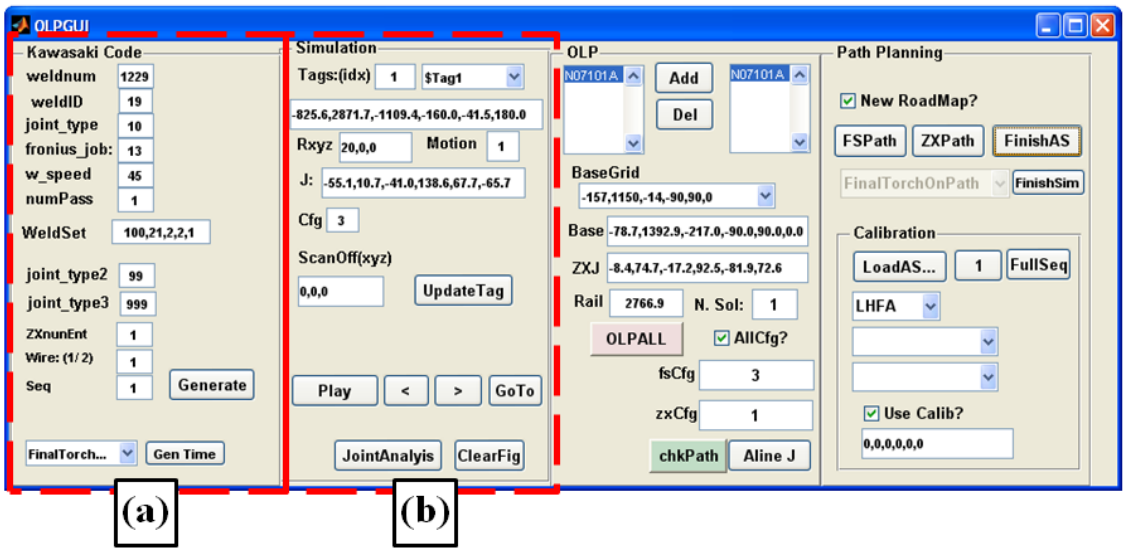


Figure 5-12 MATLAB GUI supplementary panels

(a) Code generation. (b) Simulation.

5.5 SETUP PHASE

The setup phase only needs to be performed once, before any of the seams are programmed. This phase consists of configuring three key elements that are used by algorithmic processes in the AOLP system:

- Generating and configuring CAD representations of the robots (Section 1.1.5) and workcell (Section 5.5.2).
- Generating robot targets (Section 5.5.3).
- Defining zones to assist in the robot placement process (Section 5.5.4).

5.5.1 CAD Representation of Robots and Tooling

Simplified sphere-bounding CAD models, presented in Section 3.2.1 (page 66), are used to represent robots and tooling in the AOLP system. These models are defined by the user in the Excel spreadsheet, where the Cartesian location and size of each sphere can be specified in the 'Sphere Definition' tab (refer to Appendix Section 7.1). This tab features three separate sections, allowing the user to define sphere models for the robotic system's weld robot, auxiliary robot and weld torch.

In order to validate the model specified by the user, a Visual Basic program was created. This script reads the user defined sphere data, and then interfaces with DELMIA in order to render the corresponding set of spheres. The rendered model is then superimposed over a full detail CAD representation of the robot, as shown in Figure 3-3 and Figure 3-4 (page 69), so that the user can readily assess the suitability of the simplified CAD model. The user is able to control this script via the Visual Basic GUI, shown in Figure 5-10 (page 130).

Once the user is satisfied with the generated models, the algorithms in the AOLP system can then use this Excel data, along with its own D-H kinematic models to generate a geometrically simplified representation that is suitable for the efficient processing of collision.

5.5.2 CAD Representation of Workpiece and Environment

In similar fashion to the simplified sphere-bounding models used to represent the robotic elements, CAD representation of the workpiece and surrounding environment are also simplified in order to speed up associated collision processing algorithms. To do this, the objects are represented with simplified bounding-box models. This task is performed by extracting the necessary geometric elements from the full-representation

CAD model of the workpiece, and then storing it in the Excel spreadsheet. This is carried out in a semi-automatic manner with the assistance of Visual Basic scripts that interact with the DELMIA software environment. These scripts are called from the Visual Basic GUI shown in Figure 5-10 (page 130). Once initiated, a full detail CAD model of the work object is loaded into DELMIA, and the user is able to efficiently gather relevant data used to generate the simplified bounding box CAD representations, as shown in Figure 5-13 below. Work objects are typically defined with a number of these bounding box representations; the general method for this is presented in Section 3.2.1 (page 66).

Once the user has generated enough of these bounding box models to sufficiently represent the workpeice, the Visual Basic script then pastes the collated data for all surfaces, normal directions, thicknesses and other items such as the plates name etc. into the Excel spreadsheet. When the AOLP system is initialised, MATLAB accesses this spreadsheet and gathers the bounding box data for each plate. This data is then used to construct geometric surfaces that can be used with the robot-sphere data for collision checking purposes.

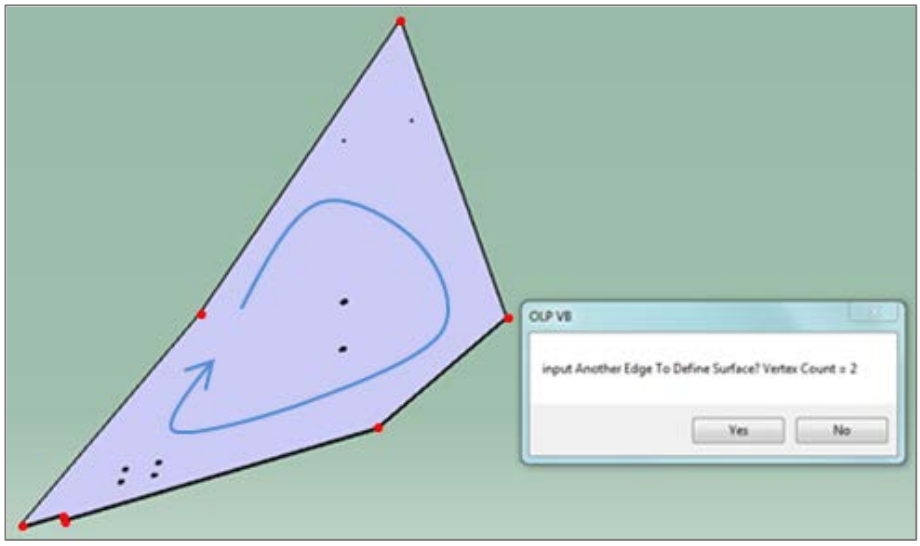


Figure 5-13 Bounding box representations.
The user is prompted to select plate vertexes (red dots) in the DELMIA environment.

5.5.3 Define Robot Targets

Another important component of the setup phase involves generating robot targets that can represent weld seams about the vehicle hull. The nominal geometry of each seam is defined by two tags, which specify the start and end locations of the seam, along with an initial orientation for the weld torch. This data is stored in the Excel spreadsheet, which is collated with a Visual Basic program in similar fashion to the bounding-box CAD models are generated in Section 5.5.3 above.

When the user clicks the 'define seam' button on the Visual Basic GUI in Figure 5-10 (page 130), the user is prompted to select the edge that represents a weld seam found at the intersection of two plates. The XYZ data of the start and end point of the seam is then extracted. To define a nominal TCP angle for the torch, the user is prompted to select two faces that intersect the seam. Using the normal directions of these two faces, a CAD model of the torch is placed at the seam's start and end points, and the user is queried whether this configuration is suitable, as shown in Figure 5-14. The user can rapidly toggle through a number of suitable torch angles until a desirable one is found. The final data is stored in the Excel spreadsheet. The algorithmic details explaining how the nominal torch orientations are generated are presented in more detail in Section 7.3.5.

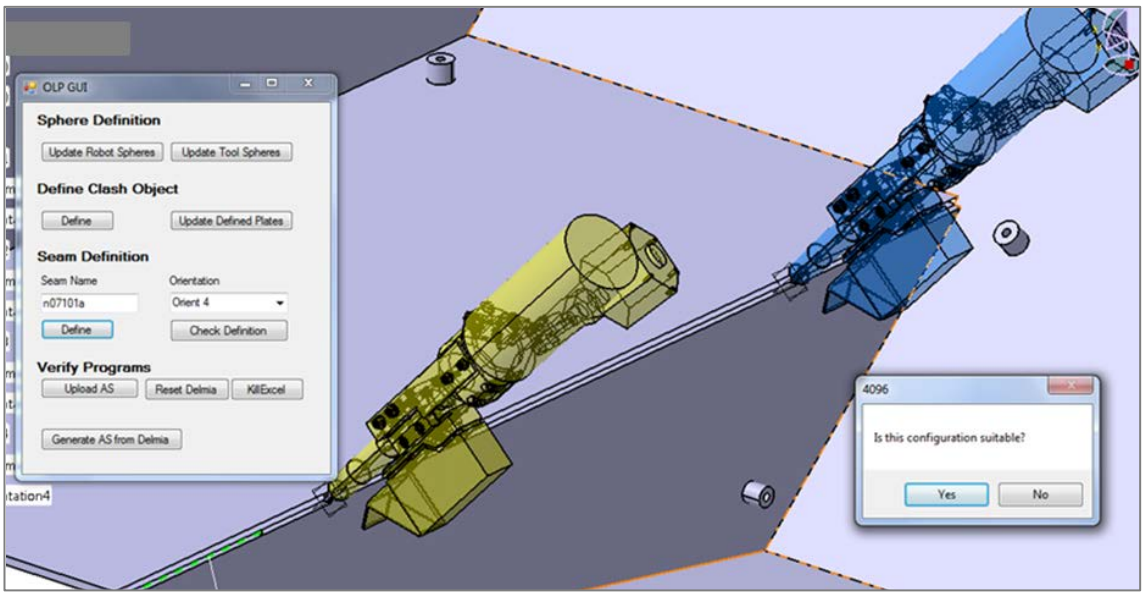


Figure 5-14 Using Visual Basic and DELMIA to define weld seams about the vehicle hull.

5.5.4 Define Zones

In order to simplify the robot placement problem (Section 3.3.2), a number of regions of space (called zones) must be defined by the user. These zones are then used by AOLP algorithms to restrict the search for locations to position the weld robot so that it can carry out its welding processes, as shown in Figure 5-21 below (page 144). The zones are defined manually by the user in the 'Zones' tab of the Excel spreadsheet (refer to Appendix Section 7.1). Each seam that is to be welded by the robotic system is assigned one of these zones. Each zone is defined in Excel with the following data:

- The 3D coordinates of the upper left corner of the zone
- The 3D coordinates of the bottom right corner of the zone
- The number of tags contained within the zone
- The orientation of the tags within the zone

5.5.5 Trunnion Orientations

Each seam that is to be welded by the robotic system has a specific trunnion orientation associated with it. These orientations are used properly position the vehicle hull so that the seam is situated optimally for the welding process. The number of individual trunnion orientations, as well as the angular displacement of each orientation is specified in the 'Trunnion' tab of the Excel spreadsheet. This data is read and stored by the MATLAB program at the start of the programming phase for use with various AOLP algorithms. This data is entered into the spreadsheet manually, more information about this part of the setup phase can be found in Appendix Section 7.1.

5.6 PROGRAMMING PHASE

Once the setup phase is complete, the programming phase can commence. The programming phase is carried out with the aid of the MATLAB GUI introduced in Section 5.4.2, and can be broken down into twelve tasks, each presented individually over the remainder of Section 5.6 below.

5.6.1 Step 1: Load Setup Data from Excel.

This step is carried out before any seams are programmed and, provided that no changes are made to the Excel spreadsheet, only needs to be performed once. The purpose of this step is to load the data contained in the spreadsheet, which was generated during the setup phase, into the MATLAB programs data structures. To do this, the user simply clicks the 'Load Excel' button located in the setup panel of the MATLAB GUI (Figure 5-15). Once clicked, the system accesses the spreadsheet and uploads the following data:

- The CAD models of the robots and surrounding environment (Sections 5.5.1 and 5.5.2).
- The nominal geometry of each seam in the vehicle hull (Section 5.5.3).
- The zones used for placing the weld robot (Section 5.5.4).
- A list of specified orientations that the trunnion can make (Section 5.5.5).

5.6.2 Step 2: Seam Selection

In this step, the seam to be programmed is selected using the setup panel of the MATLAB GUI, as shown in Figure 5-15 below. All seams that exist are pre-loaded into drop down menus on the GUI, and the user simply selects from this list the seam they wish to program. In this example, the seam 'N07101A' is selected. Once done, data read from the Excel spreadsheet relating to the required trunnion orientation and entry zone for the seam are automatically displayed. The seam's nominal length is also shown for convenience.

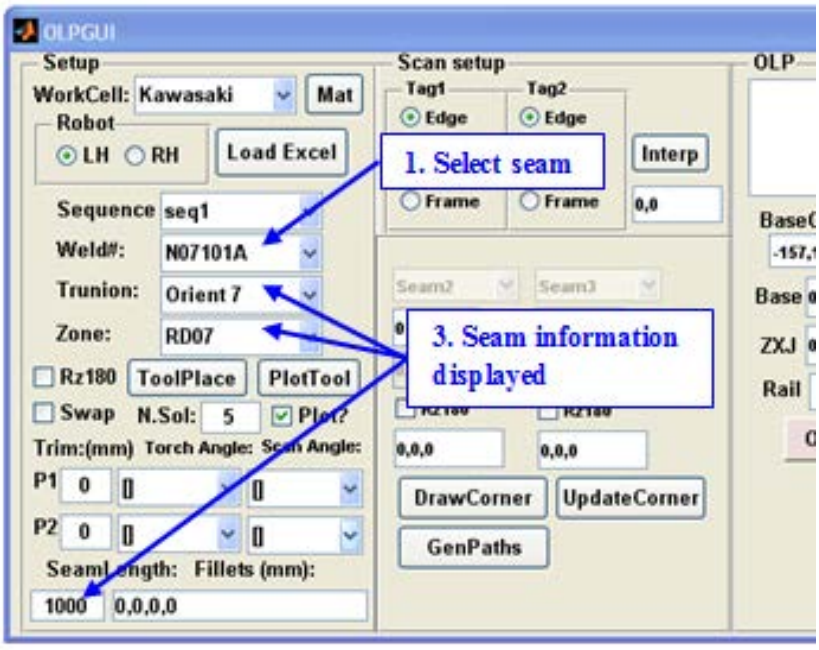


Figure 5-15 Step 2: Selecting the Seam

5.6.3 Step 3: Tool Path Generation

This step involves generating a path for the tool to follow so that it can carry out its welding operations for the seam. The two important aspects of this step are to ensure that the tool orientations along this path will allow for a defect free weld, and that the tool does not collide with the vehicle hull as it follows the specified path. The final path created from this step is used as a target for the robot placement problem, which is addressed during step 7 of the programming phase (Section 5.6.3, page 143).

The 'ToolPlace' button on the setup panel, shown in Figure 5-16 below, automatically attaches a collision model of the torch to the tags that define the start and end of the selected seam. A process of collision checking is then performed, in which a range of pre-defined tool orientations are tested for collision with the local geometry surrounding the weld seam. The torch orientations found to be collision-free are displayed in a drop down menu, allowing the user to select a suitable torch orientation for the start and end of the seam. If ideal torch orientations cannot be found, the user is able to trim each end of the seam to allow the torch more room to move without colliding with nearby plates. This process is also performed for the scan positions for the seam. These nominal scan positions are generated automatically, and the user is also able to alter these scan

positions in the 'fillets' edit bar if necessary. When performing collision testing with the scanner, the same torch collision model is used, as the scanner is included on the welding torch. A different TCP is utilised, however, in order to reflect the lasers optimal field of vision. Once suitable torch orientations have been determined by the user, they are stored in a 'tag' format. Two weld tags are generated, one each for the start and end points of the seam, and are defined by the Cartesian location of the TCP, along with the Euler angles that specify the tools orientation ([X, Y, Z, R_x, R_y, R_z]). Also generated are a number of tags representing the scan locations above the weld seam.

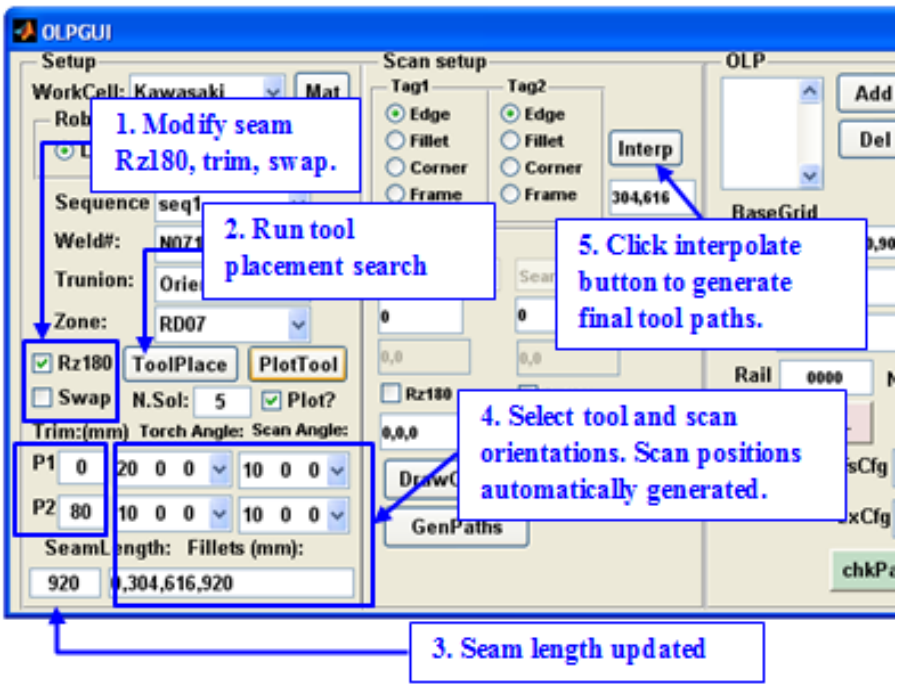


Figure 5-16 Step 3: Tool placement

To correctly position and orientate the weld torch with the seam, the user is able to control various aspects through inputs relating to:

- **'Rz180'**, a Boolean checkbox which specifies if the tool needs to be rotated around its Z direction 180˚.
- **'Swap'**, a Boolean checkbox that controls which end of the seam to start from. Checking this box will swap the order of the seams from its nominal sequence.
- **'Trim'**, specifies how far each end of the seam is to be trimmed. This provides an option when welding the entire length of the seam is not possible (for

example, when welding into a corner where the geometry provides a tight fit for the torch).

- **'N.Sol'**, which specifies the maximum number of valid solutions that the tool placement algorithm is to search for. The default number is 10. This is used as an early termination protocol, which commands the tool placement algorithms to terminate once the specified number of valid solutions has been found.

These inputs can be tweaked by the user to ensure the tool is positioned optimally for defect free weld-seams. Normally, we would like to keep the push angle (R_x) at 20˚ and keep other angles (R_y, R_z) at 0˚. However in some situations these values must be altered as the torch will not fit in some constricted areas.

After torch placement, the user can plot the selected tool orientation together with the vehicle hull to visualise the setup. This is done with the 'Plot Tool' button in Figure 5-16. The resultant figures are displayed in Figure 5-17.

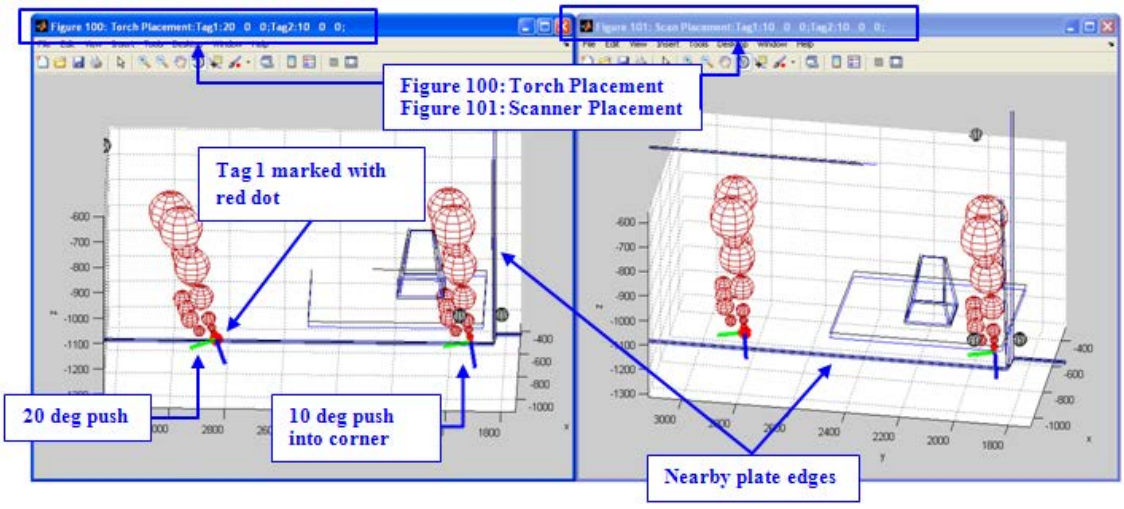


Figure 5-17 Tool placement plots. Left: Welding Torch. Right: Laser Scanner.

5.6.4 Step 4: Interpolate

This step interpolates a number of additional tags between the seam's start and end tags generated in step 3. Two paths are generated during this step: A weld path and a scan path. After clicking the 'Interp' button, shown in Figure 5-16 above, the weld path is automatically generated by linearly interpolating additional tags, every 30mm, between

the start and end of the seam. A collision check is then performed with the tool placed at each of these tags, and the generated weld path is not accepted by the system until this collision test is passed.

The complete weld and scan paths can then be viewed by, once again, clicking the "PlotTool' button used in step 2. The resultant plots are shown in Figure 5-18 below.

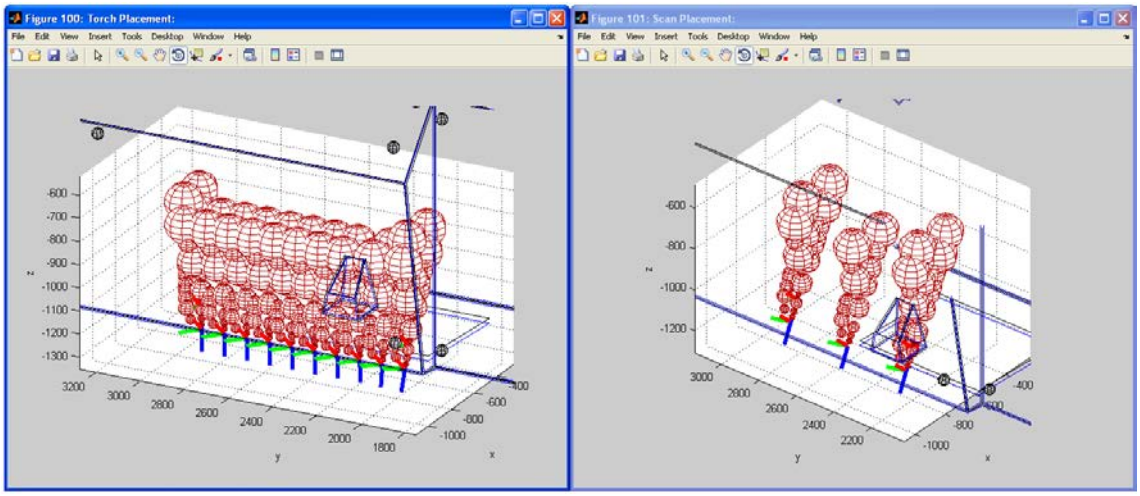


Figure 5-18 Tool plots after interpolation.
Left: Complete weld path. Right: The complete scan path for the seam.

5.6.1 Step 5: Draw Corner (Optional)

Step 5 is used when a corner scan is required to accurately capture a start or end point of the seam (refer to Section 5.2.2). Using the corner scan panel shown in Figure 5-19, the user first clicks the 'Draw Corner' button, which generates a plot of the weld seam and the collection of seams that surround it, as shown in Figure 5-20. A list of nearby seams is then uploaded into the dropdown menu in the corner scan panel, and the user is prompted to use the plot to assist in selecting the appropriate seam for the corner scan. The user can then enter or alter the settings for the corner scan, including the positions where scans are to take place, and the [R_x, R_y, R_z] Euler orientations for the scans. Clicking the 'Update Corner' button will then re-generate the specified scan path with the updated settings, and also generate a plot so the user can check their suitability, as shown in Figure 5-20. If the user is unhappy with the current set up, or the specified scan positions result in the tool colliding with the workpiece, the user can simply modify these scan settings and regenerate the plots.

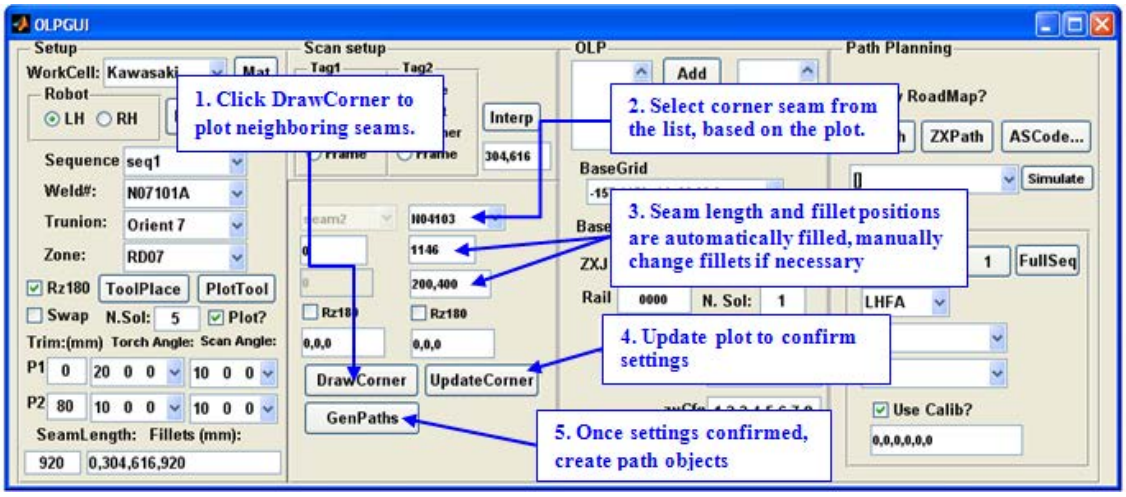


Figure 5-19 Configuring corner scans

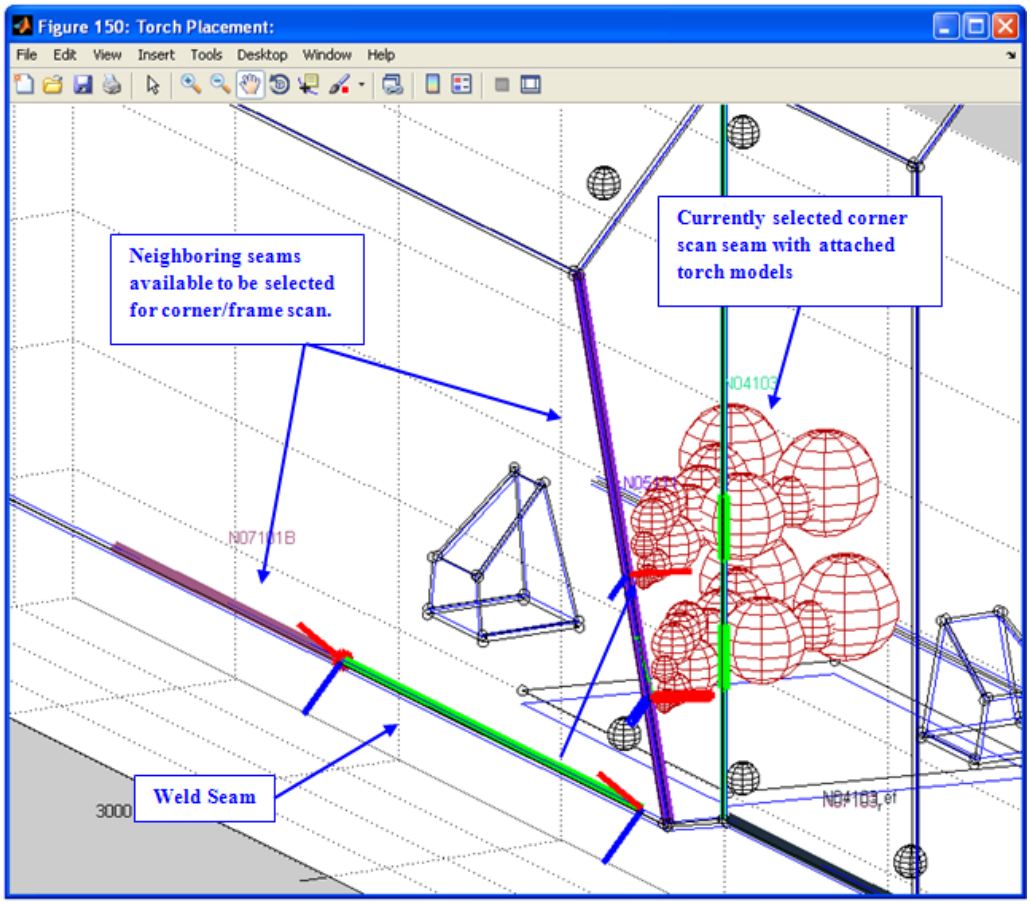


Figure 5-20 Plot of corner scan setup

5.6.2 Step 6: Generate Paths

This step serves to collate all the robot paths related to welding and scanning together so they can be used for the robot placement problem. Before carrying out this step, the user is prompted to verify that both the weld and scan paths generated between steps 2 – 5 are suitable for use. The user then simply presses the 'GenPaths' button, shown in Figure 5-19, and all of the generated paths are automatically collated in the correct sequence, which varies depending on the scan set up used.

5.6.3 Step 7: Robot Placement

Due to the complexity of the robot-on-robot-on-rail set up featured in this robotic system, the approach taken for the overall robot placement problem is to address each robot (auxiliary and weld) separately. The key component that links these two problems together is the seam's 'zone' defined during the setup phase in Section 5.5.4. This approach, depicted graphically in Figure 5-21 below, involves finding targets that are contained within the zone that can be reached by the auxiliary robot without causing collision. The targets that do not pass this test are subsequently removed from the zone. The next phase of testing involves positioning the weld robot at each of the tags remaining in the zone, and then checking whether it can carry out the required motions for the weld and scan paths generated during step 6 without travelling outside its joint limits, or causing collision with the hull. Any tags that fail this second test are also removed from the zone. The tags that remain in the seam's zone after these two tests must therefore allow the weld robot to carry out the complete weld process, and also ensures that the auxiliary robot is able to position the weld robot at this location. This two-phase approach dramatically reduces the complexity of the overall problem, avoiding the situation where one optimal result must be selected from infinite number of potential solutions. Typically, a zone with 343 positions is used, for which the resultant robot placement problem will take a number of minutes to process. As there are often a number of seams to be welded from each zone, the results of the grid testing for the auxiliary robot are saved so they can be recalled at any time. This allows us to avoid re-processing this time consuming computation when another nearby seam is programmed.

The processes used for finding robot placements for both the auxiliary and weld robots are detailed over the next two sections.

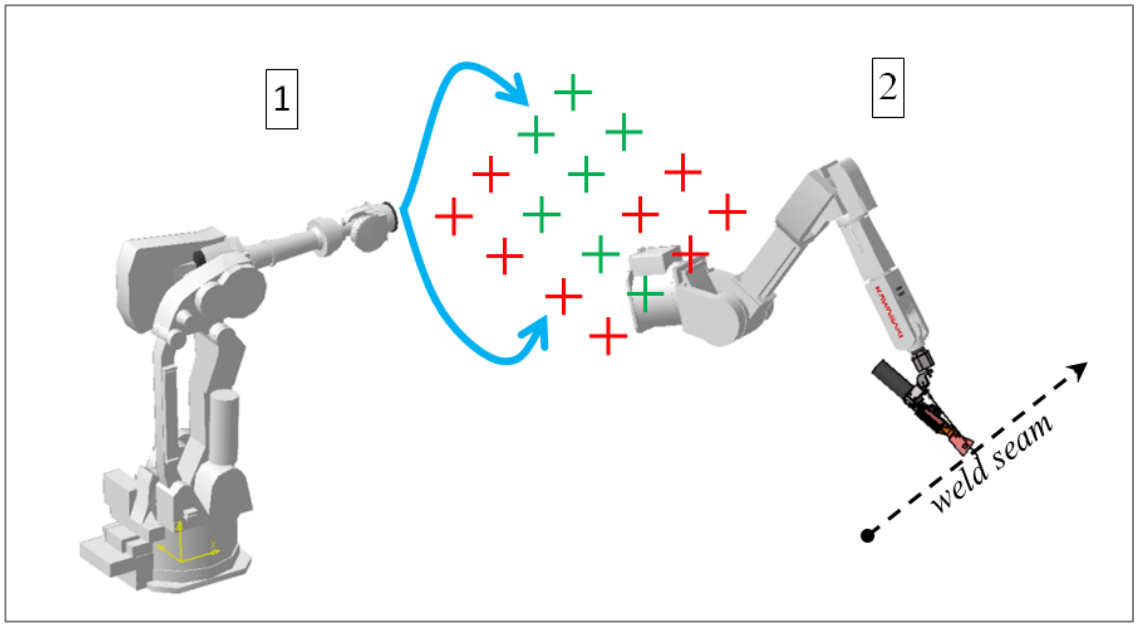


Figure 5-21 Grid-based approach for robot placement.
The auxiliary robot is tested first (1), followed by the weld robot (2). The grids must be reachable and collision-free for both robots.

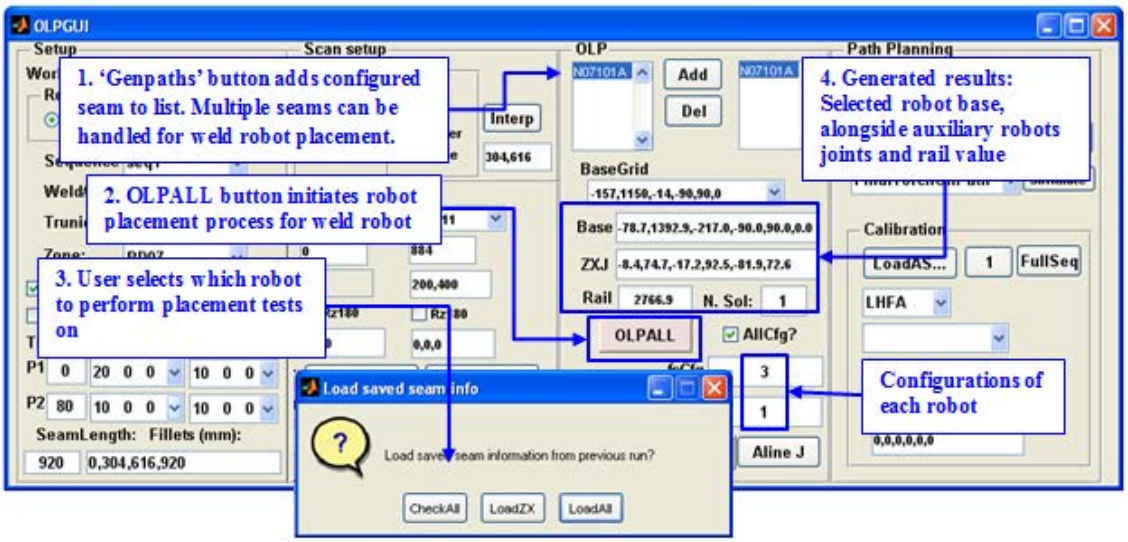


Figure 5-22 Step 5: Robot placement.

5.6.3.1 *Auxiliary Robot Placement*

To carry out the first robot placement problem, the user clicks the 'OLPALL' button on the MATLAB GUI. A pop-up prompts the user whether they want to position the auxiliary or weld robot. They must first select the 'auxiliary robot' option. Each of the tags that make up the zone are then tested for reachability and collision. The tags which are found to fail this test are removed from the zone. This test usually takes about 2 minutes, depending on the size of the zone used. Once complete, the result is saved so that this time-consuming process does not need to be performed in the future tests featuring the same zone. If the zone has been previously tested with the auxiliary robot on a previous seam, the user is alerted to this fact and can then select an option which instantly load the results from this previous test, as opposed to re-processing the entire problem.

Once completed, the 'BaseGrid' dropdown box of the MATLAB GUI is filled with all of the nodes of the zone that were found to be both reachable and collision-free for the auxiliary robot. A plot, shown in Figure 5-23, is generated at this time to provide a visual representation. The results of the reachability tests for the auxiliary robot are also plotted. The zone tags found to be valid for both robots are displayed in blue, whilst the zone tags that did not pass the robot placement tests are plotted in pink.

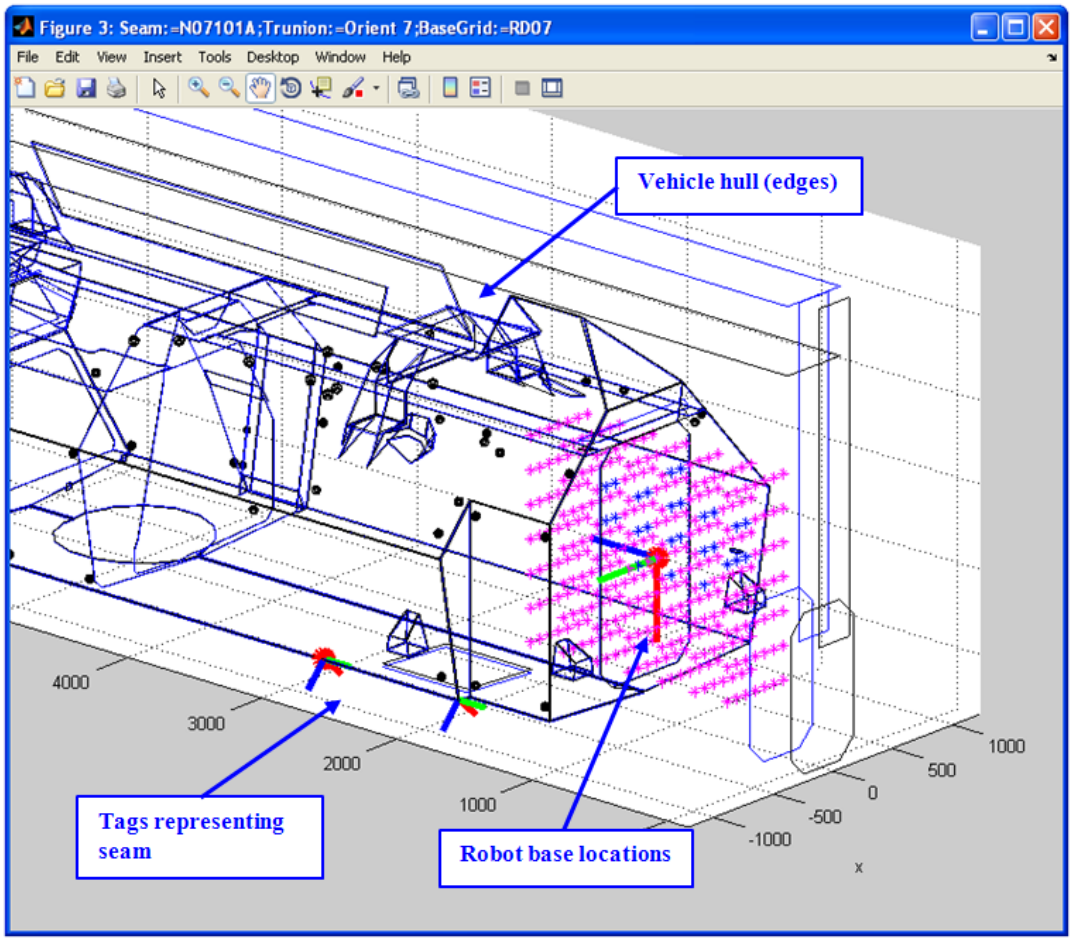


Figure 5-23 Results from the auxiliary robot placement problem

5.6.3.2 *Weld Robot Placement*

The next part of the robot placement problem focuses on properly positioning the weld robot so that it can carry out the generated weld and scan paths. The user again clicks the 'OLPALL' button, this time selecting the weld robot from the options presented in the resultant pop-up window. An algorithm then places the weld robot at each tag that remains in the zone after the auxiliary robot placement tests, and attempts to carry out the weld and scan paths generated during step 6 without exceeding the weld robots joint limits, or causing clash. The user inputs available for this step are:

1. **N.OLP.** This specifies which result to be stored and used for later steps. Normally the 1st found result is used. The program is setup to stop when n^{th} result is found, which avoids testing all potential zone tags.

2. **Plot?** A check box specifying whether to graphically simulate the placements and movements of each robot.

The step will update a few pieces of information on the user interface, including Config (The configuration of robot), Base (The base position of the weld robot), ZXJ (The joint angles for auxiliary robot), Rail (The rail value), and the path list for simulation. If this step does not find any valid solutions, the user might need to define a different entry zone, change tool setup parameters presented in Section 5.6.3 (trim, Rz180, Swap) or select different tool angle and try again. Once the process is complete and the weld robot has been positioned, a plot that shows the corresponding joint angles the weld robot uses when traversing the weld and scan paths is generated, as shown in Figure 5-24 below.

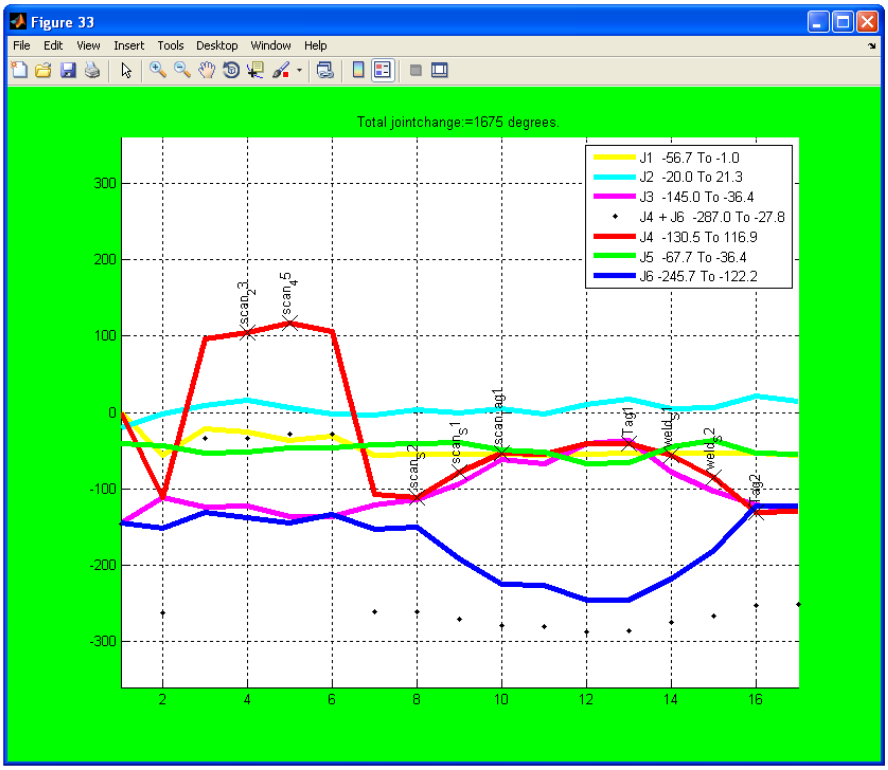


Figure 5-24 Joint analysis for robot placement problem
(Green border indicates all joints within suitable tolerances)

5.6.4 Optional Step 8: OLP Verify

This step provides a manual option for the user to alter the configuration or base position of the weld robot. This is used occasionally in more difficult seams, where a minor alteration to the result provided by the robot placement algorithm is required.

5.6.5 Step 9: Weld Robot Transition Motions

This part of the AOLP process involves the generation of a series of robot motions that can lead the weld robot from its folded position (home) to the weld seam without collision. The start and goal configurations, used as the basis for each motion planning query, are defined during the robot placement process completed prior to this step. In the same process, a series of motions that direct the robot back home from the configuration it finds itself in once it reaches the end of the weld process is also performed.

This problem is addressed via the use of a motion planning algorithm, and is easily performed as it is entirely automated via the use of the LSEA algorithm presented in Section 4.1.3. All the user needs to do it click the 'FSPath' button, shown in Figure 5-25, once the robot placement process is complete and the motion planning algorithm is then initiated. This process can be plotted in real time, which assists the user in analysing how the algorithm is addressing the problem. As this plotting significantly slows the speed in which the algorithm can solve a given problem, it is optional. Once complete, the generated roadmap is plotted for the user to inspect, as shown in Figure 5-26. Also produced is another plot, outlining the displacement of each robot joint as it traverses the complete process path, as shown in Figure 5-27. The plot indicates whether the robot is able to travel the path in a kinematically feasible manner, and also allows the user to easily inspect the range of motion used by each joint.

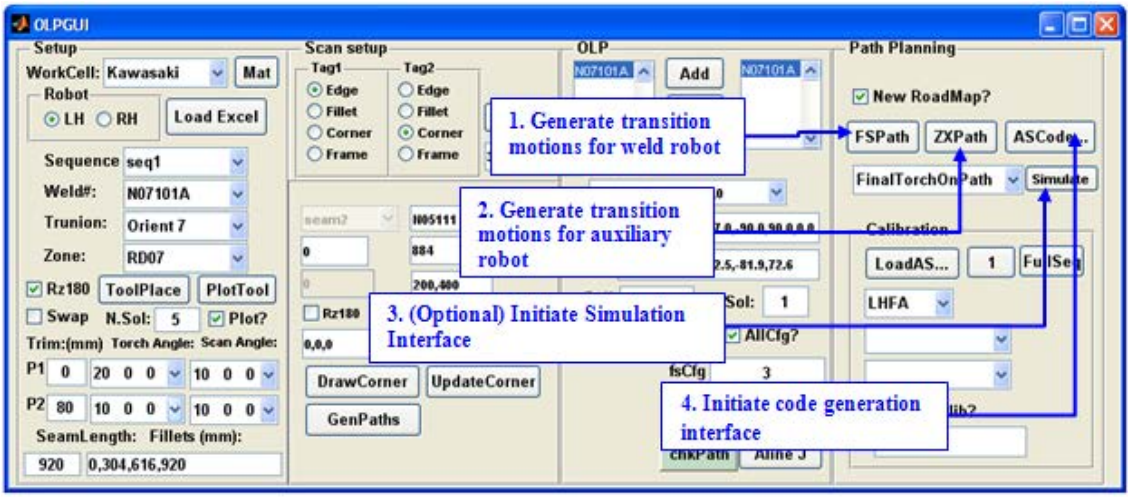


Figure 5-25 Steps 9-12

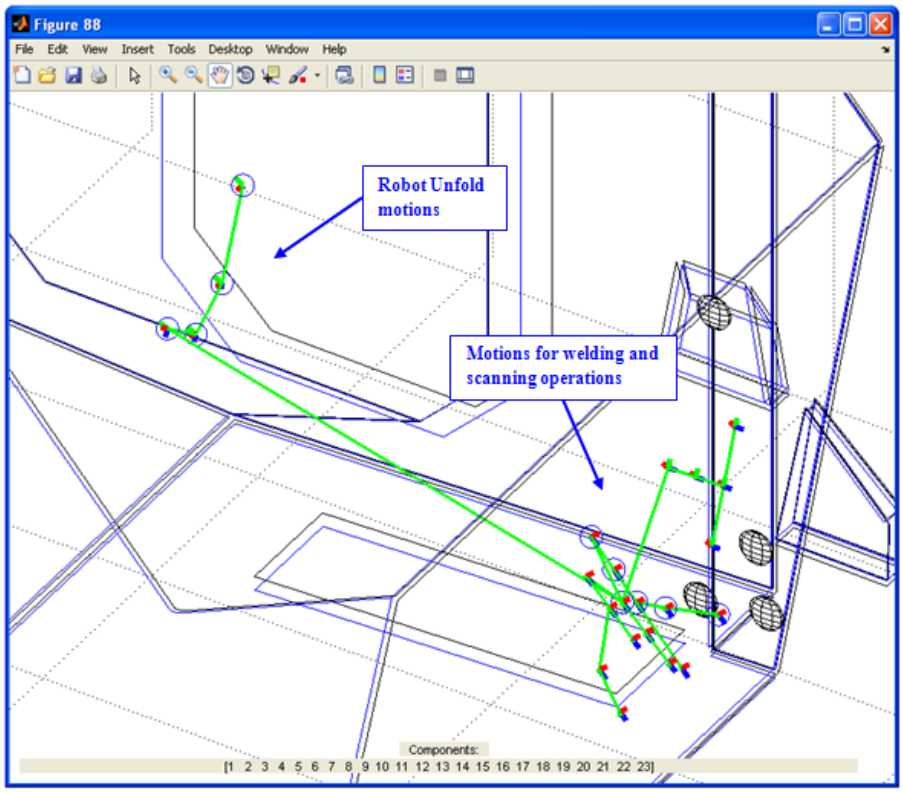


Figure 5-26 Generated roadmap for weld robot

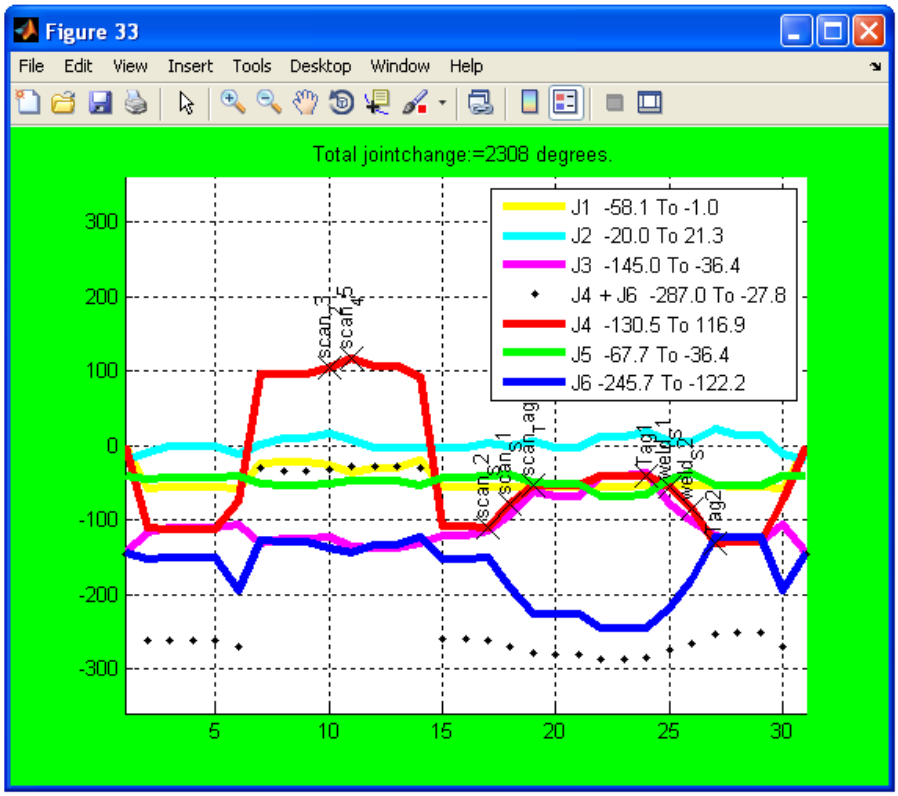


Figure 5-27 Joint analysis for complete weld robot path.
(Green border indicates all joints within acceptable ranges)

5.6.6 Step 10: Auxiliary Robot Transition Motions

This step generates the transition motions for the auxiliary robot. The goal is to transport the weld robot from a predefined home position, to the position generated during the weld robot placement process in step 7. The same LSEA algorithm (Section 4.1.3) is used to solve this motion planning problem. In some instances, the auxiliary robot needs to position the weld robot inside the vehicle hull which complicates the motion planning problem significantly. In these cases, the algorithm is assisted with the inclusion a number of user defined way-points. This process can also be plotted in a real-time simulation, and once complete, a plot of the generated roadmap is generated as shown in Figure 5-28 below.

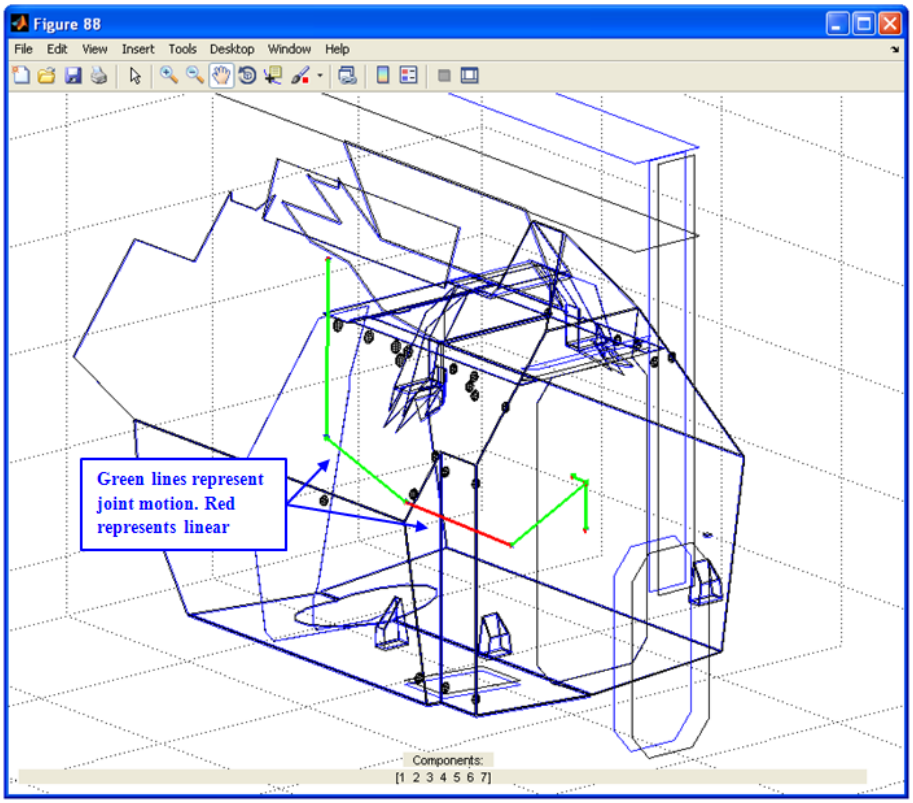


Figure 5-28 Generated roadmap for auxiliary robot

5.6.7 Step 11: Generate Robot Code

After step 10 of the programming process is complete, all required robot motions for the specified seam have been generated. What remains is to produce a robot code file that commands the physical robots to perform the set of simulated motions in the real world workcell. In this step, two robot programs are generated: One for the weld robot and another for the auxiliary robot. This step requires minimal user interaction, as it is mostly automated. The user clicks the 'AScode' button, which brings forward the supplementary code generation panel (shown in Figure 5-29) onto the main MATLAB GUI. The user can then update various inputs to the code, such as the weld number, scan type, seam name etc. Once completed, all that needs to be done is to click the generate code button, which finishes the process. The two code files generated for the seam in this example is presented in Appendix Section 7.2.

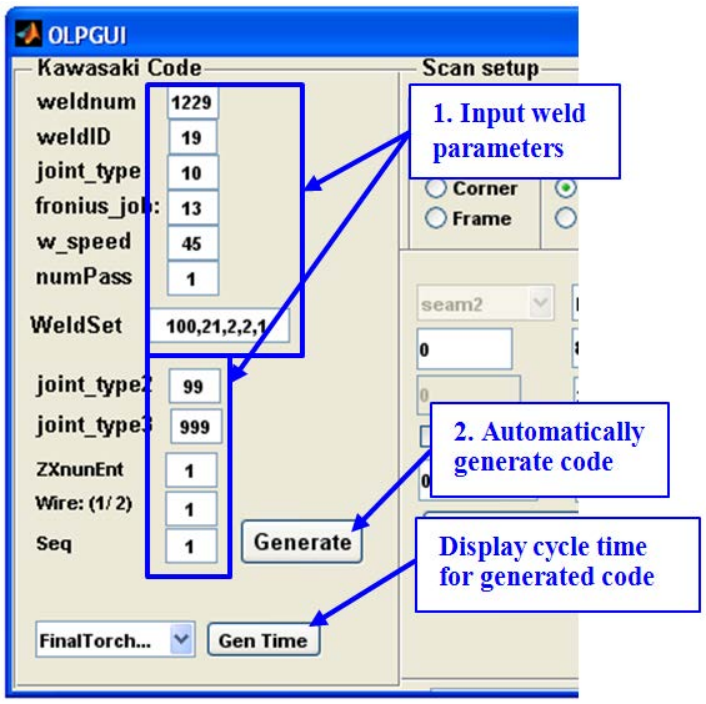


Figure 5-29 Step 11: Generate robot code

At this stage, the AOLP programing process for the given seam is complete. All that remains is to upload the generated code into the real world automation system's robot controller. However, in many situations the user will want to visualise and inspect the robot motions before uploading them for use. The simulation step, presented in the next section, provides this opportunity.

5.6.8 Step 12: Simulation

As robot paths are generated, they are uploaded automatically into the simulation interface for use. This allows the user to perform simulation of certain paths as early as step 7 of the AOLP process. The list of all paths that are available for simulation are summarised in Table 5-1 below, including the various programming stages where they become available.

Table 5-1 Paths available for MATLAB simulation

Path name	Description	Created
FinalTorchOnPath	Path for welding torch with 30mm interpolation	Step 5: OLP
FinalScannerOnPath	Path for scanner on the welding seam with fillets	Step 5: OLP
weldPath	Path for welding with fillets.	Step 7: GenPaths
scanPath(i)	Path for scanner with fillets, i=1 is temperature check path, i=2,3,4 are scan paths.	Step 7: GenPaths
PathCollection(i)	Each PathCollection represents transition motions from a certain weldPath/scanPath to the next.	Step 8: FSPath
fullPath	The complete path for weld robot including weldPath, scanPath, and PathCollection in the right order	Step 8: FSPath
ZXEntryPath	Path for auxiliary robot	Step 9: ZXPath

To carry out the simulation, the user selects a path from the drop down menu and clicks the simulation button, as shown in Figure 5-30. A supplementary simulation panel then appears to the left hand side of the GUI, which allows the user to access various simulation-specific commands. A figure displaying the robotic set-up is also generated at this time. The detailed operation of the simulation panel is explained in Figure 5-31 and a plot of a typical simulation is shown in Figure 5-32. During simulation, the user has access to various commands that allow the position of waypoints to be altered. When the robot reaches a certain waypoint, the user can simply update either the position in Cartesian space or specify a new set of joint angles, and then click the 'UpdateTag' button. If the modifications do not cause the robot to go out of joint limits, or cause collision, these changes are then kept. If the updates are invalid, the user is alerted and the system reverts to the previous settings.

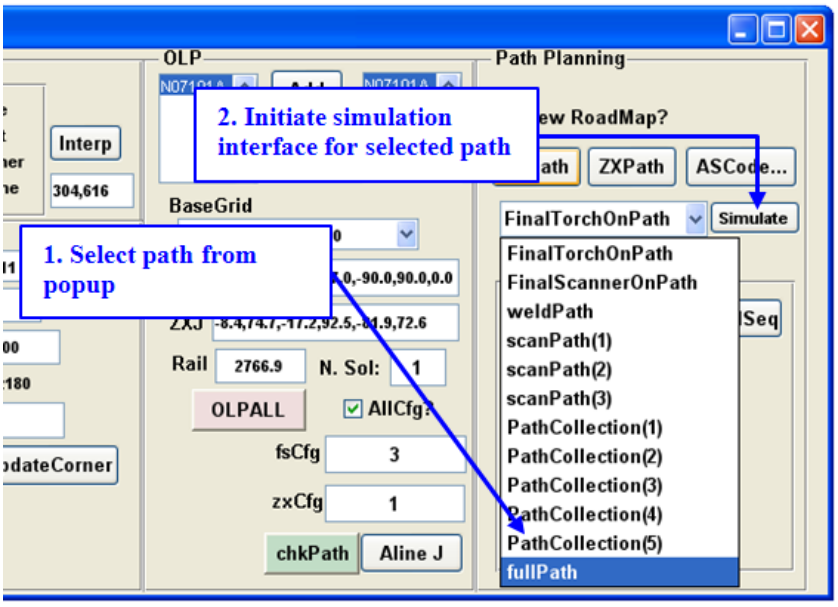


Figure 5-30 Step 12: Initiate simulation.

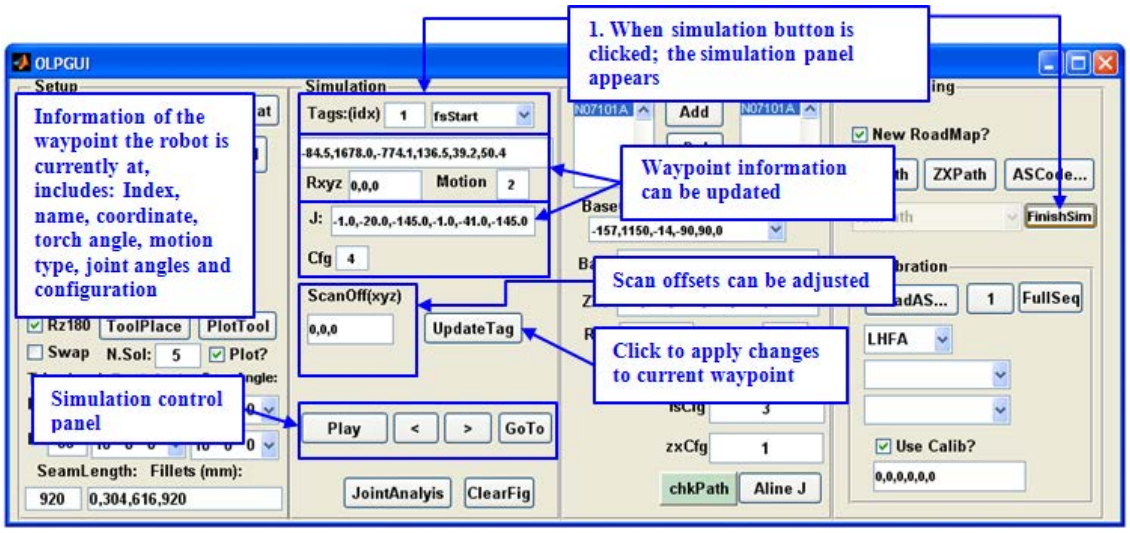


Figure 5-31 Step 12: Simulation control panel.

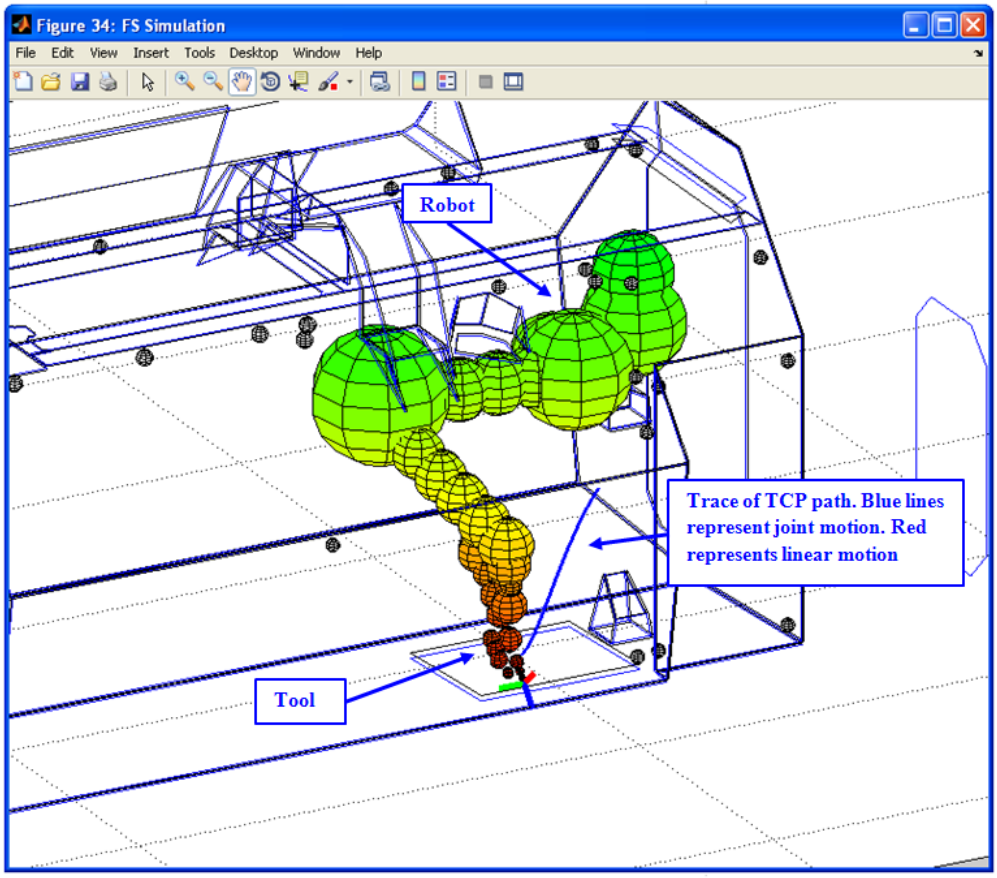


Figure 5-32 Simulation display window

5.7 UPLOADING AND USING GENERATED AOLP CODE

Once the code files for each robot are successfully generated, the next task involves uploading them into the real world workcell for use. Before the generated robot code can be adopted into the system, however, a number of steps must be carried out to commission and test it. This Section outlines the process of how this is done, which involves three stages of testing: Syntax and sequence handling, manual testing and finally a full automatic test.

5.7.1 Syntax and Sequence Handling

Controlling the order in which seams are welded by the robotic manufacturing system is a separate code file that contains the welding sequence. Each time a robot code is produced by the AOLP system, calls must be inserted into this sequence file so that the

weld code is carried out by the system at the correct moment of the manufacturing sequence. The sequence file also controls other elements of the manufacturing process, such as orientating the trunnion or running pre-heat routines.

Two robot code files are the generated (presented in Appendix Section 7.2). These files are inserted into the master file, which contains all of the other individual seam files that have been generated. When inserting the generated code into this depository, checks must be performed to ensure that syntax relating to variable names and weld numbers are unique, and are not utilised in the code file of another seam. This task is relatively simple, as these syntax checks are performed by the robot controller itself when uploading the robot code. If any errors are found, the user can simply alter the name of the offending piece of syntax on the GUI's code generation panel (Figure 5-29) and then re-generate the code. If it is only a minor change, the user can also alter the instances of the offending syntax using basic text editing software.

5.7.2 Manual Test

Once the robot codes have been successfully uploaded into the controller, manual testing of the code is performed. The robotic system is switched to a manual mode, where robot motions can only be commanded via use of the teach pendant device. The robot code is loaded into the teach pendant and the user is able to then step through this code in a line-by-line fashion. In the manual setting, the robots maximum attainable speed is slowed to 50%, but is usually set slower than this.

This phase of testing is critical. There will always be a variance between a virtual model and its corresponding real world environment, and this format of testing allows the robot programmer to ensure that no collision will occur due to these differences. The user can step through the program, line-by-line, in a slow and controlled manner. If a waypoint along the path is found to be in an unsuitable position, it can easily be modified with the teach pendant interface.

The first component of the manual test involves confirming that the scan locations specified during steps 3 and 5 of the AOLP process are able to accurately capture the local topology about the weld seam. Due to the required accuracy of the weld process

(~1mm), the quality of the locations scanned are important. For instance, if the edge of a plate is to be scanned, there cannot be any tack welds in the way of the scan, or sometimes a specified scan location is unsuitable due to reflections caused by local geometry. The quality of a scan is easily monitored through the use of the WeldCom interface (Figure 5-8, page 125), which plots the quality of the scan in real time. The operator can cycle the robot to the scan position with the scanner on and then check the quality of the resultant scan through the WeldCom interface. If the geometry cannot be interpreted, the user can alter the scan position or orientation through the teach pendant until the quality of the scan is suitable for use. These changes are easily adopted into the robot code via the teach pendant.

Once all scans are deemed suitable, the full robot code can be cycled (with the system still in manual jogging mode). The importance of this step is to check that the updated scan locations are able to re-generate a calibrated set of seam coordinates. The whole code is cycled with the weld torch power supply switched off. During this step, the user is to inspect the accuracy of the torch-tip as it carries out the seam (Figure 5-33). If found to deviate too far from the seam, the scan positions will most likely need to be reconfigured so that they generate a more optimal result.

Figure 5-33 Manual inspection of the quality of the re-generated seam coordinates

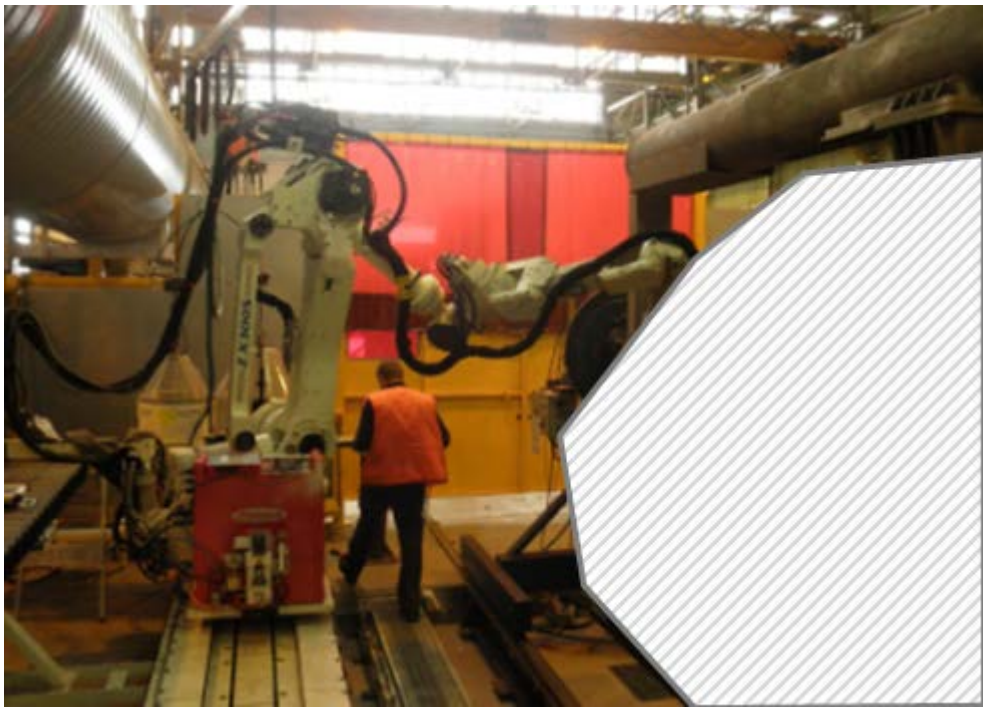


Figure 5-34 An operator cycling the robot through the generated code during the manual testing phase
(Vehicle hull edited out)

This process is carried out on the entire path, for both robots. Once the user is satisfied with the clearance of the robot motions, the quality of the generated scans and the resultant accuracy of the interpolated seam, they can proceed to the automatic testing. Before this however, the final step to this procedure is to jog through the entire robot code from the start again, to ensure all of the modifications carried out are satisfactory.

5.7.1 Full Automatic Test

This mode of testing involves switching the robot cell to automatic mode, and letting it carry out the entire generated code with the robots running at full speed. At first, this is to be run with the weld torch switched off. Then, the seam can be tested with the torch switched on, so the final weld seam can be inspected for quality. Once these tests have been performed, the code is now ready to be used for production purposes.

6 DISCUSSION & CONCLUSIONS

6.1 EFFECTIVENESS OF THE AOLP APPROACH

Due to the fundamental differences that exist between the AOLP approach and the conventional online methods used previously to program the industry partner's robotic cell, it is difficult to accurately evaluate the effectiveness of AOLP. Nevertheless, the AOLP approach demonstrated a clear advantage in terms of reduced programming times for weld seems in the robotic cell featured in Chapter 5.

Figure 6-1 graphically illustrates the observed benefit of the AOLP approach compared with manual online methods. It can be seen that, whilst the productivity of AOLP is initially hampered due to its setup phase (which is not required with online methods), this is quickly repaid by the more rapid and efficient means to generate, test and finally upload a weld seam into the robotic system for use. This suggests that if only a small number of seams are required to be programmed, conventional online programming will most likely form the more suitable option of the two. As more seams are programmed, however, the time benefit of the AOLP approach rapidly shortens the gap. Not many seams are required before the AOLP approach becomes the more beneficial option for programming the cell. The more seams programmed, the larger the benefit of the AOLP system. In addition, once the setup phase has been successfully carried out, it will not have to be repeated for any future generation of robot code, so in the event of product changes, there are immediate benefits.

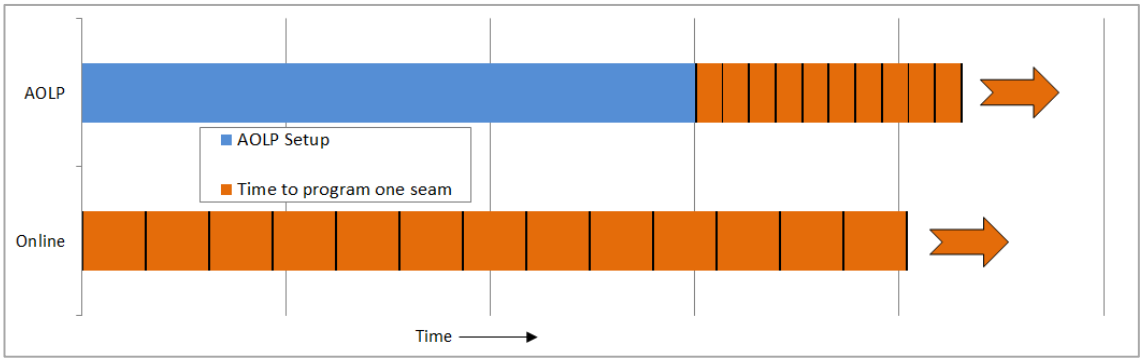


Figure 6-1 Representation of programming times for AOLP and conventional online programming

The AOLP approach is truly an offline approach to robot programming. The programmer can be situated offsite as no physical robotic hardware is required for the programming process. Provided that robot operators are available to carry out the necessary testing phase *in situ*, generated program files and subsequent feedback can be transferred electronically. Offline methods also feature drastically reduced system downtimes when programming is taking place.

Of the three distinct phases that make up the AOLP process (setup, programming and testing), the setup phase takes the longest to carry out. However, the effect of this is minimised as this setup only needs to be performed once. The complexity of the robotic welding system and workpiece used in this research also contributes to the extended time of the setup phase. More specifically:

- The complexity of the vehicle hull. The hull features a high number of plates that need to be modelled with bounding box CAD representation.
- The high number of seams in the hull (~300), which need to be individually defined with the Visual Basic/DELMIA Interface.
- Correctly defining all the required zones for the robot placement problem.

For an experienced user of the developed AOLP system, this setup phase took four days to complete. Great care must be taken when performing the setup phase to ensure that all items are modelled as accurately as possible. This reduces the risk of having to re-validate, or completely regenerate, previously programmed weld seams if errors in modelling the robot models or workcell environment are identified.

For each seam in the vehicle hull, the programming phase is typically performed in a straightforward manner. For a single seam, it was generally found to take between 5-10 minutes to generate the program file that commands the robots to carry out their welding processes. This variance in programming time is attributed to the fact that some of the seams in the hull are in easy to reach locations, and others are located in regions with restricted robot access. Seams that have few geometric constraints, such as external seams, have fewer problems requiring additional attention. Other seams are located in far more difficult regions of the vehicle hull. In these situations it can be much more difficult to properly position the weld torch without causing collision. The robot

placement (Section 5.6.3), motion planning (Section 5.6.5) and optimisation (Section 5.6.6) problems are also made far more difficult in these situations.

Nevertheless, most of the seams in the hull were programmed with little difficulty. In approximately 10% of the seams, difficulty was experienced during the programming phase, and hence required additional attention. For these seams, much of the difficulty stemmed from the fact that the sphere models used to represent the robot and torch were too large to properly fit around nearby plates. In these situations, the seam was modified by trimming both edges to allow more room for the torch. Then, during the testing phase, these points were manually modified by the robot operator with the teach pendant to move the start/end points of the seam to their required location.

The testing phase was found to take far longer to perform than the programming phase. Typically, approximately two hours was required to carry out the manual and automatic tests that are necessary for the code to be adopted into the robotic welding system for use. It is expected, however, that this timeframe would be shortened as the operator becomes more familiar with the procedure.

The time required to manually program a weld seam using conventional online methods varied greatly on this robotic system. Simple seams were reportedly programmed in a day, whereas the more difficult seams took two. The main difficulty encountered related to properly positioning the various robots to carry out the complete weld process for a single seam. For example, it was reportedly a common occurrence that once the weld robot was positioned inside the hull, a large portion of the weld seam could be programmed only to find that the final portion, or a scan location, was found to be out of reach of the robot. In these situations, the programmer would have to scrap the code generated up until that point and start again by re-positioning the weld robot to improve its chances of carrying out the entire weld process.

It is in these situations that the AOLP approach provides a major benefit over the conventional online methods previously used. The AOLP system's ability to rapidly solve robot placement problems for both auxiliary and weld robots is one tool that online robot programmers do not have at their disposal. Another benefit of AOLP, apart from the significantly reduced programming time, is the fact that the tool placement, robot placement and motion planning algorithms generate optimal results that can be

verified via algorithmic processes. This information is not available when using online methods of robot programming, where the generation of optimal results relies on the experience of the robot programmer, which often varies greatly.

6.2 CASE STUDY: APPLYING AOLP TO ANOTHER ROBOTIC SYSTEM

There was an opportunity to develop the AOLP system so that it could be utilised with another robotic welding system in the same manufacturing facility. This robotic system was used to perform the welding of small fixtures, in the form of threaded bosses, onto a number of the vehicle hull's larger plates. This process is performed before the plates are tack welded together to form the vehicle hull's structure.

A design change to the vehicle hull was made which altered the layout of these bosses, meaning that the robotic cell that performs the welding of these fixtures onto the hull would need to be re-programmed. The industry partner intended to effect this change using a conventional online programming approach; however, this was seen as a valuable opportunity to test the developed AOLP systems ability to be re-configured for use in another robotic application.

The robotic welding system used in the welding of these fixtures was fundamentally different to the robotic system presented in Chapter 5. The robotic welding process involves two separate Kuka brand robots, as shown in Figure 6-2 below. To perform the welding process for one boss, one robot first moves and grasps one of the bosses with its attached gripper, and then moves to the location on the plate the boss is to be attached. The robot holds the boss in position, whilst the other robot then moves in and deposits two tack welds, which fasten the boss onto the plate. The gripping robot then retreats from the area, and the weld robot completes the weld process by performing a full circular weld about the perimeter of the boss. In total, there were 40 bosses that would need to be programmed into the cell.

It took two weeks to develop a new AOLP system for this separate robotic welding cell. Once complete, the programming process was carried out in less than two days.

Figure 6-2 The Kuka Robot Welding Cell

Modifying the previously developed AOLP system so it could be utilised on another robotic cell was simplified as most of the algorithmic functions in the existing AOLP system required no update. It is expected that this would be the case in most other conversions. The key components in the AOLP system that required modification related to:

- Kinematic models: A different set of industrial manipulators were featured in this system, so new forward and inverse kinematic functions were developed for this specific hardware.
- GUI: Due to the fact that the new robotic system features a completely different welding process and sequence, a new GUI to handle this was created, which is shown in Figure 6-4 below.
- Robot target generation was altered significantly. All bosses are welded in the same manner and hence all that needs to be specified to the robot is the location

of the boss to be welded. These were locations were obtained from a 2D CAD design drawing, and stored in an Excel spreadsheet for use.

- Two robots are directly featured in the weld process, so robot placement problem was altered to accommodate this
- Post processing and code generation: Due to the use of different robotic hardware and weld equipment, new code generation functions were created in order to produce a robot code that is compatible with the hardware used.

Compared to the Kawasaki robotic welding system presented in Chapter 5.1, the Kuka robotic system in this application is relatively simple. The robot-on-robot-on-rail setup is not used in this cell, simplifying the robot placement problem. Also simplified is the motion planning process, as the work area is far more spacious. This meant that the complete programming process for one boss could be performed within 45 seconds, in a fully automatic manner. The testing phase for this robotic system was also rapid due to the simplified robotic setup. Codes could be rapidly cycled and checked by the robot operator. Once generated by the AOLP system, not one of the 40 generated codes were found to require significant modification, ensuring that these codes could be rapidly adopted into the robotic workcell for use.

Figure 6-3 Detail of the dual robot welding process

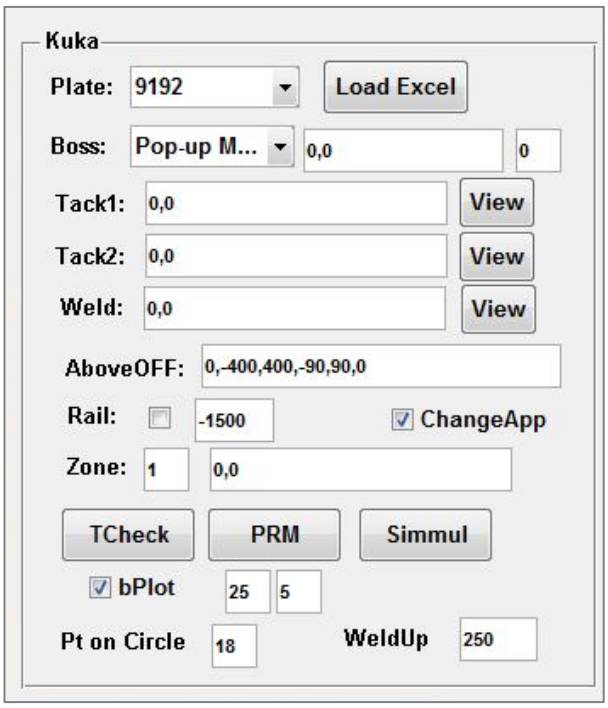


Figure 6-4 The Kuka GUI

In all, 2 weeks was spent developing this new AOLP system (including the setup phase) to accommodate the new robotic setup, and 2 days was spent programming and testing the new seams. This complete development and programming time would, indeed, exceed the time taken to program this cell from scratch with online methods. However, there now exists an AOLP system configured for this robotic workcell, which means that any future programming work can be performed far more rapidly than any other option currently available to the industry partner. Most importantly, it was further demonstrated that the AOLP code can be adapted for use in another robotic system in a relatively easy and straight forward manner.

6.3 CONCLUSIONS

In this thesis, a novel methodology for the efficient programming of industrial robotic manufacturing systems is presented.

In the current global economic climate, there is much need for effective methods of programming these industrial systems as programming is often a difficult, time consuming and costly exercise. This has particular effect in the Australian manufacturing industry, as a large portion of the total manufacturing output is carried out by small to median enterprises which typically manufacture their items in small batches. In these applications, a robotic manufacturing system would require frequent reprogramming, which poses a significant barrier for SME-based manufacturing operations, as they generally possess far fewer resources at their disposal. As a result, manufacturing-based SMEs in Australia rarely make effective use of robotic automation due to the expected high costs and difficulties associated with programming, which has a significant negative flow-on effect to the wider Australian manufacturing industry.

An extensive review of advancements in robot programming technology over the last ten years uncovered that this problem has prompted many other investigations at various institutions worldwide. The most common approaches taken by others involved the development and adaptation of additional sensors or control technologies into the online programming process, in order to simplify various aspects of the programming task.

In this body of work, an efficient approach to the offline programming of these robotic systems was proposed, developed and tested in a real world manufacturing scenario. This approach, termed the Automated Offline Programming (AOLP) method, integrates computer automation - in the form of algorithmic processes - to automate many of the laborious, time consuming and repetitive components of the offline programming process. By automating many of steps that are traditionally carried out by a human operator, the AOLP method is able to more rapidly and efficiently carry out the programming of an industrial robotic system, whilst also retaining many of the benefits offline approaches to robot programming enjoy.

The effectiveness of the AOLP system was validated with a series of tests carried out on the robotic welding system of an industry partner. The developed AOLP system was found to drastically outperform the conventional online programming techniques used to program the workcell in the past. Using the AOLP approach developed in this work, a robotic welding cell was successfully programmed with the following outcomes:

- No changes to the existing robotic system were required for the new programming technique.
- Once the AOLP system was developed for the particular robotic cell, it was found to be able to significantly reduce the time taken to program weld seams when compared to the previously used online methods.
- The developed AOLP system has also been modified for use on another robotic manufacturing system, demonstrating the AOLP software's ability to be used in other manufacturing systems and tasks.

Also presented in this thesis were additional research topics in the fields of robotic motion planning and path optimisation, presented in Chapters 4.1 and 4.2 respectively. A new and efficient approach for sampling-based motion planning was proposed and tested on a variety of different robotic systems. This algorithm makes use of a superior non-uniform sampling scheme to efficiently navigate obstacles in a robot's environment, and was found to significantly outperform a selection of other popular motion planning algorithms when tested. In addition, a new and efficient approach to the post-process optimisation of robot paths was also presented. When compared to the algorithms that it was derived from, this new approach, termed the Adaptive Partial Shortcut Algorithm, was observed to reduce the time required to smooth a typical path of an industrial manipulator style robot by up to two-thirds.

This AOLP technology is now available to our industry partners for use. The main outcome of this is that the previously time consuming and costly programming overhead of their robotic welding cell is now drastically reduced. This provides them significant benefit, as this programming overhead had previously presented major barriers for proposed future use of the cell, despite the fact that the robotic equipment had already been set up.

6.3.1 Recommendations

Recommendations for future developments of the AOLP approach presented in this thesis relate to removing the reliance on the third party software DELMIA. The primary reason for doing this is to transform future versions of AOLP into standalone pieces of software, which run completely in the MATLAB programming environment. This would be of great benefit to potential end-users, as it would not only simplify the use of the software, but also eliminate issues associated with licencing. However, if DELMIA is to be removed from the AOLP system, significant development would be required in order to handle the steps that were previously performed in the DELMIA environment. In the current version of the AOLP system, DELMIA is utilised during the setup phase for the generation of the simplified CAD models that are used (Section 3.2.1), and also for robot target generation purposes (3.2.2).

To replace the CAD generation step, a recommendation would be to completely do away with the current use of simplified CAD representations, in favour of utilising a full representation of the CAD model in the form of an .STL (or similar) file format. These files can provide benefit over the previously used bounding models due to the fact that they:

- Can be loaded directly from their source file; interfacing with Excel would not be necessary.
- Will allow the AOLP system to make use of full detail CAD representations that will better represent their real world counterparts, allowing collision processing in the AOLP system to be more accurate.
- Can be used by the AOLP system to represent all of the elements in the workcell, including the robots, tooling, workpiece and surrounding obstacles.

If these models are to be used, however, it is expected that associated collision processing algorithms will be slowed significantly due to the vast increase in geometric elements processed with each collision test. This could be addressed in a number of different ways such as developing the collision processing algorithms in a low level programming language, such as C++, which has faster computational speed, or by

utilising parallel processing hardware, such as a graphics card, to assist in speeding up computation times.

REFERENCES

1. Australian Department of Innovation, I., Science and Research. *Key Statistics Australian Small Business*. [Government Report] 2011 [cited 2013 10/10/2013]; Available from: http://www.innovation.gov.au/smallbusiness/keyfacts/Documents/SmallBusinessPublication.pdf.
2. Taskforce, P.M.s.M. *Report of the Non-Government Members*. [Government Report] 2012 8/10/2013]; Available from: http://www.innovation.gov.au/industry/manufacturing/Taskforce/Documents/SmarterManufacturing.pdf.
3. Boothroyd, G., *Economics of assembly systems.* Journal of Manufacturing Systems, 1982. **1**(1): p. 111-127.
4. Robotics, A. *ABB IRB4400 Specification*. [Product Literature] 2013 [cited 2013 21-05-2013]; Available from: http://www05.abb.com/global/scot/scot241.nsf/veritydisplay/0dc8b0df9cc39049c125772e0057e806/$file/IRB%204400%20PR10035EN%20R8.pdf.
5. Kihlman, H., *Affordable automation for airframe assembly: developing of key enabling technologies*. 2005, Linköping.
6. Ang Jr, M.H., L. Wei, and L.S. Yong. *An industrial application of control of dynamic behavior of robots-a walk-through programmed welding robot*. in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*. 2000: IEEE.
7. Dietz, T., et al. *Programming System for Efficient Use of Industrial Robots for Deburring in SME Environments*. in *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*. 2012: VDE.
8. Sugita, S., T. Itaya, and Y. Takeuchi, *Development of robot teaching support devices to automate deburring and finishing works in casting.* The International Journal of Advanced Manufacturing Technology, 2004. **23**(3-4): p. 183-189.
9. Choi, M.H. and W.W. Lee. *A force/moment sensor for intuitive robot teaching application*. in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*. 2001: IEEE.
10. Schraft, R. and C. Meyer, *The need for an intuitive teaching method for small and medium enterprises.* VDI BERICHTE, 2006. **1956**: p. 95.
11. Pan, Z. and H. Zhang. *Robotic machining from programming to process control*. in *Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress on*. 2008: IEEE.
12. Nicholson, A., *Rapid adaptive programming using image data.* University of Wollongong Thesis Collection, 2005: p. 422.
13. Zhang, H., et al. *On-line path generation for robotic deburring of cast aluminum wheels*. in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. 2006: IEEE.

14. González-Galván, E.J., et al. *An algorithm for optimal closed-path generation over arbitrary surfaces using uncalibrated vision*. in *Robotics and Automation, 2007 IEEE International Conference on*. 2007: IEEE.

15. Hu, Z., et al., *Automatic surface roughing with 3D machine vision and cooperative robot control.* Robotics and Autonomous Systems, 2007. **55**(7): p. 552-560.

16. Takarics, B., et al. *Welding trajectory reconstruction based on the Intelligent Space concept*. in *Human System Interactions, 2008 Conference on*. 2008: IEEE.

17. Solvang, B., G. Sziebig, and P. Korondi, *Robot Programming in Machining Operations.* Robot Manipulators, I-Tech Education and Publishing, 2008: p. 479-496.

18. Bi, Z. and S.Y. Lang, *A framework for CAD-and sensor-based robotic coating automation.* Industrial Informatics, IEEE Transactions on, 2007. **3**(1): p. 84-91.

19. Larkin, N., et al. *3D mapping using a ToF camera for self programming an industrial robot*. in *Advanced Intelligent Mechatronics (AIM), 2013 IEEE/ASME International Conference on*. 2013: IEEE.

20. Kim, J., *CAD-based automated robot programming in adhesive spray systems for shoe outsoles and uppers.* Journal of Robotic Systems, 2004. **21**(11): p. 625-634.

21. Pulkkinen, T., et al. *2D CAD based robot programming for processing metal profiles in short series manufacturing*. in *Control, Automation and Systems, 2008. ICCAS 2008. International Conference on*. 2008: IEEE.

22. Pires, J.N., T. Godinho, and P. Ferreira, *CAD interface for automatic robot welding programming.* Industrial Robot: An International Journal, 2004. **31**(1): p. 71-76.

23. Kim, K.-Y., D.-W. Kim, and B.O. Nnaji, *Robot arc welding task sequencing using genetic algorithms.* IIE Transactions, 2002. **34**(10): p. 865-880.

24. Kang, H. and J. Park. *Work planning using genetic algorithm and 3D simulation at a subassembly line of shipyard*. in *OCEANS'04. MTTS/IEEE TECHNO-OCEAN'04*. 2004: IEEE.

25. Zacharia, P.T. and N. Aspragathos, *Optimal robot task scheduling based on genetic algorithms.* Robotics and Computer-Integrated Manufacturing, 2005. **21**(1): p. 67-79.

26. Dai, W. and M. Kampker. *PIN-a PC-based robot simulation and offline programming system using macro programming techniques*. in *Industrial Electronics Society, 1999. IECON'99 Proceedings. The 25th Annual Conference of the IEEE*. 1999: IEEE.

27. Mitsi, S., et al., *Off-line programming of an industrial robot for manufacturing.* The International Journal of Advanced Manufacturing Technology, 2005. **26**(3): p. 262-267.

28. Soron, M. and I. Kalaykov. *Generation of continuous tool paths based on CAD models for friction stir welding in 3D*. in *Control & Automation, 2007. MED'07. Mediterranean Conference on*. 2007: IEEE.

29. Yan, Y., et al. *A Robot Simulation System Basing on AutoLisp*. in *Industrial Electronics and Applications, 2007. ICIEA 2007. 2nd IEEE Conference on*. 2007: IEEE.

30. Jaramillo-Botero, A., et al., *Robomosp.* Robotics & Automation Magazine, IEEE, 2006. **13**(4): p. 62-73.

31. Kim, C.-S., et al. *PC-based off-line programming using VRML for welding robots in shipbuilding*. in *Robotics, Automation and Mechatronics, 2004 IEEE Conference on*. 2004: IEEE.

32. Brown, R.G. *Driving digital manufacturing to reality*. in *Simulation Conference, 2000. Proceedings. Winter*. 2000: IEEE.

33. Bruccoleri, M., C. D'Onofrio, and U. La Commare. *Off-line Programming and simulation for automatic robot control software generation*. in *Industrial Informatics, 2007 5th IEEE International Conference on*. 2007: IEEE.

34. Dong, W., H. Li, and X. Teng. *Off-line programming of Spot-weld Robot for Car-body in White Based on Robcad*. in *Mechatronics and Automation, 2007. ICMA 2007. International Conference on*. 2007: IEEE.

35. Lee, D. and W. ELMaraghy, *ROBOSIM: a CAD-based off-line programming and analysis system for robotic manipulators.* Computer-aided engineering journal, 1990. **7**(5): p. 141-148.

36. Vollmann, K. *A new approach to robot simulation tools with parametric components*. in *Industrial Technology, 2002. IEEE ICIT'02. 2002 IEEE International Conference on*. 2002: IEEE.

37. Žlajpah, L., *Simulation in robotics.* Mathematics and Computers in Simulation, 2008. **79**(4): p. 879-897.

38. Bottazzi, V.S. and J.F.C. Fonseca. *Off-line robot programming framework*. in *Autonomic and Autonomous Systems and International Conference on Networking and Services, 2005. ICAS-ICNS 2005. Joint International Conference on*. 2005: IEEE.

39. Kuffner Jr, J.J. and S.M. LaValle. *RRT-connect: An efficient approach to single-query path planning*. in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*. 2000: IEEE.

40. Sánchez, G. and J.-C. Latombe, *A single-query bi-directional probabilistic roadmap planner with lazy collision checking*, in *Robotics Research*. 2003, Springer. p. 403-417.

41. Barraquand, J. and J.-C. Latombe, *Robot motion planning: A distributed representation approach.* The international journal of robotics research, 1991. **10**(6): p. 628-649.

42. Thomas, U. and R. Iser. *A new Probabilistic Path Planning Algorithm for (Dis) assembly Tasks*. in *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*. 2010: VDE.

43. Berenson, D., et al. *Manipulation planning with workspace goal regions*. in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. 2009: IEEE.

44. Amato, N.M. and G. Song, *Using motion planning to study protein folding pathways.* Journal of Computational Biology, 2002. **9**(2): p. 149-168.

45. Kavraki, L.E., et al., *Probabilistic roadmaps for path planning in high-dimensional configuration spaces.* Robotics and Automation, IEEE Transactions on, 1996. **12**(4): p. 566-580.

46. Tsianos, K.I., I.A. Sucan, and L.E. Kavraki, *Sampling-based robot motion planning: Towards realistic applications.* Computer Science Review, 2007. **1**(1): p. 2-11.

47. Dijkstra, E.W., *A note on two problems in connexion with graphs.* Numerische mathematik, 1959. **1**(1): p. 269-271.

48. Hart, P.E., N.J. Nilsson, and B. Raphael, *A formal basis for the heuristic determination of minimum cost paths.* Systems Science and Cybernetics, IEEE Transactions on, 1968. **4**(2): p. 100-107.

49. Kuffner, J.J. *Effective sampling and distance metrics for 3D rigid body path planning*. in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*. 2004: IEEE.

50. Bohlin, R. and L.E. Kavraki. *Path planning using lazy PRM*. in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*. 2000: IEEE.

51. Kim, Y.J., et al., *Fast swept volume approximation of complex polyhedral models.* Computer-Aided Design, 2004. **36**(11): p. 1013-1027.

52. Svestka, P., *On probabilistic completeness and expected complexity for probabilistic path planning.* UU-CS, 1996(1996-20).

53. Bessiere, P., et al. *The "Ariadne's clew" algorithm: global planning with local methods*. in *Intelligent Robots and Systems' 93, IROS'93. Proceedings of the 1993 IEEE/RSJ International Conference on*. 1993: IEEE.

54. Hsu, D., *Randomized single-query motion planning in expansive spaces*. 2000, Stanford University.

55. LaValle, S.M. and J.J. Kuffner Jr, *Rapidly-exploring random trees: Progress and prospects.* 2000.

56. LaValle, S.M., *Rapidly-Exploring Random Trees A Цew Tool for Path Planning.* 1998.

57. Fragkopoulos, C. and A. Gräser. *A RRT based path planning algorithm for Rehabilitation robots*. in *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*. 2010: VDE.

58. Zucker, M., J. Kuffner, and M. Branicky. *Multipartite rrts for rapid replanning in dynamic environments*. in *Robotics and Automation, 2007 IEEE International Conference on*. 2007: IEEE.

59. Ferguson, D., N. Kalra, and A. Stentz. *Replanning with rrts*. in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. 2006: IEEE.

60. Bruce, J. and M. Veloso. *Real-time randomized path planning for robot navigation*. in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*. 2002: IEEE.

61. Kagami, S., et al. *Humanoid arm motion planning using stereo vision and RRT search*. in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*. 2003: IEEE.

62. Kim, J. and J.P. Ostrowski. *Motion planning a aerial robot using rapidly-exploring random trees with dynamic constraints*. in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*. 2003: IEEE.

63. Strandberg, M. *Augmenting RRT-planners with local trees*. in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*. 2004: IEEE.

64. Svestka, P. and M.H. Overmars. *Coordinated motion planning for multiple car-like robots using probabilistic roadmaps*. in *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*. 1995: IEEE.

65. Wilmarth, S.A., N.M. Amato, and P.F. Stiller. *MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space*. in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. 1999: IEEE.

66. Kavraki, L. and J.-C. Latombe. *Randomized preprocessing of configuration for fast path planning*. in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*. 1994: IEEE.

67. Gyorfi, J.S., et al., *Evolutionary Path Planning With Subpath Constraints.* Electronics Packaging Manufacturing, IEEE Transactions on, 2010. **33**(2): p. 143-151.

68. Atramentov, A. and S.M. LaValle. *Efficient nearest neighbor searching for motion planning*. in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. 2002: IEEE.

69. Larsen, E., et al. *Fast distance queries with rectangular swept sphere volumes*. in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*. 2000: IEEE.

70. Boor, V., M.H. Overmars, and A.F. van der Stappen. *The gaussian sampling strategy for probabilistic roadmap planners*. in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. 1999: IEEE.

71. Hsu, D., et al. *The bridge test for sampling narrow passages with probabilistic roadmap planners*. in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*. 2003: IEEE.

72. Amato, N.M., O.B. Bayazit, and L.K. Dale, *OBPRM: An obstacle-based PRM for 3D workspaces.* 1998.

73. Siméon, T., J.-P. Laumond, and C. Nissoux, *Visibility-based probabilistic roadmaps for motion planning.* Advanced Robotics, 2000. **14**(6): p. 477-493.

74. Burns, B. and O. Brock. *Single-query motion planning with utility-guided random trees*. in *Robotics and Automation, 2007 IEEE International Conference on*. 2007: IEEE.

75. Hsu, D., G. Sánchez-Ante, and Z. Sun. *Hybrid PRM sampling with a cost-sensitive adaptive strategy*. in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. 2005: IEEE.

76. Thomas, S., et al. *Biasing samplers to improve motion planning performance*. in *Robotics and Automation, 2007 IEEE International Conference on*. 2007: IEEE.

77. Guernane, R. and N. Achour. *An Algorithm for Generating Safe and Execution-Optimized Paths*. in *Autonomic and Autonomous Systems, 2009. ICAS'09. Fifth International Conference on*. 2009: IEEE.

78. Geraerts, R. and M.H. Overmars. *Clearance based path optimization for motion planning*. in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*. 2004: IEEE.

79. Sekhavat, S., et al., *Multilevel path planning for nonholonomic robots using semiholonomic subsystems.* The international journal of robotics research, 1998. **17**(8): p. 840-857.

80. Kim, J., R.A. Pearce, and N.M. Amato. *Extracting optimal paths from roadmaps for motion planning*. in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*. 2003: IEEE.

81. Nieuwenhuisen, D. and M.H. Overmars. *Useful cycles in probabilistic roadmap graphs*. in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*. 2004: IEEE.

82. Klasing, K., D. Wollherr, and M. Buss. *Cell-based probabilistic roadmaps (CPRM) for efficient path planning in large environments*. in *Proc. of the 2007 Int. Conf. on Advanced Robotics*. 2007.

83. Guernane, R. and N. Achour, *Generating optimized paths for motion planning.* Robotics and Autonomous Systems, 2011. **59**(10): p. 789-800.

84. Ratliff, N., et al. *CHOMP: Gradient optimization techniques for efficient motion planning*. in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. 2009: IEEE.

85. Lin, C., P. Chang, and J. Luh, *Formulation and optimization of cubic polynomial joint trajectories for industrial robots.* Automatic Control, IEEE Transactions on, 1983. **28**(12): p. 1066-1074.

86. Isto, P. *Constructing probabilistic roadmaps with powerful local planning and path optimization*. in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*. 2002: IEEE.

87. Kallmann, M., et al. *Planning Collision-Free Reaching Motions for Interactive Object Manipulation and Grasping*. in *Computer Graphics Forum*. 2003: Wiley Online Library.

88. Guernane, R. and M. Belhocine. *A smoothing strategy for PRM paths application to six-axes MOTOMAN SV3X manipulator*. in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. 2005: IEEE.

89. Hsu, D., J.-C. Latcombe, and S. Sorkin. *Placing a robot manipulator amid obstacles for optimized execution*. in *Assembly and Task Planning, 1999.(ISATP'99) Proceedings of the 1999 IEEE International Symposium on*. 1999: IEEE.

90. Overmars, M.H., *Recent developments in motion planning*, in *Computational Science—ICCS 2002*. 2002, Springer. p. 3-13.

91. Stilman, M., *Global manipulation planning in robot joint space with task constraints.* Robotics, IEEE Transactions on, 2010. **26**(3): p. 576-584.

92. Holleman, C. and L.E. Kavraki. *A framework for using the workspace medial axis in PRM planners*. in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*. 2000: IEEE.

93. Geraerts, R. and M.H. Overmars, *Creating high-quality paths for motion planning.* The international journal of robotics research, 2007. **26**(8): p. 845-863.

94. Jiménez, P., F. Thomas, and C. Torras, *Collision detection algorithms for motion planning*, in *Robot Motion Planning and Control*. 1998, Springer. p. 305-343.

95. Denavit, J., *A kinematic notation for lower-pair mechanisms based on matrices.* Trans. of the ASME. Journal of Applied Mechanics, 1955. **22**: p. 215-221.

96. Corke, P.I., *A robotics toolbox for MATLAB.* Robotics & Automation Magazine, IEEE, 1996. **3**(1): p. 24-32.

97. Paul, R.P. and H. Zhang, *Computationally efficient kinematics for manipulators with spherical wrists based on the homogeneous transformation representation.* The international journal of robotics research, 1986. **5**(2): p. 32-44.

98. Ames, A.L., E.M. Hinman-Sweeney, and J.M. Sizemore. *Automated generation of weld path trajectories*. in *Assembly and Task Planning: From Nano to Macro Assembly and Manufacturing, 2005.(ISATP 2005). The 6th IEEE International Symposium on*. 2005: IEEE.

7 APPENDIX

7.1 EXCEL SPREADSHEET DATA

An Excel spreadsheet forms a vital component for the setup phase of the AOLP system presented in Chapter 5. The primary purpose of the spreadsheet is to provide an interface for the user to enter setup information that is to be used by the systems algorithmic processes. The spreadsheet is divided into six tabs, a summary of each tab and its purpose are as follows:

- **Trunnion.** In this tab the user inputs data relating to the trunnion that the vehicle hull is mounted upon. The purpose of the trunnion is to orient the vehicle hull so the weld robots can perform their welding operations more easily. In this tab, the user is able to specify the number of different configurations the trunnion can orient itself in, and the corresponding rotational angle that these configurations are at. The trunnion's position in the workspace is also defined in this tab.

	A	B	C	D	E	F	G
1	TrunionInWorld	-12875.226	-5854.5	-1945.5	0	0	-90
2	TunionCenter	-5854.5	12875.2	1945	0	0	0
3	Orient 1~16A						
4	Toff	-7663	0.406	-1344.32	90	0	0
5	Orient 1	0	0	0	0	0	0
6	Orient 2	0	0	0	-47.7542	0	0
7	Orient 3	0	0	0	-68.3937	0	0
8	Orient 4	0	0	0	-89.5728	0	0
9	Orient 5	0	0	0	-99.4204	0	0
10	Orient 6	0	0	0	-131.526	0	0
11	Orient 7	0	0	0	-180	0	0
12	Orient 8	0	0	0	-228.474	0	0
13	Orient 9	0	0	0	-260.445	0	0
14	Orient 10	0	0	0	-297.182	0	0
15	Orient 11	0	0	0	-312.201	0	0
16	Orient 12	0	0	0	-330	0	0
17	Orient 8A	0	0	0	-249.068	0	0
18	Orient 5A	0	0	0	-116.463	0	0
19	Orient 4A	0	0	0	-79.5453	0	0
20	Orient 10A	0	0	0	-280.5	0	0
21	Orient 16A	0	0	0	-270.472	0	0
22							

Figure 7-1 The trunnion tab layout.

- **Sphere Definition**. In the AOLP system, the geometric model of each robot is modelled by spheres. In this tab the size and location of each sphere used to generate a simplified robot model is entered by the user, and the AOLP

algorithms can then read this information for use. In addition, a Visual Basic code can read this information to render the sphere representation in the DELMIA software environment. This is useful for setup purposes, as the full detail CAD model of the robot can be superimposed over the top of this simplified representation. This allows the user to check the suitability of the generated sphere model.

	A	B	C	D	E	F	G	H	I
1	Spheres attached on the robot for clash check								
2	# Spheres:	18	X	Y	Z	R			
3	1	Joint1	0	0	300	450			
4	1	Joint2	150	288	740	260			
5	1	Joint3	400	288	2040	250			
6	1	Joint4	0	288	2040	350			
7	1	Joint5	0	1188	2040	140			
8	1	Joint6	21	1438	2040	170			
9	1	T0	0	1438	2212	125			
10	0	Link1				0			
11	4	Link2				210			
12	0	Link3				0			
13	6	Link4				100			
14	0	Link5							
15	0	Link6							
16	1	wirefeeder				200			
17									
18									
19									
20									

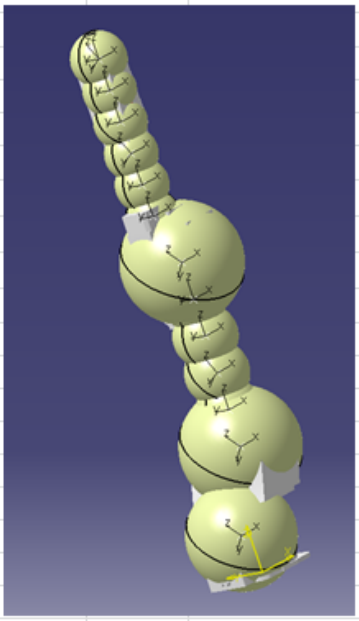


Figure 7-2 The sphere definition tab layout

- **Clash Objects**. For rapid collision processing, a simplified CAD representation of the plates making up the robot's environment was used. This tab is used to define these objects by simplifying the geometry of each plate into a series of vertexes to denote the plate's perimeter, and a thickness. Data is entered into this tab via the use of a Visual Basic code which interfaces with the DELMIA software package. The full model of the robot's workspace is loaded into DELMIA, and the user is then able to select the simplified geometry to use. This data is then automatically entered into this Excel tab for use in the AOLP environment. If necessary, the user can also manually enter or update the data in this tab as well.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Tot. Objects	189										
2	RefName	Name	Type	vertex	Depth	x	y	z	x	y	z	etc
3	Trunion	OPEN_SHELL #37572	plate	14	6	1820.924	2529.89	889.6689	6416.5	2529.894775	889.6689	6416.5
4	Trunion	OPEN_SHELL #37572	plate	17	6	6416.5	2534.92	-793.811	5836	2534.918457	-793.8113	5827.5
5	Trunion	OPEN_SHELL #31044	plate	5	8.5	1310.412	1654.87	1049.241	6416.5	1654.931885	1049.23	6416.5
6	Trunion	OPEN_SHELL #31044	plate	4	8.5	907.0759	1614.58	1013.447	1302.077	1654.436401	1048.874	1247.002
7	Trunion	OPEN_SHELL #39742	plate	5	8.5	5825.5	1654.91	-1049.23	1310.412	1654.873657	-1049.241	1255.177
8	Trunion	OPEN_SHELL #39742	plate	4	8.5	1302.077	1654.44	-1048.87	906.1848	1614.491211	-1013.367	861.4109
9	Trunion	OPEN_SHELL #11014	plate	28	8.75	8.5	1258.19	701.0157	536.7292	1311.458984	748.3964	536.7292
10	Trunion	OPEN_SHELL #19538	plate	22	8.75	5947.953	477.351	-6.53816	5432	477.3511658	-6.538161	5432
11	Trunion	OPEN_SHELL #25046	plate	4	8.5	6416.5	1373.09	-800	5834.25	1373.088745	-800	5834.25
12	Trunion	OPEN_SHELL #26284	plate	5	8.5	5834	1378.96	-798	5834	1378.566284	-799.0483	5834

Figure 7-3 The clash object definition tab

- **Seams**. This tab provides information about every seam that is to be welded by the robotic system, which is read and used by various AOLP algorithms. Each seam is defined with a name and coordinates representing the nominal start and end locations of the seam. This data is input into the spreadsheet via the use of a Visual Basic script. The user is also able to manually alter this data, if necessary.

Clipboard | Font | Alignment | Number

O18

	A	B	C	D	E	F	G	H	I	J
1	idx	Seam Name	x	y	z	Rx	Ry	Rz	Seam length	Zone
2	1	E01102A.1	1867.298	2570.331	-3.977	-67.46	85.76	-135	887.81	ex_LHF_UP
3		E01102A.2	1820.924	2523.903	-889.355	-67.46	85.76	-135		
4	2	E01102B.1	1867.343	2570.322	3.63	-112.54	-85.76	-135	888.14	ex_LHF_UP
5		E01102B.2	1820.925	2523.904	889.338	-112.54	-85.76	-135		
6	3	E01103A.1	1241.381	1963.485	-993.982479	117.485	-90	0	250.54	ex_LHF_UP
7		E01103A.2	1241.381	1963.485	-743.44114	117.485	-90	0		
8	4	E01103B.1	1241.381	1963.485	-720.55886	117.485	-90	0	337.12	ex_LHF_UP
9		E01103B.2	1241.381	1963.485	-383.44114	117.485	-90	0		
10	5	E01103C.1	1241.381	1963.485	-360.55886	117.485	-90	0	499.12	ex_LHF_UP
11		E01103C.2	1241.381	1963.485	138.5588605	117.485	-90	0		
12	6	E01103D.1	1241.381	1963.485	161.4411396	117.485	-90	0	833.18	ex_LHF_UP
13		E01103D.2	1241.381	1963.485	994.6221467	117.485	-90	0		
14	7	E01104.1	8.5	1742.713	-795	139.985	-90	0	274.43	ex_RHF_UP
15		E01104.2	8.5	1742.713	-520.569	139.985	-90	0		
16	8	E01105.1	8.5	1742.713	794.076	139.985	-90	0	321.66	ex_LHF_UP
17		E01105.2	8.5	1742.713	472.414	139.985	-90	0		
18	9	E01106.1	5827.5	2528.535	-800.971	45	86.998	-90	88.40	ex_RR_LHR
19		E01106.2	5827.5	2523.908	-889.253	45	86.998	-90		
20	10	E01107A.1	962.0868	1914.399	998.9034971	130.102	-4.389	9.969	145.00	ex_LHF_UP
21		E01107A.2	1104.479	1939.425	1010	130.102	-4.389	9.969		
22	11	E01107B.1	1106.59	1939.795	982.082111	-32.601	4.389	-170.031	145.00	ex_LHF_UP
23		E01107B.2	964.1972	1914.77	970.9856081	-32.601	4.389	-170.031		

Figure 7-4 The seams tab layout

- **Zone**. Due to the complexity of the robot placement problem a number of predefined volumes of space are used to provide a reduced search area to simplify the problem. This tab provides information about these zones for the AOLP system, including the location and volume of each zone, and the number

of tags contained within the zone. The orientation of these tags is also required, in the standard Euler format. All data in this tab is entered manually by the user.

	A	B	C	D	E	F	G
1	Sequence 1 LHR						
2	WS07LHR	X	Y	Z	Rx	Ry	Rz
3	UpLeftCorner	300	5500	-1300	-90	0	-90
4	DownRightCorner	-1100	8200	-500			
5	numOfGrids	7	7	7			
6	TotalGrids	343					
7							
8	FG07	X	Y	Z	Rx	Ry	Rz
9	UpLeftCorner	360	7448	278	0	90	-90
10	DownRightCorner	-389	8187	-284			
11	numOfGrids	7	7	7			
12	TotalGrids	343					
13							
14	RD07	X	Y	Z	Rx	Ry	Rz
15	UpLeftCorner	-275	2000	290.3	-90	90	0
16	DownRightCorner	274.6	300	-1130			
17	numOfGrids	7	7	7			
18	TotalGrids	343					
19	Sequence 1 RHR						
20	ex_RR_RHR	X	Y	Z	Rx	Ry	Rz
21	UpLeftCorner	-300	1500	1350	0	60	180
22	DownRightCorner	1600	0	-500			
23	numOfGrids	7	7	7			
24	TotalGrids	343					

Figure 7-5 The zone tab layout

7.2 GENERATED CODE EXAMPLE

The robot code file generated from the AOLP system for the example seam programmed in Section 5.6.7 is displayed in this appendix. The code is split into two parts, one for the weld robot and one for the auxiliary robot, which are presented in the separated sections below.

WELD ROBOT CODE

```
.PROGRAM sequence1() ;sequence code
        wire = 1
        SPEED fast ALWAYS
        ACCURACY small ALWAYS
        weldID = 19
        SIGNAL 111,112
        CALL do.weld(19,0)
        SWAIT 1097 ; preheat completed & orientation OK
        CALL zone("request",1,1)
        CALL a_ent_19 ;fs enter code
        CALL aN07101A ;fs main code
        CALL a_exit_19 ;fs exit code
        CALL zone("release",1,0)
.END ;sequence code

.PROGRAM aN07101A() ;fs main code
        weldID = 19
        weldnum = 1229
        fronius_job = 13
        weld_speed = 45
        joint_type = 10
        ;
        ;temperature check
        SIGNAL -97 ; Reset at tempcheck position
        WAIT SIG(1097) OR SIG(1098) ; preheat completed & orientation OK
        IF SIG(1110) OR (wld[19]<>0) GOTO bypassed
        CALL get.base ; update tool data with ZX
```

```
BREAK
TOOL torch ; assign tool
JMOVE #tempcheck[1229]
BREAK
SIGNAL 97 ; at temperature check position
WAIT (SIG(1108) OR SIG(1109) OR SIG(1112)) ; WAIT for plc to check
IF SIG(1112) THEN
        wld[19] = 0 ; set weld as fail
        GOTO tempfail_high
END ; IF
IF SIG(1109) THEN
        wld[19] = 0 ; set weld as fail
        SIGNAL 140 ; weld unavailable, optional
        SWAIT 1140 ; PLC acknowledged, optional
        SIGNAL -140 ; reset, optional
        GOTO tempfail_low
END ; IF
IF SIG(1109) GOTO tempfail_low
SWAIT 1108 ; temperature is ok
SIGNAL -97
;
JMOVE #aN07101A_T2[2]
JMOVE #aN07101A_T2[3]
;
JAPPRO search[1229,1],100
LMOVE search[1229,1]
BREAK
CALL get.fillet(1229,1,10,&search[1229,1])
BREAK
LMOVE search[1229,2]
BREAK
CALL get.fillet(1229,2,10,&search[1229,2])
BREAK
LDEPART 100
```

```
;
JMOVE #aN07101A_T3[2]
;
JAPPRO search[1229,3],100
LMOVE search[1229,3]
BREAK
CALL get.fillet(1229,3,27,&search[1229,3])
BREAK
LMOVE search[1229,4]
BREAK
CALL get.fillet(1229,4,27,&search[1229,4])
BREAK
LMOVE search[1229,5]
BREAK
CALL get.edge(1229,5,&s[1229,5],27,&search[1229,5])
BREAK
LDEPART 100
; weldPath calculation
CALL
intersection(&wcorner[1229,1],&fillet[1229,1],&fillet[1229,2],&fillet[1229,3],
&fillet[1229,4])
;
CALL get.seven(&search[1229,1])
CALL get.weld(1229,1,&fillet[1229,1],&orient[1229,1],trav)
CALL get.weld(1229,2,&fillet[1229,3],&orient[1229,2],trav)
CALL get.weld(1229,3,&fillet[1229,4],&orient[1229,3],trav)
CALL get.weld(1229,4,&wcorner[1229,5],&orient[1229,4],trav)
;
POINT weld[1229,1] = weld[1229,1]+ TRANS(0,0,0,,,,trav)
POINT weld[1229,2] = weld[1229,2]+ TRANS(0,0,0,,,,trav)
POINT weld[1229,3] = weld[1229,3]+ TRANS(0,0,0,,,,trav)
POINT weld[1229,4] = weld[1229,4]+ TRANS(0,0,0,,,,trav)
BITS 161,15 = fronius_job
; W1SET 1 = 100,21,2,2,1
; weld motions
```

```
        JAPPRO weld[1229,1],100
        SIGNAL 99 ;weld in operation
        LWC weld[1229,2],1
        LWC weld[1229,3],1
        LWE weld[1229,4],1,
        SWAIT -arcest
        SIGNAL -99 ;weld in operation
        weldcompleted = weldcompleted+1 ; increment number of completed welds
        wld[19] = 1 ;weld completed
        ;
        tempfail_low:
        bypassed:
        tempfail_high:
        IF SIG(1110) THEN
                SIGNAL 110 ; bypass completed
        END ; IF
        IF SIG(1110) AND wld[19] == 0 THEN ; bypassed
                weldcompleted = weldcompleted+1
                wld[19] = 1 ;weld completed
        END ; IF:
        SIGNAL -97 ; Reset at tempcheck position:
        RETURN
.END ;fs main code
;
.PROGRAM a_ent_19() ;fs enter code
        SPEED 50
        JMOVE #aN07101A_a[1]
        CALL zx.move(1)
        JMOVE #aN07101A_a[2]
        JMOVE #aN07101A_a[3]
        RETURN
.END ;fsenter code

.PROGRAM a_exit_19() ;fs exit code
```

```
        SPEED 50
        JMOVE #aN07101A_r[2]
        JMOVE #aN07101A_r[3]
        CALL zx.move(2)
        RETURN
.END ;fsexit code

.JOINTS ;joint angles of each waypoint
        #tempcheck[1229] -55.96 -1.73 -111.06 -111.91 -43.24 -151.10 2766.93
        #aN07101A_T2[1] -31.43 -2.19 -137.05 105.55 -47.00 -133.39 2766.93
        #aN07101A_T2[2] -20.61 -3.43 -130.88 91.38 -54.00 -122.37 2766.93
        #aN07101A_T2[3] -56.66 -3.86 -121.12 -107.64 -42.30 -153.34 2766.93
        #aN07101A_T3[1] -56.66 -2.19 -67.40 -54.48 -52.00 -225.82 2766.93
        #aN07101A_T3[2] -55.12 10.73 -41.00 -41.36 -67.68 -245.68 2766.93
        #aN07101A_a[1] -1.00 -20.00 -145.00 -1.00 -41.00 -145.00 2766.93
        #aN07101A_a[2] -58.14 -9.33 -117.86 -109.66 -44.88 -153.25 2766.93
        #aN07101A_a[3] -55.96 -1.73 -111.06 -111.91 -43.24 -151.10 2766.93
        #aN07101A_r[1] -55.57 14.57 -128.59 -129.47 -54.30 -122.21 2766.93
        #aN07101A_r[2] -57.40 -11.06 -105.47 -74.59 -42.09 -195.05 2766.93
        #aN07101A_r[3] -1.00 -20.00 -145.00 -1.00 -41.00 -145.00 2766.93
.END ;JOINTS

.TRANS ; cartesian coordinates of each waypoint
        search[1229,1] -46.86 -804.64 -798.52 -44.56 97.05 97.11 2766.93
        search[1229,2] -46.86 -839.37 -601.56 -44.56 97.05 97.11 2766.93
        search[1229,3] -368.24 -774.98 -923.37 -75.10 137.53 -168.90 2766.93
        search[1229,4] -680.24 -774.98 -923.37 -75.10 137.53 -168.90 2766.93
        search[1229,5] -984.24 -774.98 -923.37 -75.10 137.53 -168.90 2766.93
        weld[1229,1] -1014.64 -809.15 -962.79 -61.22 134.73 -158.86 2766.93
        weld[1229,2] -710.66 -809.15 -962.79 -61.22 134.73 -158.86 2766.93
        weld[1229,3] -398.68 -809.15 -962.79 -61.22 134.73 -158.86 2766.93
        weld[1229,4] -94.64 -809.15 -962.79 -75.10 137.53 -168.90 2766.93
        orient[1229,1] -1014.64 -809.15 -962.79 -61.22 134.73 -158.86 2766.93
        orient[1229,2] -710.66 -809.15 -962.79 -61.22 134.73 -158.86 2766.93
```

```
        orient[1229,3] -398.68 -809.15 -962.79 -61.22 134.73 -158.86 2766.93
        orient[1229,4] -94.64 -809.15 -962.79 -75.10 137.53 -168.90 2766.93
        wcorner[1229,1] -1014.64 -809.15 -962.79 -61.22 134.73 -158.86 2766.93
        s[1229,3] -944.24 -774.98 -923.37 -75.10 137.53 -168.90 2766.93
.END ;TRANS
```

AUXILIARY ROBOT CODE

```
.PROGRAM sequence1() ;zx sequence code
        DELAY 0; display update
        loop:
        SPEED fast ALWAYS
        ACCURACY small ALWAYS
        SIGNAL 100; Set @ pose signal
        WAIT SIG(1100) OR SIG(1107);FA request pose change or sequence complete
        SIGNAL -100; Reset @ pose signal
        GOTO exit IF SIG(1107); FA MIG has completed sequence
        WAIT BITS(1101,5)<>0; FA has updated pose
        position = BITS(1101,5); assign pose data
        BITS 101,5 = position; echo back pose data
        SWAIT -1100;FA has rcvd data
        WAIT SIG(1100) OR SIG(1106); data not ok or ok
        GOTO loop IF SIG(1100); data check failed
        SWAIT 1106;data ok
        BITS 101,5 = 0; reset echo data
        CASE position OF
        VALUE 1:
        CALL a_ent_19 ;zx enter code
        VALUE 2:
        CALL a_exit_19 ;zx exit code
        ANY :
        ;error
        PRINT 2: "Incorrect number sent for this sequence"
        PULSE 123,5 ;set number error
```

```
        PAUSE
        SIGNAL -123; reset number error
        END
        SIGNAL 100; ZX @ new position
        GOTO loop
        exit:
        RETURN
.END ;sequence code
;
.PROGRAM a_ent_19() ;zx enter code
        JMOVE #aN07101A_a[1]
        JMOVE #aN07101A_a[2]
        LMOVE #aN07101A_a[3]
        JMOVE #aN07101A_a[4]
        JMOVE #aN07101A_a[5]
        RETURN
.END
;
.PROGRAM a_exit_19() ;zx exit code
        JMOVE #aN07101A_a[5]
        JMOVE #aN07101A_a[4]
        LMOVE #aN07101A_a[3]
        JMOVE #aN07101A_a[2]
        JMOVE #aN07101A_a[1]
        RETURN
.END
;
.JOINTS
        #aN07101A_a[1] 1.00 -55.00 -17.00 -1.00 108.00 -180.00
        #aN07101A_a[2] -1.87 8.12 -69.62 1.89 99.42 -182.02
        #aN07101A_a[3] 22.91 36.47 -60.44 -64.18 76.13 -155.94
        #aN07101A_a[4] -2.42 67.97 -23.48 -85.29 77.00 -134.91
        #aN07101A_a[5] -8.43 74.74 -17.24 -87.48 81.95 -107.42
.END ;JOINTS
```

7.3 ALGORITHMIC DETAILS

7.3.1 Coordinate Systems

To completely specify the location of an object we must define both its position and orientation with a suitable coordinate system. In a 3D environment, the position of an object is almost always described using the [X,Y,Z] format of the Cartesian coordinate system. However, describing the orientation of an object in 3D space is more complex. There are also a number of different ways to do this, the most common being the Euler and Quaternion representations.

In the AOLP system, the position and orientation of an object is defined with a homogenous coordinate transformation matrix, which is made up of four sub-matrices as shown below:

$$T = \begin{bmatrix} R & p \\ \eta^T & \sigma \end{bmatrix}$$

Rotation, Translation, Perspective, Scale

The sub-matrix R is a 3×3 *rotation matrix,* which represents the orientation of the objects mobile coordinate frame $\widehat{M_f}$, relative to the global reference frame. The *translation vector* p is a 3×1 column vector representing the position of the origin on $\widehat{M_f}$ relative to the global reference frame. The scalar value σ is a non-zero scale factor, which, in our application, is always set to one. Finally, the *perspective vector* η^T is used to view a transformation from a different perspective, which is set to [0,0,0] (first person).

As shown in Figure 7-6, each object is assigned its own coordinate frame, referred to as the mobile frame $\widehat{M_f} = [\vec{u}, \vec{v}, \vec{w}]$. The location and orientation of the mobile frame $\widehat{M_f}$ is described with the aforementioned homogenous coordinate transformation matrix T. The orientation of $\widehat{M_f}$ is conveyed through the rotation sub-matrix of T, which contains the unit vectors $\vec{u} = [u_1, u_2, u_3]$, $\vec{v} = [v_1, v_2, v_3]$ and $\vec{w} = [w_1, w_2, w_3]$. These three unit vectors represent how each of the three principle axes of $\widehat{M_f}$ is orientated relative to the

global coordinate frame. The fourth column displays the Cartesian coordinates $[X,Y,Z]^T$ of the origin of M_f relative to the same global origin.

$$T = \begin{bmatrix} u_1 & v_1 & w_1 & X \\ u_2 & v_2 & w_2 & Y \\ u_3 & v_3 & w_3 & Z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

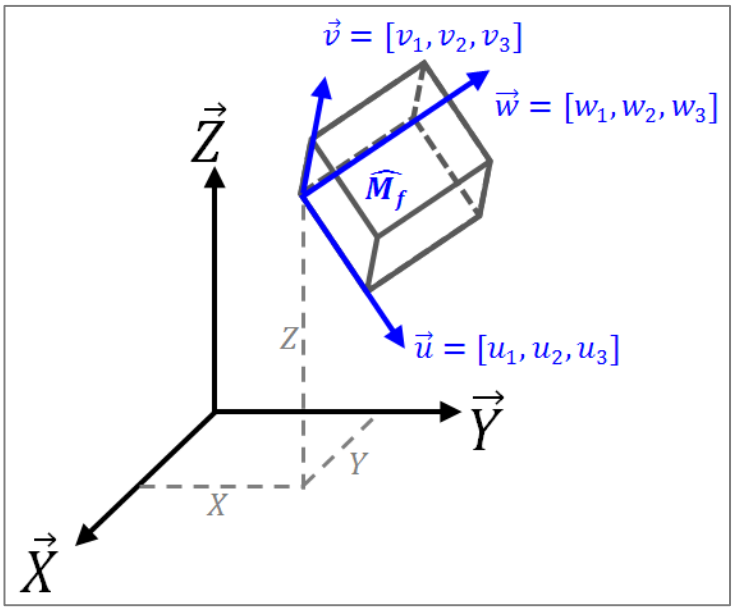


Figure 7-6 Coordinate frame of an object $\widehat{M_f}$ positioned relative to a global (or fixed) frame

Orientation is typically defined using three separate rotations about the global reference frame. These rotations are known as Euler angles. First, $\widehat{M_f}$ is rotated through R_x degrees about the Z-axis, followed by a rotation about the y-axis of R_y degrees, followed by a rotation about the x-axis of R_z degrees. Using this method, when rotated using the $[R_x, R_y, R_z]$ convention, the values of the rotation matrix R of T becomes: R_z

$$R([R_x, R_y, R_z]) = \cdots$$

$$\begin{bmatrix} cos\, R_y\, cos\, R_x & sin\, R_z\, sin\, R_y\, cos\, R_x - cos\, Rz\, sin\, R_x & cos\, R_z\, sin\, R_y\, cos\, R_x - sin\, R_z\, sin\, R_x \\ cos\, R_y\, sin\, R_x & sin\, R_z\, sin\, R_y\, sin\, R_x - cos\, Rz\, cos\, R_x & cos\, R_z\, sin\, R_y\, sin\, R_x - sin\, R_z\, cos\, R_x \\ -\, sin\, R_y & sin\, R_z\, cos\, R_y & cos\, R_z\, cos\, R_y \end{bmatrix}$$

As matrix multiplication is not commutative (i.e for the matrices A and B: $AB \neq BA$), the order in which the Euler rotations is carried out has an effect on the resulting rotation made. Another drawback to Euler angle representation is that a particular orientation will not have a unique form. Two different Euler angles can produce a

physically identical orientation, which can cause errors while trying to compare two equivalent coordinates. The major benefit in using the coordinate transformation matrix approach to represent the position and orientation of an object is that these matrices are conveniently configured for computation of geometric translation and rotation.

7.3.2 Conversion Between Different Coordinate Systems

As discussed, there are a range of methods that can be used to define an objects orientation in 3D space. As each method has its own particular advantages and disadvantages, different robot manufacturers and OLP software packages utilise different representation methods. In order to be compatible with as many different platforms as possible, several algorithmic functions that are able to make a conversion between the following coordinate representations have been developed for this works specific application:

1. Position and Quaternion: This is the simplest way of uniquely representing a 3D coordinate. Quaternion methods, whilst are computationally simple, are unintuitive to human perception. This means they are typically only used in computational processes that do not involve information transfer to a human.
2. Homogenous coordinate transformation matrix (presented in Section 3.4.1): A 4x4 matrix which stores data relating to position and orientation in terms of unit vectors which represent the coordinate frame of the object, relative to a global coordinate frame. The matrix format used allows most convenient representation for the calculation of geometric translation and rotation by way of matrix multiplication.
3. ZYX Euler angles. The Euler angle format is intuitive to understand, making it popular when human interaction in involved as a human can use the Euler description to directly visualise an objects orientation. Euler angles are the most common coordinate system used in OLP software interfaces, such as DELMIA.
4. ZYZ Euler angles. Same as above, however order of rotation is different (rotate about Z-axis, then Y-Aaxis, then the Z-Axis again. Kawasaki robots make use of this method.

5. XYZ fixed angles. This is similar to Euler angle, while the axis of rotation is fixed during transformation. This definition is convenient and intuitive to define torch orientation, such as push angle, during tool placement step.

7.3.3 Sphere-Sphere Collision Processing

The most computationally simple distance calculation involved in the AOLP systems collision processing algorithms involves calculating the clearance C_l between two spheres in 3D Cartesian space. C_l is evaluated as the difference between the Euclidian distance D between the two sphere centres $p_1 = [x_1, y_1, z_1]$ and $p_2 = [x_2, y_2, z_2]$, and the sum of their respective radii r_1 and r_2:

$$C_l = D - (r_1 + r_2) = \left(\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}\right) - (r_1 + r_2)$$

As shown in Figure 7-7 below, if the clearance is found to be less than or equal to zero, the spheres are deemed to be in collision. Otherwise, we can assume that their boundaries are not intersecting.

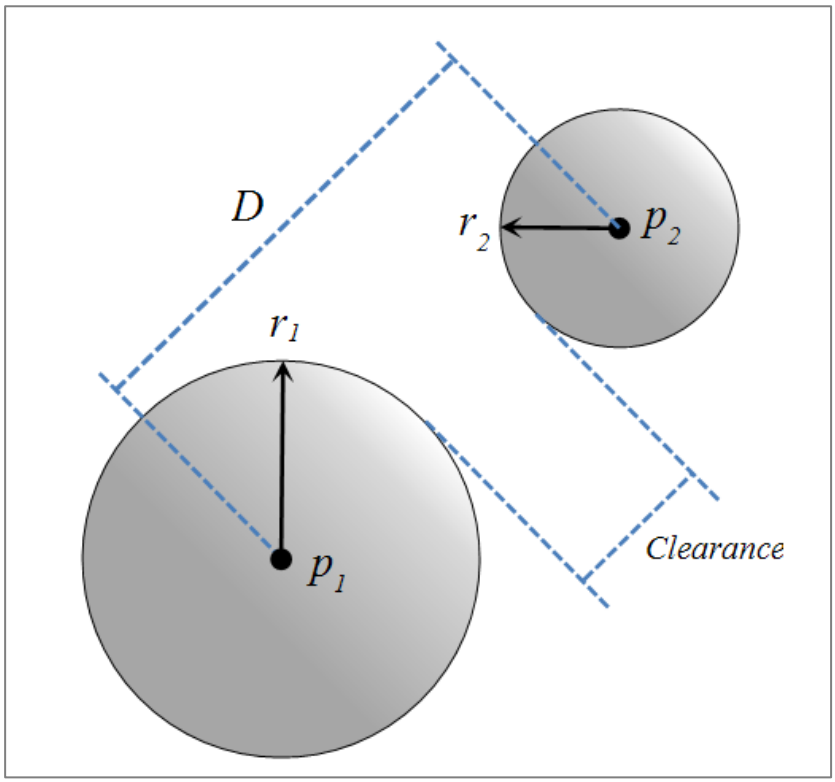


Figure 7-7 Clearance between two spheres.

7.3.4 Sphere-Plate Collision Processing

Calculating the clearance between a sphere and a plate in 3D space is a more complicated process. Ultimately, the clearance between these two objects is calculated as either:

1. The distance between the sphere and the nearest edge of the plate, or
2. The distance between the sphere and the plane at the surface of the plate

In order to logically determine the correct method to use in calculating the clearance, a vector-based homogenous transform is performed to map the centre of the sphere p_1 onto the plane created at the surface of the plate. Once done, we are presented with two distinct possibilities, as shown in Figure 7-8:

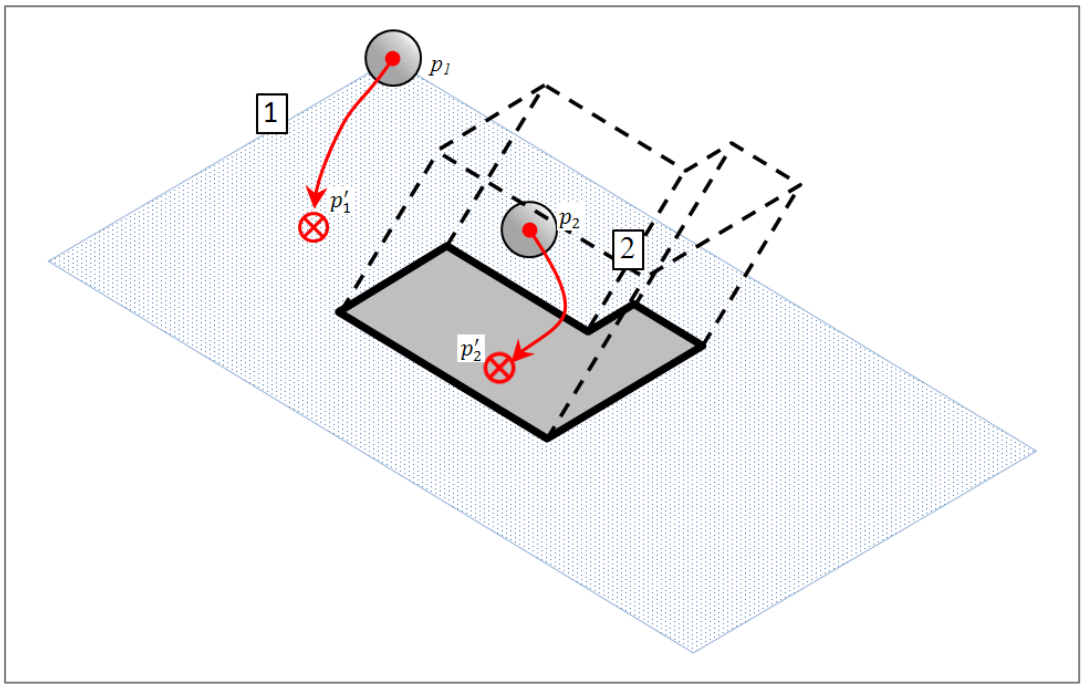


Figure 7-8 Transforming sphere centres onto the plane of a plate.

- In situation 1, depicted on the left hand side of Figure 7-8, we see that the newly transformed sphere centre p_1' lies outside of the polygon representing the perimeter of the plate. In this instance, the minimum distance from the original sphere centre p_1 to the plate will be calculated as the distance from p_1 to the nearest edge of the plate (refer to Section 7.3.4.2 below).
- In situation 2, depicted to the right of Figure 7-8, the newly transformed sphere centre p_2' lies inside the polygon representing the perimeter of the plate. In this instance, the minimum distance from the original sphere center p_2 and the plate will be calculated as the distance from p_2 to the plane at the surface of the plate (refer to Section 7.3.4.1 below).

MATLAB provides an inbuilt function 'inpolygon' which is able to rapidly query whether a point lies inside or on the boundary of a two dimensional polygon. Inpolygon uses the winding number approach, which provides a rapid and effective means of calculation.

Once either of the two situations is identified, then the proper clearance calculation can be applied, both methods of performing this calculation are presented in sections 7.3.4.1 and 7.3.4.2 below.

7.3.4.1 *Point-Plane Distance*

The general problem is presented in Figure 7-9 below. To find the distance between a given point $\boldsymbol{w} = (x_0, y_0, z_0)$ and the plane $a\,x + b\,y + c\,z + d = 0$, we define a vector $\vec{\boldsymbol{v}}$ between the plane and the point $\boldsymbol{w}$:

$$\vec{\boldsymbol{v}} = \begin{bmatrix} x - x_0 \\ y - y_0 \\ z - z_0 \end{bmatrix}$$

If the plane has a normal vector $\vec{\boldsymbol{n}} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$, then the distance D between the point and the plane can then be calculated by projecting $\vec{\boldsymbol{n}}$ onto $\vec{\boldsymbol{v}}$:

$$D = \frac{\vec{v} \cdot \vec{n}}{|\vec{v}|} = \frac{a\,x_0 + b\,y_0 + c\,z_0 + d}{\sqrt{a^2 + b^2 + c^2}} \quad (1)$$

The sign of D indicates which side of the plane the sphere is located on: A positive value indicates the sphere centre is located on the same side as the normal, a negative indicates it is on the opposite side.

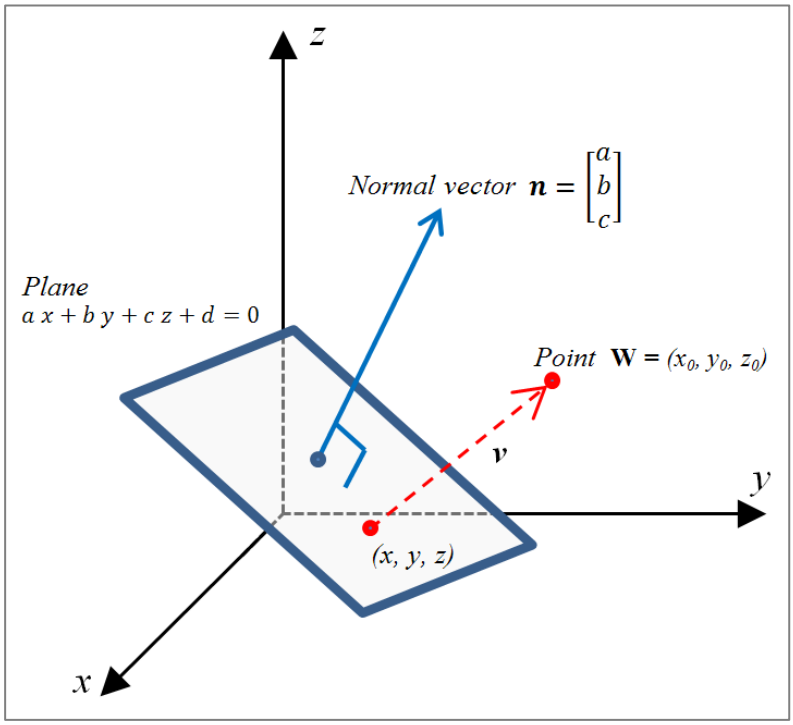


Figure 7-9 Distance between a point and a plane.

Once D has been evaluated, calculating the clearance C_l between the sphere and the plate is performed by computing the difference between D and the radius r of the sphere:

$$C_l = D - r$$

If the value of D from equation (1), is found to be negative, then the thickness of the plate t must be factored into the clearance as the sphere is located on the reverse side of the plate:

$$C_l = -D - r - t$$

7.3.4.2 *Point-Line Distance*

In the situation where the sphere will be closest to one of the plate's edges, we perform a point-line distance calculation for each edge of the plate. The edge which features the minimum distance is then used to calculate the final clearance between the sphere and the plate.

The general problem is presented in Figure 7-10 below. To find the distance between a given point $\boldsymbol{w} = (x_0, y_0, z_0)$ and the finite line formed between two points $\boldsymbol{p_1} = (x_1, y_1, z_1)$ and $\boldsymbol{p_2} = (x_2, y_2, z_2)$ we use a vector-based approach. First we define two vectors:

$$\overrightarrow{\boldsymbol{v_1}} = \boldsymbol{w} - \boldsymbol{p_1} = \begin{bmatrix} x_0 - x_1 \\ y_0 - y_1 \\ z_0 - z_1 \end{bmatrix}$$

$$\overrightarrow{\boldsymbol{v_2}} = \boldsymbol{p_1} - \boldsymbol{p_2} = \begin{bmatrix} x_1 - x_2 \\ y_1 - y_2 \\ z_1 - z_2 \end{bmatrix}$$

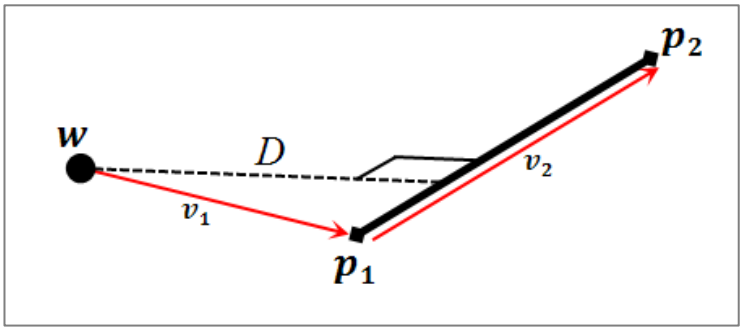


Figure 7-10 Distance between a point w and the line formed between p_1 and p_2.

A scalar projection of $\overrightarrow{\boldsymbol{v_1}}$ onto $\overrightarrow{\boldsymbol{v_2}}$ is performed, where the resultant scalar $\boldsymbol{s}$ can be calculated as:

$$\boldsymbol{s} = |v_1| cos\theta \qquad (2)$$

Since we are not interested in calculating the angle θ between $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$, we can use the relation: $\frac{\overrightarrow{v_1} \cdot \overrightarrow{v_2}}{|\overrightarrow{v_1}||\overrightarrow{v_2}|} = cos\theta$ to simplify equation (2):

$$\boldsymbol{s} = |\overrightarrow{\boldsymbol{v_1}}| cos\theta = |\overrightarrow{\boldsymbol{v_1}}| \frac{\overrightarrow{\boldsymbol{v_1}} \cdot \overrightarrow{\boldsymbol{v_2}}}{|\overrightarrow{\boldsymbol{v_1}}||\overrightarrow{\boldsymbol{v_2}}|} = \frac{\overrightarrow{\boldsymbol{v_1}} \cdot \overrightarrow{\boldsymbol{v_2}}}{|\overrightarrow{\boldsymbol{v_2}}|}$$

Where $\overrightarrow{v_1} \cdot \overrightarrow{v_2}$ is the dot product of the two vectors, and $|\overrightarrow{v_2}|$ represents the magnitude.

As shown in Figure 7-11, three distinct situations arise from the evaluation of the scalar $\boldsymbol{s}$:

1. As depicted in situation 1 of Figure 7-11 if $\boldsymbol{s} < 0$ the closest part of the line to $\boldsymbol{w}$ will be the end-point $\boldsymbol{p_1}$. In this situation the distance between the line and $\boldsymbol{w}$ is calculated as the Euclidean distance between them:

$$D = distance(\boldsymbol{w}, \boldsymbol{p_1}) = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2 + (z_0 - z_1)^2}.$$

2. In similar fashion, if the value of s is greater than the length of $\boldsymbol{v_2}$, as shown in situation 3 of Figure 7-11, then the closest part of the line to $\boldsymbol{w}$ will be the end-point $\boldsymbol{p_2}$. In this situation the distance between the line and $\boldsymbol{w}$ is calculated as the Euclidean distance between them:

$$D = distance(\boldsymbol{w}, \boldsymbol{p_2}) = \sqrt{(x_0 - x_2)^2 + (y_0 - y_2)^2 + (z_0 - z_2)^2}.$$

3. Finally, as shown in situation 2 of Figure 7-11, if $0 \leq s \leq distance(\boldsymbol{p_1}, \boldsymbol{p_2})$, then the closest part of the line to $\boldsymbol{p_1}$ will be an arbitrary point which lies somewhere on the vector $\boldsymbol{v_2}$. The distance from $\boldsymbol{w}$ to this arbitrary point is found with the cross product of $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$. It is important to note that the magnitude of the resultant cross product will be equal to the area of a parallelogram spanned by $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$, which allows us to say:

$$area(parallelogram) = |\boldsymbol{v_2}| * distance(\boldsymbol{w}, \boldsymbol{v_2})$$
$$\therefore |\overrightarrow{\boldsymbol{v_1}} \times \overrightarrow{\boldsymbol{v_2}}| = |\overrightarrow{\boldsymbol{v_2}}| * distance(\boldsymbol{w}, \boldsymbol{v_2})$$

$$\therefore\ D = distance(\boldsymbol{w}, \boldsymbol{v_2}) = \frac{|\overrightarrow{\boldsymbol{v_1}} \times \overrightarrow{\boldsymbol{v_2}}|}{|\overrightarrow{\boldsymbol{v_2}}|}$$

Where $\times$ represents a vector cross product, and $|\overrightarrow{v_2}| * |\overrightarrow{v_3}|$ represents the multiplication of the magnitudes of $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$. The cross product $|\overrightarrow{\boldsymbol{v_1}} \times \overrightarrow{\boldsymbol{v_2}}|$ is solved using inbuilt MATLAB functions.

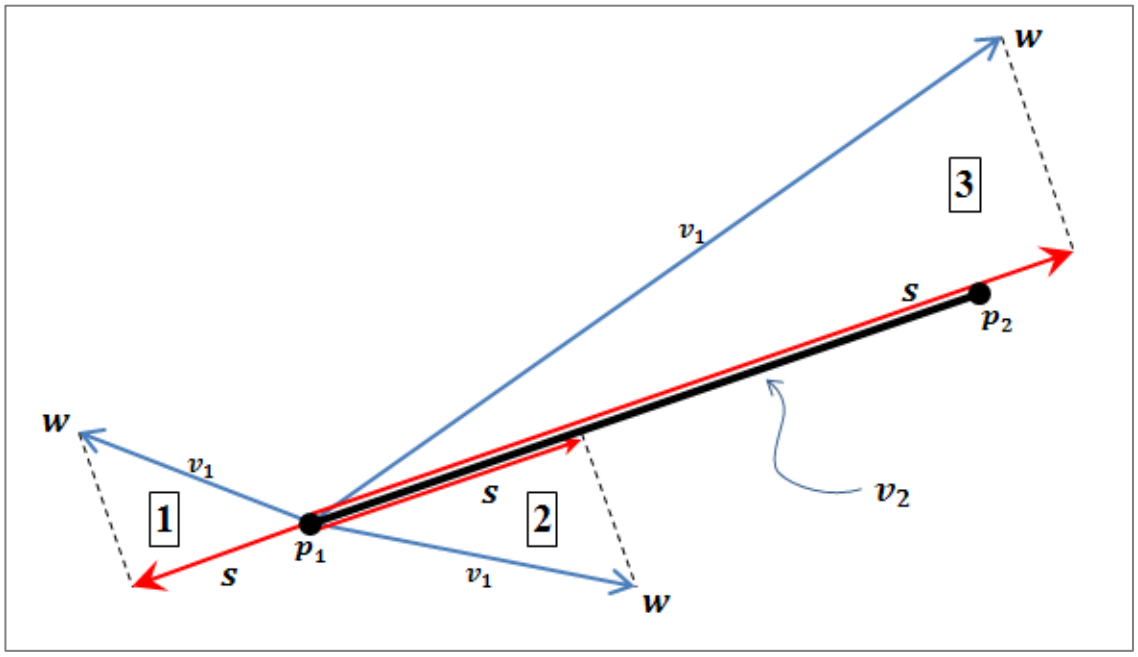


Figure 7-11 The scalar projection s of v_1 onto v_2 can have three different results (red arrows).

Once the plate edge which shares the minimum distance D to $\boldsymbol{w}$ has been found, we calculate the clearance C_l between the sphere and the edge as the difference between D and the radius r of the sphere:

$$C_l = D - r$$

7.3.5 Generation of Seam Geometry

Given the local coordinate systems for the seam $\hat{C}_{seam} = \{\vec{v}_x^S, \vec{v}_y^S, \vec{v}_z^S\}$, and for the torch $\hat{C}_{Torch} = \{\vec{v}_x^T, \vec{v}_y^T, \vec{v}_z^T\}$, we need to generate $\hat{C}_{seam}$ in a way that will orient $\hat{C}_{Torch}$ so it positions the torch model in a suitable orientation relative to the two plates which form

the weld seam. The torch frame $\hat{C}_{Torch}$ is predefined, so the problem involves generating $\hat{C}_{seam}$ appropriately.

As presented previously in Section 5.5.3, Visual Basic is used in conjunction with DELMIA to efficiently gather geometric information about the plates that form the weld seam in question. Figure 7-12 depicts the three vectors acquired from geometric data during this process, which are then used to appropriately define $\hat{C}_{seam}$:

- $\vec{v}_{edge}$: the unit vector which lies on the edge formed by the intersection of the two plates being welded together
- $\vec{v}_{n1}$: The normal vector of one plate that forms the weld seam
- $\vec{v}_{n2}$: The normal vector of the other plate that forms the weld seam

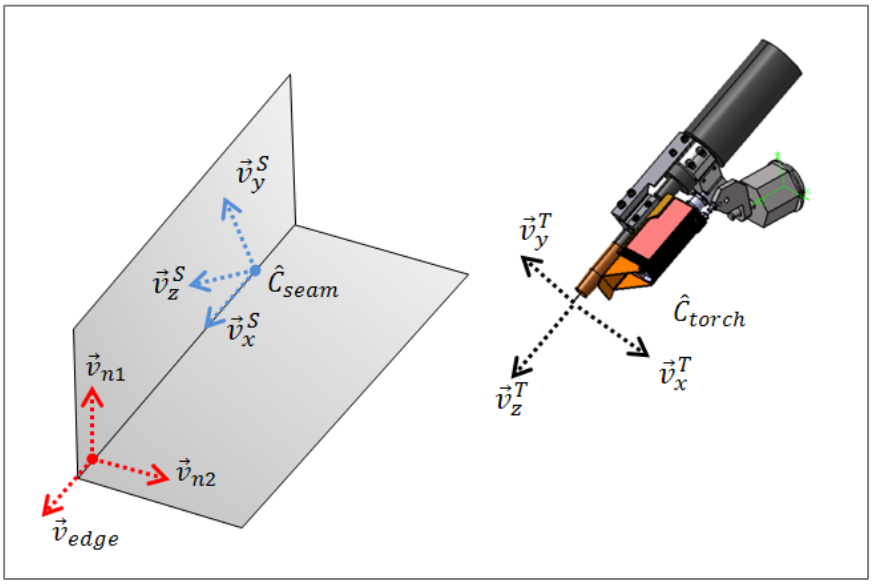


Figure 7-12 Vector information used to define coordinate systems that orientate the weld torch.

With these three vectors $\hat{C}_{seam}$ is generated by first defining $\vec{v}_x^S$, which will guide the torch from the start of the seam to the end. This is done by simply defining it as the direction of the edge formed by the intersection of the two plates: $\vec{v}_x^S = v_{edge}$. Then, $\vec{v}_z^S$ is generated by subtracting the two normal face vectors from each other: $\vec{v}_z^S = -\vec{v}_{n1} - \vec{v}_{n2}$. Finally, $\vec{v}_y^S$ is generated from the cross product of $\vec{v}_x^S$ and $\vec{v}_z^S$: $\vec{v}_y^S = \vec{v}_x^S \times \vec{v}_z^S$. Once these calculations are performed, the resultant $\hat{C}_{seam}$ can be used in conjunction with $\hat{C}_{Torch}$ to orientate the torch in order to properly position it relative to the plates that

make up the weld seam. This is done by using homogenous coordinate transformation (refer to Section 3.4.1) between the two generated reference frames and then positioning the newly orientated coordinate frame of the torch model at each end of the seam.

7.4 PUBLICATIONS

Offline Programming for a Complex Welding System using DELMIA Automation

Joseph Polden, Zengxi Pan, Nathan Larkin, Stephen Van Duin, John Norrish
Faculty of Engineering, University of Wollongong, Wollongong, NSW, 2530, Australia
Email: zengxi@uow.edu.au Tel: 612 4221 5498

***Abstract*—This paper presents an offline programming (OLP) system for a complex robotic welding cell using DELMIA Automation. The goals of this research are aimed at investigating the feasibility of taking a commercially available robotic simulation package, DELMIA, and to use a Visual Basic Automation interface to reduce the programming time by creating automated 'modules' to carry out some of the tasks in the OLP process. The paper first investigates and presents the structure of OLP as a discreet method of individual steps. These steps are then evaluated for their potential as an automation candidate. The methods in which these steps are automated are then presented. A general analysis of the developed OLP system was carried out, providing a scope for future research and development**

Keywords: DELMIA Automation, offline programming, robotic welding

I. Introduction

A manufacturing facility in Australia has, over the last number of years, been tasked with handling the manufacture of an automobile hull. The vehicle hull is composed of weldable steel plates, and is constructed in a monocoque type assembly with all of the various plates that make up the hull being welded together by the same way as a ship vessel. At the time of writing, the welding of the hull was being carried by manual processes alone; however an increase in future production demand has pushed the manufacturing facility to automate the welding processes on the hull via implementation of a complex robotic welding workcell.

Due to the high number of seams to be welded and the complex geometry which is inherent in the hull's design; a specialised robotic cell was needed to be created in order to maximise the number of external and internal seams that could be completed by the cell. This cell consists of two 6-DOF articulated welding robots. Each of these robots is then in-turn mounted on another, larger, 6-DOF 'auxiliary' robot and linear rail to create a form of homogenised 13-DOF robot, as shown in *Figure 1*. The linear rail is utilised to allow the robots mobility about the stationary hull that is to be welded. The hull itself is also mounted on a rotating trunnion to allow the homogenised welding robot access to areas such as the roof of the hull, or allow the weld robot better internal access through an opening such as the windscreen orifice. The robotic cell features appendages such as laser scanner and heat sensors for calibration purpose.

The robot-on-robot set up required a state-of-the-art communication system to ensure each robot was interfaced and could communicate correctly.

The robotic cell was originally programmed via online jog-and-teach method. However the highly complex nature of the cell is a great hindrance to an efficient programming solution. A lot of time was invested in teaching the welding robot all the seam locations, as the extra degrees of freedom added by the auxiliary robot removed a lot of intuitiveness in manually jogging the robot to a specific target location without clashing with the vehicle hull; particularly when negotiating through complex internal geometry.

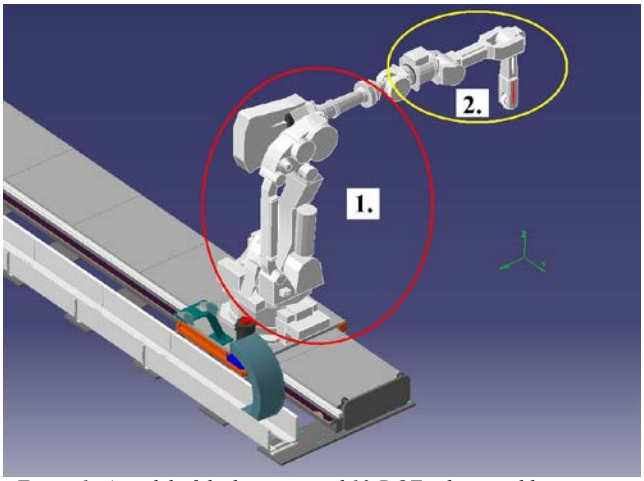


Figure 1: A model of the homogenized 13-DOF robotic welding system, featuring two separate 6-DOF robot and a linear rail.

The manufacturing company is now anticipating orders of other configurations of the vehicle, meaning that the robot cell will need to be reprogrammed to accommodate these various models of the vehicle. After the initial difficulties experienced when utilising the online programming method; it was deemed necessary to explore options in which the robotic cell can be re-programmed for these new designs in a much more efficient manner. Researchers at the University of Wollongong proposed an offline programming approach as an alternative, hoping to create an automated programming system utilising a simulation package widely used in industry today.

At the outset of the project, a literature review of current OLP software was conducted [1]. The review indicated that

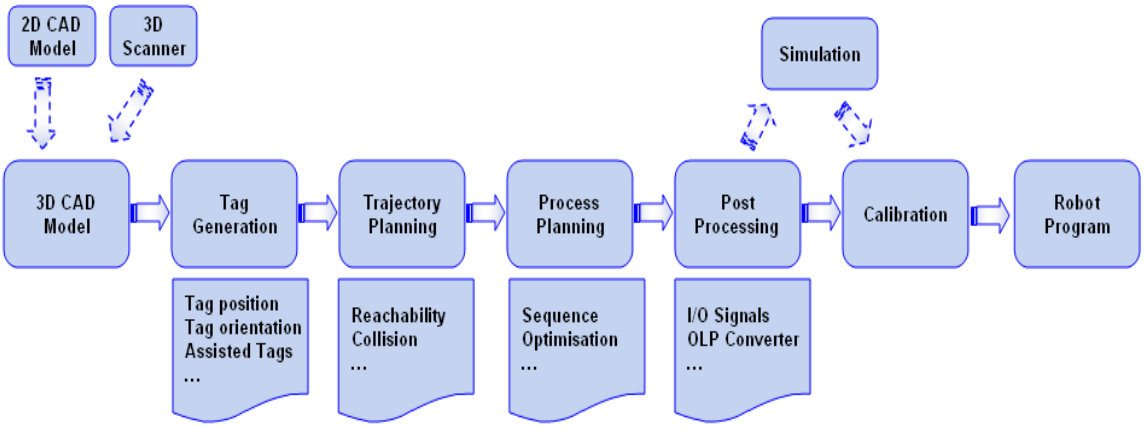


Figure 2: Block diagram of overall offline programming structure [1]

OLP softwares mainly came from 3 sources; from generic robotics software producers, from robot manufacturers [2-5] and finally from research institutions that produce their own programming and simulation software, usually developed around existing CAD software such as AutoCAD or SolidWorks [6-9] or from scratch using OpenGL, VRML and Java technology [10-12].

To create an OLP software package for this complex welding cell, it was decided to utilise a commercially available generic robotic software package. This was chosen as a generic system can be much more flexible in its compatibility with various brands of system hardware and also features virtual reality, allowing the user to be fully immersed into the simulation environment. The DELMIA software package was chosen. Its current and widespread use in robot programming for industry and manufacturing processes, along with its Visual Basic programming interface, were major factors behind its selection. Details of the DELMIA robotics simulation software package are covered below, in section II.

II. DELMIA FOR OFFLINE PROGRAMMING & AUTOMATION

DELMIA is a robotics and manufacturing based OLP package that is utilized widely within various respective manufacturing industries today. DELMIA utilizes a 3D simulation environment to test and optimize robot programs before implementation into real world applications. DELMIA features assorted 'toolboxes' that are available to provide a programmer with numerous functions which are useful to the various specific areas of robotic OLP, such as; robot target definition, reachability analysis, clash testing, path planning/augmentation etc.
A user carrying out the programming of a robotic cell with DELMIA would follow the general steps for OLP highlighted in *Figure 2*.

To aid in the programming of the complex welding cell in question, it is proposed that some of the steps in the above OLP process be automated. These areas related to 3 specific sections of the OLP process: The automatic extraction of seam data from CAD models, the automated generation of reachability and collision assessments and the automatic creation of the robot process with simulation.

The proposed automation of these DELMIA functions was carried out with the Visual Basic interface that comes with DELMIA. DELMIA's robotics related commands which were not accessible through the VBA functionality were controlled via the windows GUI automation program; AutoIT.

III. AUTOMATIC SEAM EXTRACTION

Defining the weld seams to be carried out by the robotic cell is the first step in the OLP process. In DELMIA, these seams are defined by first loading a 3D CAD model of the work object to be welded. The programmer identifies a seam, and then defines it by individually allocating two separate tags at each end of the specific seam. The tag's individual XYZ orientation angles are then augmented by the user to specify the correct approach angles for the manipulator/weld-torch. This process is not a difficult one, however when a high number of seams are to be defined by the user, it has proven to be a monotonous and time consuming task. The VBA/DELMIA interface was identified as a tool which can analyse the drawing features that make up the CAD model of the work object. A programming module was created to aid the programmer in efficiently defining these weld seams. The module assists by providing a 'semi-automated' method of defining the weld seam and then automatically snapping the tags to the seam.

The semi automated approach reduces the time taken to define a seam by first prompting the user to select the edge

they wish to be defined as a seam. The tags are then attached automatically to each end of the selected seam. The user is then prompted to click the two adjacent faces that make up the seam, this is done to define approach/orientation angles for the weld torch as it carries out the weld process, as seen in *Figure 3* below. The semi automated approach developed provides the programmer with a more effective interface to tag their work objects than standard methods available in DELMIA. This is due to the fact that standard methods for tagging in DELMIA require that the user first add two separate tags in the correct location to define the seam, and then they are able to orientate it for the correct approach angles. The semi-automated approach, however, already assumes you are searching for an edge to define as a seam; once the edge is selected by the user the tags are added and orientated automatically. This significantly reduces the amount of individual operations the user has to carry out in order to define a seam, hence cutting down the overall time required. Initial tests on tagging a vehicle hull indicate that the time taken to define each seam in the entire hull was more than halved when using the semi automated approach over standard methods currently available in DELMIA.

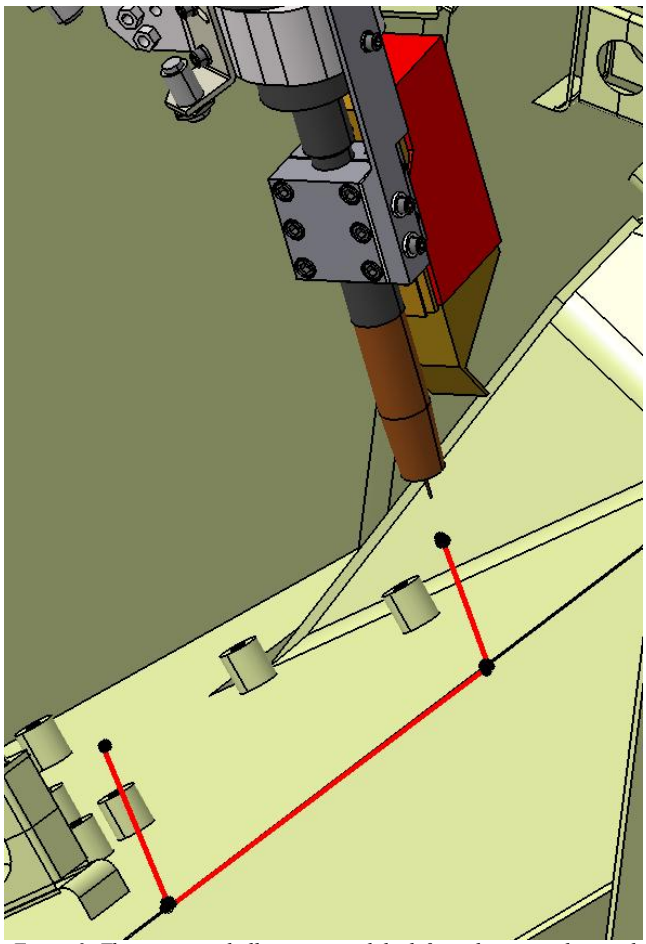


Figure 3: The semi-auto hull tagging module defines the seam edges and approach/orientation angles for the weld torch.

Once the seams have been defined by the user, the data relating to their position and orientation is automatically added to an excel spreadsheet for further use at a later stage of the OLP process.

IV. Path Planning

Once the programmer has defined all of the seams within the vehicle that are to be welded they then move onto the next programming task, which relates to planning the motions required for the robotic cell to correctly carry out the welds. To undertake this task with the complex welding cell being used in this research, the programmer first has to program the auxiliary robot to carry the weld robot to a specific point in space which fulfils two criteria. Firstly, this point in space has to be close enough to the seam to ensure that the weld robot can reach its target. Secondly, specific orientations for the weld robot have to be selected so that it can carry out the welding of the seam without colliding with the work object. As each seam to weld has different locations and orientations in space; the programmer essentially has to find a new point in space that for fills the above criteria for each individual seam that is to be welded. Due to the high number of individual seams that feature on the vehicle in the complex welding cell, the repetitive nature of finding these points becomes monotonous and time consuming. However the repetitiveness of the task also highlighted that it is an ideal candidate for automation, meaning that a lot of time to program the cell will be saved if it were possible to control DELMIA in a way so that it can define automatically for the programmer these points in space, meaning that they don't have to spend the time to find it 'manually'.

The approach for automating this task was to address the two criteria mentioned above as separate modules. The first, which related to automatically finding points in space which fulfilled reachability criteria, was addressed before moving onto the second criteria of defining the correct motions and orientations for clash free motion when welding the seams. To automate the reachability assessments, a 3D array of potential positions for the auxiliary robot is defined by the programmer in an excel spreadsheet; the user can define the position of the array, the volume it occupies and the number of nodes or targets within the array. The inverse kinematics of the weld robot were mapped in Matlab, which utilised the previously defined excel data to calculate the reachability of the weld robot from each test node in the array to the previously defined weld seams within that area. These reachability results are fed back to the spreadsheet so that the programmer can see how many seams can be welded from each potential target in the 3D array of nodes. DELMIA is capable of carrying out reachability tests; however the VBA interface in DELMIA is restricted in its access to this class of commands, making automation of these commands with the VBA interface a non-functional pursuit. Matlab

was used and the M-file created was able to automatically carry out these tests in a very fast and efficient manner. The results displayed to the programmer were offered in an intuitive yet detailed presentation. The programmer is able to see which nodes can provide the weld robot with reachability to a single seam, multiple seams (which are listed) or no seams at all. Data relating to where the node is in 3D Cartesian space is also displayed, along with the orientation (roll/pitch/yaw angles) of the node.

After the reachability testing is concluded, the next 'module' to automate was addressed. This related to automatically testing whether the weld robot can move from the specific pre-defined nodes to the weld seam without clash. Once again DELMIA has capabilities to carry out these commands in a 'manual' manner, however the VBA interface provided does not currently give access to the commands which control these functions, meaning that automating these commands to be carried out without a human operator present would need another approach. To automate these commands, the DELMIA program was interfaced with AutoIT, which is a program designed to control and operate with the windows GUI; meaning that instead of a human operator manually clicking through the DELMIA commands and testing clash, AutoIT was programmed to carry out the same operations. As AutoIT can interface directly with all buttons, text fields etc that DELMIA has to offer; it can effectively carry out these operations much faster than any human. AutoIT was programmed to read the excel spreadsheet utilised in earlier sections of this research. It reads the calculated reachability data and then prompts DELMIA to move the position of the weld robot to each node previously deemed to have a positive reachability result. The robot is then commanded to move to a specified seam that has been deemed reachable at that particular node. As this motion is carried out, DELMIA monitors for clash. AutoIT cycles through each node and returns the clash results back to the excel spreadsheet. Each available robot configuration is also tested for clash, and the data relating to which specific configurations provide a clash free motion is also displayed in the excel spreadsheet of results.

The result of these automated tests is essentially an array of targets for the auxiliary robot to carry the weld robot to. The robot programmer can be safe in the knowledge that these targets will firstly have at least one seam that is within reach of the weld robot from the particular array node. These nodes will also provide a clash free motion for the weld robot as it carries out its weld process on the seams. For optimisation, the programmer can easily check the created excel spreadsheet to select which nodes will be utilised in the final robot program. This is done easily and intuitively as they are provided with the data relating to the seams that are reachable from specific nodes; so they can easily select the fewest individual nodes required to carry out all seams within one particular area of the hull. This will effectively cut down the number of times the weld robot will have to be repositioned within the hull as it carries out its weld processes.

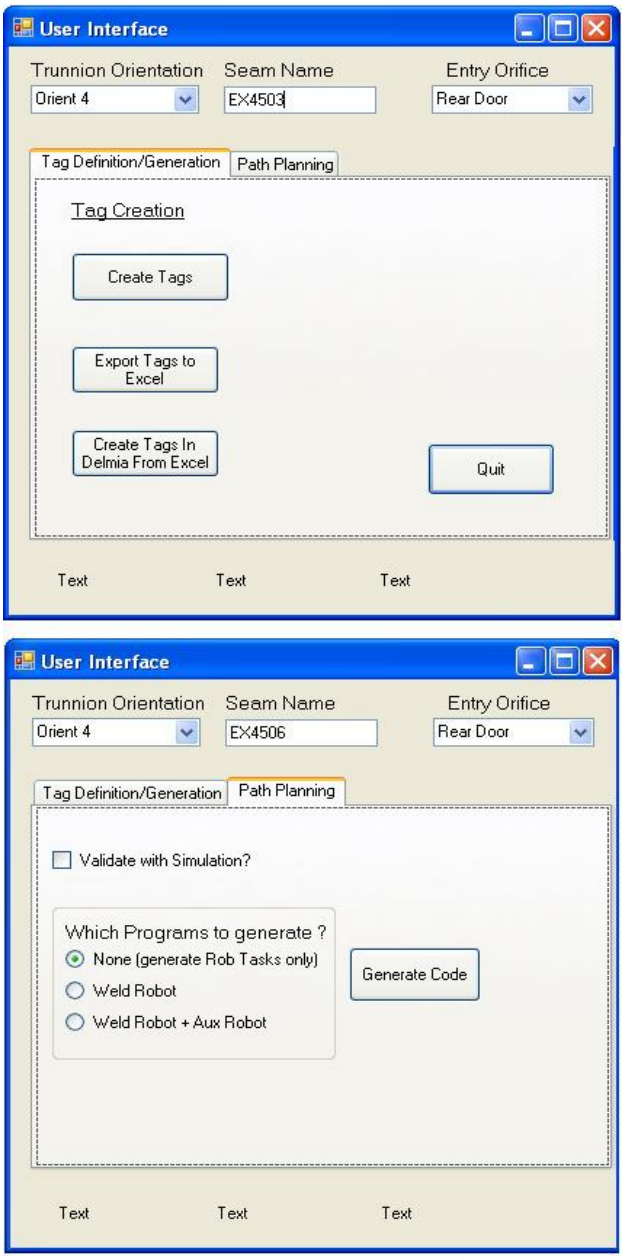


Figure 4: The Visual Basic user interface

Once the correct positioning for the weld robot and suitable configurations for clash free motion have been defined the robot tasks for each robot is created using this data. At the users request, the complete process is simulated from the beginning to verify that the procedure is carried out correctly and without clash. Once the process is verified, the Visual Basic interface exports the native robot programs to a folder on the computer desktop. If, for some reason, during the verification process an error such as clash or unreachability occurs then an error file is also exported with the program. This text file contains the nature of the error and the simulation time at which it

occurred, making it easy for the programmer to trace through the simulation and fix any problems before exporting the program again.

V. Experimental Results

The effectiveness of the created automation system was tested on a number of seams on the vehicle hull. Visual Basic was used to create a user interface, shown in *Figure 4*, in which the programmer has access to the various buttons and controls that manage the operation of the OLP automation system created. The first automation control the user utilizes in the OLP process is the automated generation of the seam targets for the welding robot. The proposed seams to be created were located in an internal section at the rear of the hull in order to fully test the capabilities of the automation system.

The user first enters a desired seam name into an input box on the Visual Basic user control panel, and then clicks the 'Create Tags' button. A blank model of the vehicle hull in a new DELMIA window with all of its drawing features exposed to the user is then opened. A pop up window prompts the user to select the edge between two plates that will form the first seam that is to be defined. The user is then prompted to click on the incident faces to this selected edge. Once the faces have been selected the blank model of the vehicle hull is closed and two tags are placed at each end of the previously selected edge on the hull in the complex welding cell. The previously selected faces automatically orientate the seam tags with the appropriate approach angles for the weld torch, which includes any push/pull angle if required in a corner. These tags are then automatically renamed to the previously defined seam name and are saved into a specific 'semi-auto' tag group in DELMIA. The seam is then fully defined and the user can now move on to repeat the process as many times as they want. Three seams were defined inside the rear section of the hull using the above method. The 'export seam data to excel' button on the user control panel was used to export all data relating to the seam tags locations/orientations to a specific Excel spreadsheet. A list of seams in the excel sheet was created, and the new tags imported are added to the end of this list. This data is used during the later stages of the OLP automation process. Another button on this tab has the capability of importing this list of tags back into the DELMIA environment if required, allowing the user to make modifications to the tag's location/orientation in the excel environment and then import these modified tags back into the simulation model.

Once the Seams have been defined the user then clicks on the 'Path Planning' tab on the user interface to expose the next set of automation controls. Before beginning the path planning automation functions, a matrix of targets for the Auxiliary Robot to place the Weld Robot has to be defined about the rear of the Hull. This is done in the same spreadsheet as the stored seam tag data. The user defines this matrix by specifying in a specific section of the excel sheet the upper left and lower right hand corners of the desired positioning matrix and also entering the desired number of nodes in the matrix. These coordinates are found using the DELMIA simulation model and the DELMIA compass tool. Once a satisfactory positioning matrix has been defined by the user it need not be changed as all future work at the rear of the hull will reference this defined matrix.

The user selects which robot programs they would like to generate as a result of these automated tests, in this instance the button to generate both the weld robot and auxiliary robot programs was selected. The programmer can also check the 'Validate with Simulation' box, which runs a final validation check on reach and clash at the end of the OLP process, *Figure 5*. All that remains for the user to do is enter the name of the weld seam they wish to create the programs for and utilise the 'Generate Code' button on the user interface to initialise the Matlab, Visual Basic and AutoIT automation components listed in section IV.

After validating the process with simulation from start to finish the final output is two separate robot programs. The first program commands the auxiliary robot to move the weld robot to a suitable position close to the weld seams. The second robot program commands the weld robot to move from this base position to carry out the seam weld without clashing with the vehicle hull. If desired the user can at this stage make 'manual' adjustments to the robot task within the simulation environment by using traditional DELMIA commands and then export the modified robot programs again.

Figure 5: Output generates a robot task that can be simulated by DELMIA to validate clash

Once the first seam has been completely programmed, the operator then moves on to obtaining the robot programs for the remaining two seams at the rear section of the hull. This

is done by inputting the next remaining seam name into the user control panel and then clicking the 'Generate Program' button again. Once the program has finished obtaining the code for these seams, the final seam at the rear of the hull was addressed in the same fashion. The time to obtain the programs to correctly weld these three seams took the automation module approximately 5 minutes from start to finish. This result provides a significant improvement over using traditional 'online' jogging methods currently employed by the manufacturing facility.

VI. Conclusion and Future Work

This research has shown that it is possible to modify currently available simulation software to automate some of the steps in the OLP process. The developed modules provide automation functionality to both the tag generation and trajectory planning stages of the OLP process, giving an overall improvement to the time taken for a programmer to produce code for a robotised welding cell.

Whilst the created system is able to provide a level of success in delivering a working package, there are a number of noted issues which have a negative impact on its operation. The main issue negatively affecting the process relates to the level of functionality that is exposed in the DELMIA/VBA interface. DELMIA, in its current development state, provides access to the majority of its functions via the VBA programming interface. However, access to DELMIA's robotics related functions was minimal. This resulted in having to use indirect methods, such as accessing the commands through DELMIAs GUI rather than getting direct access to use certain functions or commands. Examples of this inaccessibility include restricted access to the reachability and clash checking commands. Other problems which featured in this developed OLP package relate to general bugs between DELMIA and the VBA interface. These bugs usually relate to a rapid succession of commands being misinterpreted by DELMIA, causing errors or the program to shut down.

Whilst the overall speed is a significant improvement over 'manual' methods of offline programming, there exists room for improvement in the developed OLP automation package. To overcome this, work has begun on creating new OLP modules in Matlab to replace some of the tasks in the OLP process, removing the reliance on the slower VBA/DELMIA interface in much the same way Matlab was implemented in section IV to carry out the reachability assessments of the weld robot. The main goal behind this is to improve the efficiency of these tasks, whilst also improving the reliability of the program by moving the role of DELMIA/VBA towards being just a tool for simulation.

VII. Acknowledgement

This work is funded by the Australian Defence Materials Technology Centre (DMTC) under project 2.4 and 3.5.

References

[1] Pan Z; Polden J; Larkin N;Van Duin; Norrish J; *"Recent Progress on Programming Methods for Industrial Robots"*, ISR/ROBOTIK, Munich, Germany, June 7-9, 2010.

[2] R.G. Brown; *Driving digital manufacturing to reality;* Proceedings of 2000 Winter Simulatin Conference, Vol. 1, pp. 224-228,10-13 Dec, 2000.

[3] Bruccoleri, M.; D'Onofrio, C.; La Commare, U.; *Off-line Programming and simulation for automatic robot control software generation*; 5th International Conference on Industrial Informatics, Volume 1, Page(s):491 – 496, J 23-27 June 2007.

[4] W. Dong; H. Li; X. Teng; *Off-line programming of Spot-weld Robot for Car-body in White Based on Robcad;* International Conference on Mechatronics and Automation, ICMA 2007, Page(s):763 – 768, 5-8 Aug. 2007.

[5] D.M.A. Lee and W.H. Elmaraghy; R*OBOSIM: a CAD-based off-line programming and analysis system for robotic manipulators;* Computer-Aided Engineering Journal, October, 1990.

[6] J.N. Pries, T. Godinho and P. Ferreira; *CAD interface for automatic robotic welding programming;* Industrial Robot: An International Journal, Vol 31. Number 1, pp. 71-76, 2004.

[7] S. Mitsi, et al; *Off-line programming of an industrial robot for manufacturing*; International Journal of Advanced Manufacturing Technology, vol. 26, 262-267, 2005

[8] Soron, M.; Kalaykov, I.; Generation of continuous tool paths based on CAD models for Friction Stir Welding in 3D; Mediterranean Conference on Control & Automation, MED '07. Page(s):1 – 5, 27-29 June 2007.

[9] Yang Y.; Chen X.; Ling C.; Kang B.; *A Robot Simulation System Basing on AutoLisp*; 2nd International Conference on Industrial Electronics and Applications, ICIEA 2007, Page(s):2154 – 2156,23-25 May 2007.

[10] W. Dai; Kampker, M.; *PIN-a PC-based robot simulation and offline programming system using macro programming techniques;* The 25th Annual Conference of the Industrial Electronics Society, 1999. Volume 1, Page(s):442 - 446 vol.1,29 Nov.-3 Dec. 1999.

[11] Jaramillo-Botero, A.; Matta-Gomez, A.; Correa-Caicedo, J.F.; Perea-Castro, W.; *ROBOMOSP*; IEEE Robotics & Automation Magazine, Volume 13, Issue 4, Page(s):62 – 73, Dec. 2006.

[12] Chang-Sei Kim; Keum-Shik Hong; Hans Yong-Sub Han; Soo-Ho Kim; Soon-Chang Kwon; *PC-based off-line programming using VRML for welding robots in shipbuilding;* IEEE Conference on Robotics, Automation and Mechatronics, Page(s):949 - 954 vol.2, 1-3 Dec. 2004.

Recent progress on programming methods for industrial robots

Abstract
"Although an automated flexible production cell is an intriguing prospect for small to median enterprises (SMEs) in current global market conditions, the complexity of programming remains one of the major hurdles preventing automation using industrial robots for SMEs. This paper provides a comprehensive review of the recent research progresses on the programming methods for industrial robots, including online programming, offline programming (OLP), and programming using Augmented Reality (AR). With the development of more powerful 3D CAD/PLM software, computer vision, sensor technology, etc. new programming methods suitable for SMEs are expected to grow in years to come. (C) 2011 Elsevier Ltd. All rights reserved."

Keywords
industrial, robots, methods, recent, programming, progress

Disciplines
Engineering | Science and Technology Studies

Publication Details
Pan, Z., Polden, J., Larkin, N., van Duin, S. & Norrish, J. (2012). Recent progress on programming methods for industrial robots. Robotics and Computer Integrated Manufacturing, 28 (2), 87-94.

This journal article is available at Research Online: http://ro.uow.edu.au/eispapers/29

Recent Progress on Programming Methods for Industrial Robots

Zengxi Pan, Joseph Polden, Nathan Larkin, Stephen Van Duin, John Norrish
Faculty of Engineering, University of Wollongong, Wollongong, NSW, 2522, Australia

Abstract

Although an automated flexible production cell is an intriguing prospect for small to median enterprises (SMEs) in current global market conditions, the complexity of programming remains one of the major hurdles preventing automation using industrial robots for SMEs. This paper provides a comprehensive review of the recent research progresses on the programming methods for industrial robots, including online programming, offline programming (OLP), and programming using Augmented Reality (AR). With the development of more powerful 3D CAD/PLM software, computer vision, sensor technology, etc, new programming methods suitable for SMEs are expected to grow in years to come.

Keywords: Industrial robot, SMEs, Offline programming, Online programming, Augmented reality

1 Introduction

In the era of globalisation, manufacturing industries are facing increasing dynamics of innovations, shortened product life cycles, and a continuing diversification of the product range. At the same time, they are under the pressure of the shortage and high cost of skilled workers. Industrial robots based automation represents the best solution for both productivity and flexibility. Nevertheless, the programming of industrial robotic system for a specific application is still very difficult, time-consuming, and expensive.

For example, manually programming a robotic arc welding system for the manufacture of a large vehicle hull takes more than eight months, while the cycle time of the welding process itself is only sixteen hours. In this case, the programming time is approximately 360 times the execution time. As a result, small to median sized enterprises (SMEs) are not able to be benefit from robotic automation due to this programming time overhead.

In practical industrial applications, today there are two main categories of robotic programming methods, which are, online programming (including lead-through and walk-through) and offline programming (OLP). [1] Conventionally for online programming, the teach pendant is used to manually move the end-effector to the desired position and orientation at each stage of the robot task. Relevant robot configurations are recorded by the robot controller. And a robot program is then written to command the robot to move through the recorded end-effector postures.

Although the concept is simple, it is only suitable for programming the application of an uncomplicated process onto a workpiece with a simple geometry. In addition, the quality of the program is limited by the skills of the operator and once the program is generated, it is very difficult to make further amendments. In spite of these drawbacks, it is widely used as its intuitiveness, low programming skill requirement, and low initial cost. A few new programming methods are proposed in this category to alleviate the burden of jogging assisted by implementing additional sensors and control technologies.

Nowadays, OLP method, which is based on the 3D model of the complete robot work cell, is becoming more popular. Without removing the tedious programming overhead, OLP shifts the burden of programming from the robot operator in the workshop to the software engineer in the office. OLP has its strength on programming complex systems and is proved to be more efficient and cost-effective for production with large volumes.

Compared to the online programming method, it is more reliable and provides certain flexibility to the changes of product design. Since it relies heavily on the modelling of the robot and the workpiece, additional calibration procedures are usually inevitable to meet process accuracy requirements. Although there are many different OLP software packages available on the market, employing an OLP system usually means great programming effort, large capital investment and long delivery time.

A number of researchers intend to combine the knowledge of real world and the CAD model together to enjoy the benefits of both methods. Robot Programming using augmented reality (RPAR) is a typical example recently developed aiming to improve the intuitiveness and flexibility of OLP task. With different levels of involvement of human-robot interaction, sensor technology and CAD, the boundary between online and offline programming becomes blurred. Recent examples of RPAR will be presented in a separate section although they have not been seen in practical industry yet.

This paper will provide a comprehensive review of research progresses on the robotic programming methods in the last 10 years. It can be seen that the majority of research efforts are focused on providing a suitable robotic programming method for SMEs, by improving online programming methods, OLP methods or combining these two methods together

using new concepts such as AR. Welding (mainly arc welding) and machining (mainly deburring), are the most widely investigated processes due to the fact that welding is the most common task for an industrial robot and machining is considered as a potential challenging application for an industrial robot.

This paper is organised in six sections. Following this introduction, section two presents the recent development on online programming aided by various sensors and control technologies. Section three introduces the structure of OLP methods and provides a survey of available OLP software. Section four describes the features of RPAR with some examples. Summary and research trend are presented in Sections five followed by an acknowledgement in Section six.

2 Online programming

Online programming has conventionally been carried out by skilled robot operators by guiding the robot through the desired path using a teach pendant, namely the lead-through method. Typically, the lead-through method includes the steps of jogging the robot through the desired path, recording the specific points in robot controller, and utilizing the recorded points to create movement commands. The robot operator programming a robot using a lead-through method is responsible for guiding the robot and maintaining the desired position and orientation of the robot in six degree-of-freedoms (DOFs).

Although the conventional online programming method is simple and has been widely used, it has several drawbacks. Firstly, jogging a robot using a teach pendant is not intuitive as many coordinate systems are usually defined in a robotic system. The operator must always track which coordinate frame the robot is set in when jogging. Guiding the robot through the desired motion accurately while never allowing a collision with an object in the workspace is usually a very difficult and time-consuming task, especially when the workpiece has a complex geometry or the process itself is very complicated.

In addition, when a program is generated, a lot of testing work has to be done before the program is satisfactory for reliability and safety reasons. Thirdly, the robot program generated using the lead-through method lacks the flexibility and reusability. The tedious programming process has to be repeated again for a workpiece with only a slight difference. Other drawbacks of lead-through method include; the robot cannot be used for production during the teaching period, the operator is exposed to a hostile environment, and the quality of motions taught rely on the skill level of the operator.

In spite of all the above mentioned drawbacks, online programming is still the only programming choice for most SMEs.

Online programming methods using more intuitive human-machine interfaces (HMI) and sensory information have been proposed from several institutions. Table 1 lists the recent research efforts on assisted online programming. The assisted online programming can be categorised into operator assisted online programming and sensor guided online programming.

2.1 Operator assisted online programming

To make jogging a robot in 3D space more intuitive, a few assistant teaching devices have been developed for walk-through teaching. Sugita [2] presented a teaching method using teaching support devices developed for a deburring and finishing robot. Two teaching support devices were introduced to measure the position and direction vector of the dummy tool on the tip of the posture measuring unit, and used to generate robot program in robot coordinate.

Choi [3] presented the development of a force/moment direction sensor named COSMO that can improve the teach pendant based robot teaching. An experiment teaching a six axis commercial robot using the sensor is described where operator holds the sensor with a hand, and moves the robot by pushing, pulling, and twisting the sensor in the direction of the desired motion. No prior knowledge of the coordinate system is required. The sensor used in the device is a micro-switch, and this intuitive robot teaching can be implemented at a very low cost.

Schraft [4] proposed an intuitive teaching method to use a walk-through attempt to provide a tool for fast and effective teaching of industrial robots in this niche. The user guides the robot with a handle that is equipped with a force torque sensor and commands the robot using a speech dialog system. The acquired trajectory can be adapted by using a PDA and 3-D graphical user interfaces.

2.2 Sensor guided online programming

Using assistant teaching devices usually means introducing additional sensors and calibration procedures to an already very complex robotic system. Pan [5] developed a programming by guiding (PbG) method based on ABB's IRC5 controller with the optional force control feature. Two major functions provided by the commercial available force controller make the entire programming process collision free and automatic. The first function is walk-through, in which robot

is compliant in selected directions (force control directions) and stiff in other directions (position control directions). To change the position or orientation of the robot, the robot operator could simply push or drag the robot with one hand. The second function is called path-learning, in which robot is compliant in the normal-to-path-direction to make the tool constantly contact the work piece.

As the accuracy of the final program is determined by the robot force controller and does not rely on the skill of the robot operator, a 3D robot path with higher accuracy can be generated automatically. This is of extreme benefit for applications when process tools have contact with workpiece, such as in machining processes.

While Pan and Zhang's method still requires jogging during the first stage of programming to guide the robot motion, some other researchers have eliminated jogging from the entire programming process by involving other sensor technologies. Zhang [6] used the same controller platform and extended the concept by adding a visual servo. The system configuration of this system is shown in Figure 1. A hybrid position/force/vision control platform was developed to control the robot motion in different directions using various sensor feedbacks. The system is able to generate a robot program by automatically following a path marked with a standard marker pen. The position control is used to maintain the tool orientation; vision sensing is used to follow the curve; and force sensing is used to maintain the contact between the tool and the workpiece.

Solvang [8] also presented a vision based programming methodology by identifying a path drawn onto the workpiece. This line is captured by a single camera for the 2D (x and y) coordinates. The depth coordinate (z) is achieved by a virtual "hit and withdrawal" procedure using a commercial available simulation program that uses the industrial robot to map the surface of the workpiece. During the mapping process, the robot moves along the existing 2D path and at every point of the path contacts the work-piece surface. When contact occurs, the z coordinate is stored establishing the position. Although CAD model of the workpiece is used, the major part of this method is still sensor-robot interaction rather than offline path planning.

Nicholson [9] developed a rapid robot programming method using image data for weld reclamation repair works. Instead of drawing marks on the workpiece, the user interacts with the image to define select/define the robot 2D path. This selection is done via a drawing module which allows the user to generate an area onto the picture of the workpiece. The z coordinate is determined using the touch sensing built into the welding system. Unlike most vision based systems, which are reliable on calibration results and sensitive to lightening condition, this method provide robust results due to its simplicity.

In some situations, projecting structured light using a laser is more feasible than drawing marks on the workpiece. Gonzalex [10] presented aspects related to the generation and tracking of closed trajectories over a surface of unknown geometry using structured lighting in the form of a laser spot matrix. Simple image analysis algorithms can be used to detect the centre of laser spots in the images. After the process of surface characterization is complete, the user selects, in camera-space, a starting point and a direction of reference over the surface for the robot path. As the image plane information gathered from the projection of structured lighting is limited, a second order polynomial function is defined to approximate the 3D curve welding path considering the best fit to the surface. A closed 3D path is achieved by connecting the starting point and ending point of the neighbouring segment of the trajectory.

Hu [11] developed a strategy to automate a leather surface roughing process using structured light 3D machine vision for object profile perception. The structured light scanning system consists of an analogue camera, laser line generator and driven linear slide to provide scanning motion for the camera and laser. Non-Uniform Rational B-Spline (NURBS) interpolation is applied to reconstruct a smooth continuous trajectory from the discrete path coordinates.

Stereo vision was also used to acquire 3D coordinates for robot programming and distinct features such as corners and edges could be easily identified from a workpiece. Takarics [12] attempted to use the stereo vision technology to program a weld trajectory based on the intelligent space concept using two fixed cameras. The weld seam is recognized in two images by edge detection algorithms and the path trajectory was generated by the 3D reconstruction from both images. The method is capable of generate a 2D planar curved path for arc welding processes.

Although dramatic progress has been carried out to make online programming more intuitive, less reliant on operator skill, and more automatic, most of the research outcomes are not commercial available aside from [6]. This is partially because most of these methods are limited to their specific setups and are yet to be applied to general applications. As cost-effective sensor assisted online programming solutions become commercially available, the installations of robotic automation cells will become more cost effective for SMEs.

3 Development of OLP

OLP methods, which utilise 3D CAD data of a workpiece to generate and simulate robot programs, are widely used for automation system with large product volumes. Herein the complete robot cell is modelled in 3D. The user can test the reachability, fine-tune properties of robot movements and handle process related information before generating a program that can be downloaded to the robot.

OLP offers many advantages over the online method. Firstly, the programming process does not require the actual robot, minimising the production robot down time. Robot programs can be developed earlier in the design/production cycle and programming can be carried out in parallel with production rather than in series with it. Secondly, programs generated offline are more flexible than jog-and-teach method. Program changes can be incorporated quickly by only substituting the necessary part of the program and previously developed routines can be easily included in new programs. Thirdly, simulation is usually incorporated into the OLP method. As a result, programs can be pre-checked, thereby confirming the robots' movements, minimising the chance of error and therefore improving productivity and safety. There is also a greater possibility for optimization of the workspace layout and the planning of robot tasks.

Although OLP has the above mentioned advantages, it is not popular for SMEs users due to its obvious drawbacks. It is difficult to economically justify an OLP for smaller product volumes due to the high cost of the OLP package and programming overhead required to customise the software for a specific application. Development of customised software for off-line programming is time-consuming and requires high level programming skills. Typically, these skills are not available from the process engineers and operators who often perform the robot programming in process today. As OLP methods rely accurate modelling of the robot and work cell, additional calibration procedures using extra sensors are in many cases inevitable to meet process requirements.

While OLP software providers emphasise on making the OLP package more powerful, modular, and flexible to reduce secondary development for specific applications, academic researchers have dedicated attention to improved process planning algorithms and have developed a few OLP software package using open source technology.

3.1 Steps of OLP

OLP is more complex than online programming as the programming method not only needs to acquire the 3D robot targets but also needs to plan the trajectory of robot motion and optimise the sequence of the process. The key steps of OLP are shown in Figure 2.

3.1.1 Generation of 3D CAD model

OLP starts from a 3D CAD model of the workpiece, while it is very common for a product to have a CAD model, for parts without a 3D model, or a product that has changed after its CAD model is finalized there are several methods available to generate the required 3D computer model.

In some situations, a 3D scanner can be used to capture the workpiece geometry [13]. The collected points-cloud is converted to the surface model of the workpiece and a smoothing/filtering procedure removes sensory noise before the model can be used for tag creation.

In other situations, when only a 2D CAD is available, the 3D model of the workpiece can be obtained from either multiple views of a 2D drawing [14], by additional sensors, or the robot is simply programmed in 2D [15].

Although there are various types of CAD files, most modern OLP software packages are capable of converting other types of CAD files to a compatible. Conversion between different types of CAD files is less a problem these days with the developments in the CAD/CAM industry.

3.1.2 Tag creation

This step involves extracting robot position tags from 3D CAD data with a specific tool centre point (TCP). It is usually a time consuming process and can require secondary programming for automatic tag recognition. OLP software is available that provides built-in functions to generate tags from features, such as corners and edges, from CAD data. The position and orientation information of the tool must be generated from a combination of CAD model and process requirements. Assistant tags such as home points, approach points, and retreat points are also specified manually in a CAD environment. Attempts have been made to automatically extract robot motion information from the CAD data such as the system proposed by [16].

3.1.3 Trajectory planning

Since the inverse kinematics of industrial articulated robots usually have multiple solutions in Cartesian space, the robot configuration needs to be selected by considering issues such as reachability, minimising configuration transition, collision avoidance, etc. As most of the existing OLP software is not able to provide an optimal solution automatically, either manual assignment or secondary software development using APIs is necessary at this step.

3.1.4 Process planning

Planning a complex manufacturing process involves a higher level of optimisation for resource assignment, cooperation of multiple robots to minimise cycle time. As this step is more relevant to the requirement of a specific process, it is not available in commercial OLP software. For robotic weld large structures, the task sequencing of a large number of welds within limited cycle time can be treated as a general "travelling salesman problem" (TSP), solutions based on genetic algorithm have been proposed by a few researchers [17-19].

3.1.5 Post processing

The post processing stage includes adding necessary I/O control signals for equipment in the work cell, smoothing and fine tuning the path if necessary, and conversion to the program language of the specific robot target. Post processing is more of an issue for generic OLP software as it requires compatibility among different robot manufacturers [22].

3.1.6 Simulation

Robotic work cell simulation is considered as a significant tool that OLP software packages bring to the robotic programming. Simulation enables the program to be verified without the use of an existing physical robot, which reduces the downtime of a robotic system [13, 24].

3.1.7 Calibration

Ideally, a program generated in an OLP system would be downloaded to the robot controller and put into action immediately [20]. In practise, however, the deviation between the actual geometry of elements in the work cell, such as the workpiece, and the nominal geometry makes calibration almost necessary for all OLP system.

3.2 Existing robotics OLP software

Robotic manipulators are highly complex systems. Consequently, the development of computational platforms that allow for their precise modelling, and close to real-life simulation of their behaviour, constitute a fundamental tool for robot designers, users, and students of the field. This reason has inspired the creation of numerous graphical software environments, from non robot manufacturers, academic researcher and also from the robot manufacturers themselves.

3.2.1 OLP software from robot manufacturers

It can be seen from table 2, that almost every robot manufacturer has its own OLP software. Since the OLP software is more compatible to the robot hardware, secondary development of the OLP system is relatively easier. The cost of this type of OLP package is generally lower than one using genetic OLP software as the hardware and software are packaged together. This explains why ABB RobotStudio is by far the most widely used OLP software.

3.2.2 Generic OLP software

This category includes two most powerful OLP software, Delmia (formally IGRIP, ENVISION with third party add-ons from Kineo, CENIT) from Dassault Systems and RobCAD (Em-Workplace) from Technomatix. The advantage of generic packages is that they are more flexible for hardware from different manufacturers and often link to product lifecycle management (PLM) packages to provide production line optimisation. Major automobile and airplane manufacturers use these packages to integrate the robotic systems into their general automated production line. Also, both software packages have the feature of Virtual Reality which allows the user to be fully immerged into the simulation environment.

Today, OLP systems are able to do more than just simulate robot trajectories and perform assembly simulation. Simulation technologies are also able to model the interaction of several manufacturing processes, manufacturing resources, and product maintenance issues.

3.2.3 Open source or academic OLP software

Due to the high cost and limited accessibility of commercial OLP software, a number of researchers have developed alternative OLP software. While some researchers [27-29] [16] have developed OLP packages based on the existing CAD software, such as AutoCAD and Solidworks, others [20] [30-31] have started from scratch using OpenGL, VRML and Java technology.

3.3 Gaps of OLP software and requirements

Due to the costs and the complexity, the advantages of OLP are not sufficient for the use of this technology in the manufacturing operations of SMEs.

There is no available OLP system on the market, which has implemented the complete OLP chain, although many links exist separately. For example, DELMIA V5 Robotics provides functions for tag creation and trajectory planning. However, they still need be created manually in OLP environment or coded using automation (VB/VBA) technology. For arc welding of a complex structure, steps 2, 3, and 4 of OLP are extremely tedious. It may take a few weeks to generate tags for the hundreds of seams inside a vehicle hull, including both accurate position and proper orientation. Although some components for path planning and process optimisation, such as collision detection, layout planning, time measurement, etc are available, a complete path planning function does not exist and is fully relied on the process knowledge of the programmer.

4 Programming using Augmented Reality

Burdea [32] provided a review of the synergy between virtual reality and robotics. AR is an emerging technology that has been derived from VR [33]. AR is an environment where computer-generated 3D objects are blended onto a real world scene, to enhance a user's interaction with the real world [34]. The use of Augmented Reality for robot programming represents a revolutionary concept.

Augmented reality, i.e. interactively overlaying the real environment with virtual spatial information, can be used to make the advantages of graphically-interactive simulation directly available in the real production environment and to provide an efficient and intuitive communication channel for spatial information [35]. As shown below in Figure 4 a virtual model of an aeroplane washing robot can be superimposed over a scaled model of an aeroplane. The virtual model of the robot can be moved about the model airplane to generate a robot sequence that can later be calibrated and programmed for an actual airplane washing robot.

These Robot Programming using AR (RPAR) techniques allow a form of offline robot programming to take place without having to model the workpiece in the virtual environment. RPAR is also useful when an in-situ approach is required as the virtual robot can be augmented into the real-world workcell. This approach can eliminate a lot of technical difficulties that can relate to calibration issues between the virtual and real worlds.

RPAR carries through some of the inherent benefits of OLP, such as not having to take a physical robot out of production, and the safety and operational benefits are retained as well. Another advantage of the proposed RPAR environment with a virtual robot is the programming of large robots where the online method is unfeasible (such as airplane washing robots) as the proposed methodology for planning collision-free path and the RPAP approach are scalable. [33]

A RPAR system was created by utilizing the video-based tracking method in the ARToolkit and creating the necessary coding using the C programming language. This ARToolkit method utilises identification tags with unique patterns printed on them placed around critical elements of the workcell. The Head Mounted Display (HMD) of the RPAR system consists of a single IEEE Firefly camera and an i-glasses video display goggle. This setup utilises a tracking approach where markers attached to objects (static or moving) are tracked using a cameras attached to the HMD. The cameras not only provide the video images needed for processing, but also provide the user with a view of the real world [33].

The technologies AR offer a highly potential utility concerning the improved application of simulation technique during the planning and development of production systems. Figure 5 shows in which ways the respective technologies can be applied during each stage of the simulation process [37].

5 Summary

Conventional online programming is a completely manual process. The robot operator has the freedom to move the robot, select the configuration and plan the process. It is an efficient and cost effective solution for a simple robotic system. As the process becomes more complex, the suitability of online programming reduces.

On the other hand, OLP is a complete automatic programming process. Once the complete work cell is modelled in CAD and OLP code is developed for a specific application, the robot program is generated automatically. As modelling and OLP coding creates a large cost overhead, it is only economically justified for production with large volumes, usually by large enterprises. Table 3 compares the pros and cons of various online and offline robot programming methods. Programming using VR/AR are not included as they have not been practically used by industry yet.

In the last 10 years, extensive research efforts have been carried out on the methodologies for programming industrial robots suitable for SMEs. The boundary between online and offline programming methods are becoming blurred as many new proposed methods includes components from both sides.

Progress in online programming is largely based around sensor and control technologies to assist the operator in creating complex robot motion more easily. Development in OLP bifurcates into different directions. While the commercial OLP providers are developing more powerful, modular and compatible OLP packages, academic researchers have not give up on low cost open source OLP solutions. RPAR is originated from the idea of making OLP more interactive and flexible. In fact, it combines the features of both online and offline programming. With the development of more powerful 3D CAD/PLM software, computer vision, sensor technology, etc, new programming methods suitable for SMEs are expected to grow in years to come.

6 Acknowledgement

This work is funded by the Australian Defence Materials Technology Centre (DMTC) under project 2.4 and 3.5.

7 Literature

[1] M.H. Jr, L. Wei, L.S. Yong; *An industrial application of control of dynamic behaviour of robots- a walk-through programmed welding robot*; Proceeding of the 2000 IEEE International Conference on Robotics and Automation, San Francisco, CA, April 2000.

[2] S.SUGITA, ET, AL, *Development of robot teaching support devices to automate deburring and finishing works in casting,* The International Journal of Advanced Manufacturing Technology, Springer-Verlag London, Dec 2003

[3] M.H. Choi, W.W. Lee; *A force/moment sensor for intuitive robot teaching application*; Proceedings of IEEE International Conference on Robotics and Automation, ICRA. Page(s):4011 - 4016 vol.4, 2001.

[4] R.D. Schraft, C. Meyer; The need for an intuitive teaching method for small and medium enterprises; 2006 ISR-Robotik, Munich, Germany, May 15-18 2006

[5] Z. Pan, H. Zhang; R*obotic programming for manufacturing industry;* Proceedings of ICMEM, International Conference on Mechanical Engineering and Mechanics, Wuxi, China, 5-7 Nov. 2007.

[6] H. Zhang, H. Chen *et al*; *On-line path generation for robotic deburring of cast aluminium wheels;* Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Oct. 9-15, Beijing, China, 2006.

[7] Z. Pan, H. Zhang; R*obotic machining from programming to process control: a complete solution with force control;* Industrial Robot: An International Journal; Vol. 35 Issue 5, page 400-409, 2008

[8] B. Solvang, G. Sziebig and P. Korondi; *Robot programming in machining operations*; chapter in book title: Robot Manipulators, 978-953-7619-06-0, , intechweb, 2008

[9] A. Nicholson; *Rapid adaptive programming using image data,* Ph.D. dissertation, University of Wollongong, Australia, 2005.

[10] E.J. Gonzalex-Galvan, *et al; An algorithm for optimal closed-path generation over arbitrary surfaces using uncalibrated vision*; IEEE International Conference on Robotics and Automation, Roma, Italy, 10-14 April, 2007.

[11] Z. Hu, C. Marshall, R. Bicker, and P. Taylor; *Automatic surface roughing with 3D machine vision and cooperative robot control*; Robotics and Autonomous Systems, Volume 55, Issue 7, 31 July 2007, Pages 552-560

[12] Takarics, B., Szemes, P.T., Nemeth, G.;,Korondi, P.; *Welding trajectory reconstruction based on the Intelligent Space concept;* 2008 Conference on Human System Interactions, Page(s):791 – 796, 25-27 May 2008

[13] Bi, Z.M.; Lang, S.Y.T.; *A Framework for CAD- and Sensor-Based Robotic Coating Automation*; IEEE Transactions on Industrial Informatics, Volume 3, Issue 1, Page(s):84 – 91, Feb. 2007.

[14] J.Y. Kim; *CAD-Based automated robot programming in adhesive spray systems for shoe outsoles and uppers*; Journal of Robotic Systems, 625-634, vol 21, 2004

[15] T. Pulkkinen, et al; *2D CAD based robot programming for processing metal profiles in short series manufacturing*; International Conference on Control, Automation and Systems; Seoul, Korea, Oct. 14-17, 2008

[16] J.N. Pries, T. Godinho and P. Ferreira; *CAD interface for automatic robotic welding programming;* Industrial Robot: An International Journal, Vol 31. Number 1, pp. 71-76, 2004.
[17] K. Y. Kim, D.W. Kim, B.O. Nnaji; *Robot arc welding task sequencing using genetic algorithms*; IIE Transactions, 34, pp. 865-880, 2002
[18] H.J. Kang, J.Y. Park; *Work planning using genetic algorithm and 3D simulation at a subassembly line of shipyard*; OCEANS '04. MTTS/IEEE TECHNO-OCEAN '04, Page(s):218 - 222 Vol.1, 9-12 Nov. 2004.
[19] P.Th. Zacharia, N.A. Aspragathos; *Optimal robot task scheduling based on genetic algorithms;* Robotics and Computer-Integrated Manufacturing, 21, pp.67-79, 2005.
[20] W. Dai; Kampker, M.; *PIN-a PC-based robot simulation and offline programming system using macro programming techniques;* The 25th Annual Conference of the Industrial Electronics Society, 1999. Volume 1, Page(s):442 - 446 vol.1,29 Nov.-3 Dec. 1999.
[21] R.G. Brown; *Driving digital manufacturing to reality;* Proceedings of 2000 Winter Simulatin Conference, Vol. 1, pp. 224-228,10-13 Dec, 2000
[22] Bruccoleri, M.; D'Onofrio, C.; La Commare, U.; *Off-line Programming and simulation for automatic robot control software generation*; 5th International Conference on Industrial Informatics, Volume 1, Page(s):491 – 496, J 23-27 June 2007.
[23] W. Dong; H. Li; X. Teng; *Off-line programming of Spot-weld Robot for Car-body in White Based on Robcad;* International Conference on Mechatronics and Automation, ICMA 2007, Page(s):763 – 768, 5-8 Aug. 2007.
[24] D.M.A. Lee and W.H. Elmaraghy; R*OBOSIM: a CAD-based off-line programming and analysis system for robotic manipulators;* Computer-Aided Engineering Journal, October, 1990.
[25] Vollmann, K.; *A new approach to robot simulation tools with parametric components*; IEEE ICIT '02. 2002 IEEE International Conference on Industrial Technology, Volume 2, Page(s):881 - 885 vol.2, 11-14 Dec. 2002.
[26] L. Zlajpha; *Simulation in robotics*; Mathematics and Computers in Simulation, pp. 879-897, vol. 79, 2008.
[27] S. Mitsi, et al; *Off-line programming of an industrial robot for manufacturing*; International Journal of Advanced Manufacturing Technology, vol. 26, 262-267, 2005
[28] Soron, M.; Kalaykov, I.; Generation of continuous tool paths based on CAD models for Friction Stir Welding in 3D; Mediterranean Conference on Control & Automation, MED '07. Page(s):1 – 5, 27-29 June 2007.
[29] Yang Y.; Chen X.; Ling C.; Kang B.; *A Robot Simulation System Basing on AutoLisp*; 2nd International Conference on Industrial Electronics and Applications, ICIEA 2007, Page(s):2154 – 2156,23-25 May 2007.
[30] Jaramillo-Botero, A.; Matta-Gomez, A.; Correa-Caicedo, J.F.; Perea-Castro, W.; *ROBOMOSP*; IEEE Robotics & Automation Magazine, Volume 13, Issue 4, Page(s):62 – 73, Dec. 2006.
[31] Chang-Sei Kim; Keum-Shik Hong; Hans Yong-Sub Han; Soo-Ho Kim; Soon-Chang Kwon; *PC-based off-line programming using VRML for welding robots in shipbuilding;* IEEE Conference on Robotics, Automation and Mechatronics, Page(s):949 - 954 vol.2, 1-3 Dec. 2004.
[32] Burdea, G.C.; Invited review: the synergy between virtual reality and robotics; IEEE Transactions on Robotics and Automation, Volume 15, Issue 3, Page(s):400 - 410 June 1999.
[33] J.W.S. Chong, S.K. Ong, A.Y.C. Nee, K. Youcef-Youmi; *Robot programming using augmented reality: An interactive method for planning collision-free paths*; Robotics and Computer-Integrated Manufacturing, Volume 25, Issue 3, Pages 689-701 June 2009.
[34] T. Pettersen, et al; Augmented reality for programming industrial robots; Proceedings of the 2nd IEEE and ACM International Symposium on Mixed and Augmented Reality; pp. 319-320, 7-10 Oct. 2003.
[35] G. Reinhart, U. Munzert, W. Vogl; *A programming system for robot-based remote-laser-welding with conventional optics;* CIRP Annals - Manufacturing Technology, Volume 57, Issue 1, Pages 37-40, 2008
[36] Bottazzi, V.S.; Fonseca, J.F.C.; *Off-Line Robot Programming Framework;* Joint International Conference on Autonomic and Autonomous Systems and Networking and Services, ICAS-ICNS 2005. Page(s):71 – 71, 23-28 Oct. 2005.
[37] W. Dangelmaier, et al; Virtual and augmented reality support for discrete manufacturing system simulation; Computers in Industry, pp. 371-383, vol. 56, 2005

Advanced Materials Research Vols. 591-593 (2012) pp 1371-1375
Online available since 2012/Nov/29 at www.scientific.net
© (2012) Trans Tech Publications, Switzerland
doi:10.4028/www.scientific.net/AMR.591-593.1371

Bringing Path Planning and Lean Automation Together

Joseph Polden[1, a], Zengxi Pan[1,b], Nathan Larkin [1,c], Stephen van Duin [1,d]

[1]Faculty of Engineering, University of Wollongong

Northfields Avenue, Wollongong NSW 2522, Australia

[a]jwp973@uowmail.edu.au, [b]zengxi, [c]nlarkin, [d]svanduin@uow.edu.au

Keywords: Path Planning, Lean Automation, Robotics, Probabilistic Roadmap Planner

Abstract. In order to achieve higher productivity and flexibility, manufacturing industry is turning increasingly to robotics based lean automation systems. This lean approach presents a series of new challenges for the control, operation and programming of robotic hardware implemented to carry out a range of manufacturing processes. This paper reviews relevant path planning methodologies alongside a specific set of requirements for a manipulator operating in a lean automation workcell. Then, new challenges to path planning for a lean automation system are presented. Finally, a framework for a new path planner is developed and its performance is compared to existing methods.

The full article below removed for copyright reason, please refer to:
Polden, J., Pan, Z., Larkin, N. & Van Duin, S. (2012). Bringing path planning and lean automation together. *Advanced Materials Research*, 591-593, 1371-1375.
doi: 10.4028/www.scientific.net/AMR.591-593.1371

Manufacturing Engineering and Automation II
10.4028/www.scientific.net/AMR.591-593

Bringing Path Planning and Lean Automation Together

10.4028/www.scientific.net/AMR.591-593.1371

INTECH
open science | open minds

OPEN ACCESS
ARTICLE

International Journal of Advanced Robotic Systems

Path Planning with a Lazy Significant Edge Algorithm (LSEA)

Regular Paper

Joseph Polden[1,*], Zengxi Pan[1], Nathan Larkin[1] and Stephen Van Duin[1]

1 Defence Material Technology Centre, Faculty of Engineering, University of Wollongong, Australia
* Corresponding author E-mail: jwp973@uowmail.edu.au

Received 21 Jun 2012; Accepted 18 Sep 2012

DOI: 10.5772/53516

© 2013 Polden et al.; licensee InTech. This is an open access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract Probabilistic methods have been proven to be effective for robotic path planning in a geometrically complex environment. In this paper, we propose a novel approach, which utilizes a specialized roadmap expansion phase, to improve lazy probabilistic path planning. This expansion phase analyses roadmap connectivity information to bias sampling towards objects in the workspace that have not yet been navigated by the robot. A new method to reduce the number of samples required to navigate narrow passages is also proposed and tested. Experimental results show that the new algorithm is more efficient than the traditional path planning methodologies. It was able to generate solutions for a variety of path planning problems faster, using fewer samples to arrive at a valid solution.

Keywords Path Planning, Lazy Evaluation, Probabilistic Roadmap (PRM), Bridge Test

1. Introduction

The general path planning problem is given as; "planning a collision free path for a robot made of an arbitrary number of polyhedral bodies among an arbitrary number of polyhedral obstacles, between two collision free positions of the robot" [1]. Path planning methods developed during earlier years approached this problem by generating explicit representations of the robots surrounding environment, so that it can be navigated via the use of mathematical algorithms. However, as path planning applications became more complex, these planners began to struggle with the amount of data required to process a valid solution.

A major breakthrough in the field of path planning came with the development of probabilistic path planning methods, such as the Probabilistic Roadmap (PRM) planner [2]. These planners utilize a randomized sampling based approach, which builds a simplified model of the robot's free space. This removes the high computational load involved in calculating an explicit representation of the environment. The random nature of sampling the environment ensures that probabilistic planners have a quality known as probabilistic completeness: if a solution is possible the planner will find it, provided that the time frame is not finite. Probabilistic methods, however, suffer from the fact that they cannot explicitly recognize if a solution will be geometrically impossible.

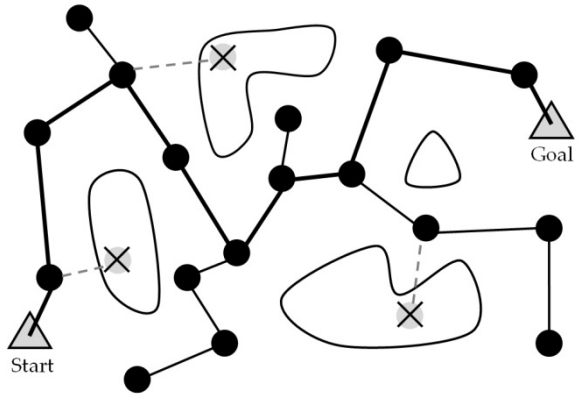


Figure 1. A network, or roadmap, graph ($\dot{G}$) represents the C_{free} space of a robots environment.

Probabilistic planners have been utilized extensively since their development and have been applied to a variety of different applications. The research presented in this paper focuses on the development of new, more effective, sampling methods which are implemented on a traditional lazy probabilistic path planner [3]. This is done through an iterative process of roadmap analysis and enhancement that intelligently guides the construction of the network graph. The overall result is a planner with a similar operation to traditional methods, but that is able to solve path planning queries in complex environments more quickly.

2. Background and Literature Review

Probabilistic methods generally operate on the premise of a robot and its associated configuration space (C_{space}). Loosely termed, C_{space} is the group of all kinematically feasible configurations of a robot about its workspace. For path planning applications, it is common to separate C_{space} into two distinct subsets; C_{free} and C_{forbid}. C_{free} represents the free configuration space of the robot: a group of configurations that do not clash with the surrounding environment. Conversely C_{forbid}, the forbidden configuration space, represents a collection of robot configurations that are. Generally speaking, probabilistic methods try to capture a simplified representation of the robot's C_{free} space. Once the planner has a sufficient understanding of the regions the robot can move about in without clashing, a series of clash free motions to solve a given path planning query can be generated.

An important tool utilized by a variety of probabilistic planners is an undirected network graph, commonly referred to as the Roadmap ($\dot{G}$), as shown in Figure 1. The nodes of $\dot{G}$ are used to represent individual C_{free} robot configurations captured by the sampler and the interlinking edges are used to represent a clash free motion between them. $\dot{G}$ can be used to solve a given path planning query by linking start/goal configurations (q_{start} & q_{goal}, respectively) to $\dot{G}$ and then using traditional graph searching techniques to trace a path from q_{start} to q_{goal}. This path can then be translated into a series of clash free motions used to direct the robot's subsequent movements. By utilizing $\dot{G}$ as a simplified representation of C_{free} we can solve path planning problems without having to generate an explicit representation, which can be costly to compute. Currently, network based planners are used in industry to solve path planning problems for a variety of applications including; industrial manipulators [3], mobile robot navigation [4], assembly tasks [5] and computer game animation. This is a testament to the flexibility, power and ease of use of probabilistic methods.

PRM planners were an early development in probabilistic methods that made effective use of the roadmap graph structure. PRM planners remain popular even today. They are effective for use in complex problems, whilst remaining relatively simple and robust in their operation. PRM planners operate in two distinct phases; the construction of $\dot{G}$, and querying $\dot{G}$ to solve the path planning problem. The first phase consists of building $\dot{G}$ by incrementally sampling random robot configurations for a clash. Samples found to lie in C_{free} are added to the map as network nodes. Subsequent motions between these nodes are also tested for clashing and are added as edges if they lie entirely in C_{free}. This process continues iteratively until a certain density of nodes/connections is achieved. The roadmap is then stored for querying. The query phase involves linking q_{start} & q_{goal} to the stored roadmap and then utilizing graph searching techniques to return a path between them. If no path is available, further sampling can be done. If the environment remains unchanged, the roadmap can be utilized multiple times to solve different path planning queries. Experiments show that the PRM planner is very effective. With the basis of its operation reinforced by probabilistic completeness, it proved to be both fast and reliable even when in use in environments featuring high dimensionality [2], or robots with many degrees of freedom (DOF) [6].

The probabilistic concepts and network structure used in PRM planners provide an effective framework to solve complex path planning problems. However much research has been devoted to improving the details of how these roadmap style planners operate. Observations of the PRM planners operation noted that the majority of computation time is devoted to clash checking robot configurations. In order to reduce the number of clash checks required to generate a solution, many heuristic sampling techniques were developed to replace basic random sampling strategies. Many of these heuristic sampling methods attempt to generate a portion of samples nearby the boundary of clash objects in the workspace. This was done explicitly using geometric information [7] or by using paired samples of specific clash criteria [8]. Other attempts involve partitioning the workspace in order to categorize the different regions in

terms of their complexity [9], [10]. If the sampler knows the 'difficult' regions, it is able to bias its sampling to these spaces, in order to more effectively utilize computational resources. Another focus of research centres on probabilistic methods' weakness in sampling narrow passages in C_{space}. Random sampling schemes employed by traditional PRM planners greatly reduce the likelihood of sampling multiple configurations inside these narrow passages. An effective method of addressing this is the bridge test, as shown in Figure 2 [11]. The bridge test will sample a pair of configurations. If these both fail, the midpoint is tested. If this midpoint is C_{free}, only then is it inserted into $\mathcal{G}$. Bridge testing proves effective at adding samples directly in narrow passages, not just at the boundaries of clash objects and is easily implemented into the structure of network based planners.

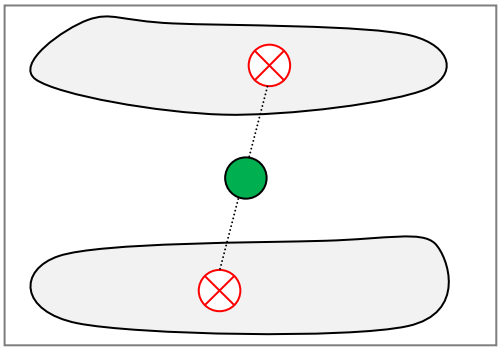


Figure 2. Bridge test criteria.

The 'Lazy-PRM' planner [3] provided a simple and robust method for reducing the number of clash checks to solve a given query. From a high level standpoint, the Lazy-PRM algorithm operates in a similar fashion to its predecessor, the traditional PRM. However a significant difference in operation comes from applying a delayed, or 'lazy', evaluation of clash. The algorithm initially constructs a kinematically sound roadmap, which is assumed to be entirely clash free. Then, repeatedly, the shortest available path through the roadmap is sent to the clash checker for evaluation. If a clash is detected on the path, the offending element (node/edge) is removed from $\mathcal{G}$. If sufficient elements are removed, causing the start and goal configurations to become disconnected, an expansion phase is initiated. The expansion phase adds more nodes in order to reconnect the roadmap. The search then continues iteratively until a continuous clash free path is found, or it is deemed that no solution is available. The Lazy-PRM planner effectively reduces the amount of samples required to solve path planning queries. By delaying the clash checks, redundant sampling of regions that could never provide a solution are removed, saving on computational expenses.

The PRM algorithm spends a large portion of its calculation time pre-processing the roadmap, making it more suited for applications in which a road map can be queried multiple times. Tree based algorithms were developed for applications that needed a solution generated rapidly for single use. [12] developed a notable variant of this style of planner, which grows its $\mathcal{G}$ outwards from an initial root node, one vertex at a time, towards the goal configuration. The growth of the tree can be controlled in a variety of different ways and many approaches have been investigated. Multiple tree approaches have also been made in which two trees are grown simultaneously, with the goal of connecting them together to solve the given query [13].

Recent research focuses on developing planners for specific or complex applications, such as for use on robots with non-holonomic constraints [14], hyper redundant manipulators [6], robots operating in dynamic environments [4], or to generate paths that are more optimal [15]. It is important to note that recent research focuses predominantly on the modification of some component of the probabilistic methodology, rather than on entirely new planning methods; a testament to the reliability and power of probabilistic methods in general.

3. Lazy Significant Edge Algorithm (LSEA)

This section is devoted to describing a new method of lazy path planning. The method offers improvement over the traditional Lazy-PRM planner by reducing the amount of clash checks required to arrive at a solution and is also more effective at navigating difficult regions of C_{space}.

3.1 Algorithm Overview

LSEA, like many other path planners, is an iterative based planner. It constructs an initial roadmap which is then analysed and augmented in a repetitive manner until a termination criteria is met. The algorithm is terminated if a solution to the given query is found, or if no solution is found within a given timeframe. The pseudo code of LSEA is shown in Figure 3. The planner begins by generating a roadmap of kinematically viable nodes and edges, which are initially assumed to be clash free. The algorithm searches for the shortest path through $\mathcal{G}$, which is tested for collision. If a clash is found on this path the subsequent edge/node is removed from $\mathcal{G}$ and the next shortest path is searched for. If no path is returned, the roadmap graph has been reduced to a discontinuous state and a roadmap expansion phase is initiated. The expansion phase re-joins the discontinuous components of the roadmap by intelligently adding nodes to areas of interest. A portion of nodes in this step are generated through uniform random sampling as well, so as to maintain the probabilistic completeness of the algorithm.

1: generate $\mathcal{G}_{init}$
2: **Main Loop**
3: return shortest path, P_s, through $\mathcal{G}_{init}$
4: **If** P_s exists: → clash check P_s
5: **If** ($P_s \in C_{free}$): → solution found. exit **Main loop**.
6: **else** clash detected in P_s → remove element.
7: **else** P_s does not exist: comp (q_{start}) =/= comp (q_{goal})
→ Expansion Phase (see detail)
8: **End Loop**

Expansion Phase
1: Collect all edges, E_{forbid}, in $\mathcal{G}$ ($E_{forbid} \in C_{forbid}$)
2: **for each** $\in E_{forbid}$:
3: add $E_{forbid}(i)$ back into $\mathcal{G}$.
4: **if** comp (q_{start}) = comp (q_{goal})
→ add E_i to significant edge group, SE.
5: **end for**
6: **for each** $\in$ SE
7: distribute N_{samp} nodes about midpoint of SE(i)
8: **end for**
9: lazy bridge test: generate nodes.
10: connect all expansion nodes to $\mathcal{G}$ (component-*n* strategy)
11: **return** main loop

Figure 3. Pseudo code of LSEA operation

The expansion phase is the most important stage of this algorithm. It uses the component connectivity information of the existing roadmap to determine which regions of the workspace have not been successfully navigated by the robot. Roadmap edges which pass through un-navigated clash objects, named significant edges, are used to bias the sampling strategy. The overall effect of the expansion phase is a scheme of sampling that adds more samples to regions of C_{space} that have not been successfully navigated, effectively reducing the amount of redundant sampling carried out. In addition to the roadmap expansion, a type of 'lazy' bridge test (LBT) is implemented. LBT is only activated in certain instances and uses prior clash information to reduce the computational expense of traditional bridge testing techniques. Details of the algorithms' components and their methods of operation are detailed in the following chapter sections.

3.2 Construction of Initial Roadmap

To build the initial roadmap ($\mathcal{G}_{init}$), N_{init} configurations are generated using uniform random sampling about the robots C_{space}. These nodes are then connected to the nearest N_{neighb} nodes using the *component-n* [16] connection strategy. If an edge is not kinematically feasible (e.g., out of joint or orientation limits) it is not included in $\mathcal{G}_{init}$. At this stage, all nodes and edges in $\mathcal{G}_{init}$ are assumed to be clash free. As the algorithm progresses, the values of N_{init} and N_{neighb} will have a significant effect on both the optimality of the returned path and the time taken to find a solution and so should be selected carefully for the given path planning problem.

3.3 Selection of shortest path and clash test

There exist many different methods of selecting the shortest path through a connected network graph, each with varying levels of complexity and performance. LSEA utilizes the A^* search algorithm [17] to return the shortest path through $\mathcal{G}$ between q_{start} & q_{goal}.

When the shortest path through the roadmap is found, each element in the path is checked for clashes. All nodes along the path are tested first, followed by the edges. If any node/edge is found to be clashing, the process is terminated immediately and the offending element is removed from $\mathcal{G}$. If it is a node that is removed, all edges that are connected to the node are removed as well. If the entire path is clash free, we have found a solution to the given query and the path planning process is successfully completed. To save processing time, all results of clash tests are stored in the roadmap structure, so that these elements do not need to be tested again over future iterations.

3.4 Iterative Expansion Phase

In a complex path planning problem, $\mathcal{G}_{init}$ will not be sufficient to solve the given search query. When the query fails, we are left with a disconnected roadmap consisting of at least two separate components. Upon reaching this condition, we initiate the expansion phase of LSEA. The roadmap is augmented by analysing its current state and placing new samples in areas that have not yet been navigated by the robot. These areas are found by utilizing roadmap component connectivity information to determine a collection of significant edges about the roadmap. The expansion phase then actively samples about these significant edges. Following this, if certain conditions are met, lazy bridge testing is carried out in an attempt to place clash free nodes in narrow passages of the workspace.

3.4.1 Significant edge search.

To determine a list of significant edges, we initially collect a group of candidate edges; edges that have been removed from $\mathcal{G}$ (note that edges that were removed with nodes are not considered as candidates). For each of these edges, we place them back into $\mathcal{G}$ and check to see if the start and goal queries have been rejoined into the same component. If a candidate edge satisfies this condition we have successfully defined a significant edge. Significant edges always occur about a clash object that has not been navigated by the robot yet. If an edge is obstructed by a clash object, but does not reduce the component count (refer detail *iv* in figure 4), the clash object has effectively already been navigated by the robot. Once a collection of significant edges has been determined, we randomly select a portion of them to be used as the basis for sampling during the expansion phase.

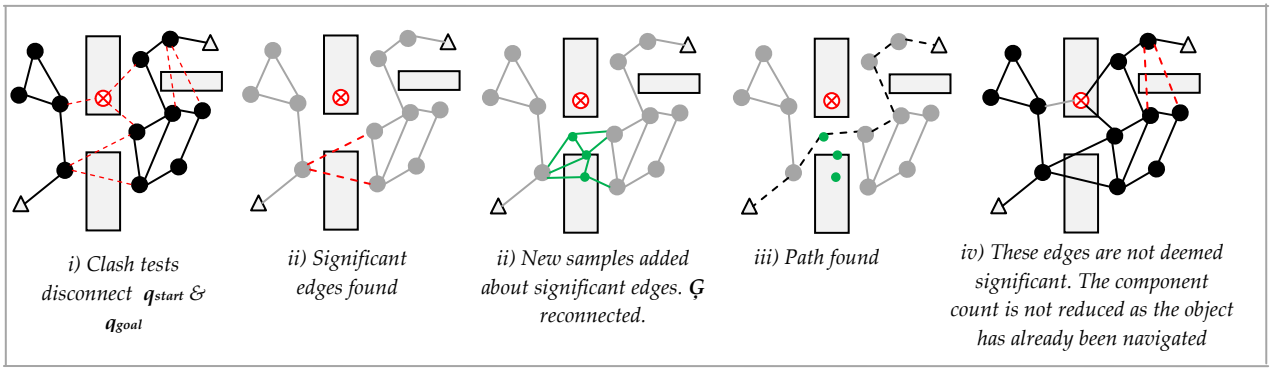


Figure 4. Graphic overview of the LSE algorithm operation

N_{samp} samples are placed about each significant edge via the use of a bivariate distribution [3], centred at the mid-point of the edge. The shape/size of the distribution is controlled by the length and direction of the edge. The newly distributed nodes are then connected to Ģ in the same manner as during the construction phase. After all significant edges have been sampled, N_{rand} nodes are also distributed randomly about the roadmap, in order to ensure probabilistic completeness and help the expansion of the roadmap over future iterations. In some rare instances no significant edges are present in Ģ, which indicates that whilst the current query has failed, the general exploration of C_{free} is progressing. In these instances, random sampling carried out as normal to re-connect the roadmap and the algorithm progresses as normal. Once the expansion phase is completed, the roadmap's connectivity is returned to a sufficient state to continue the planners operation. The expanded roadmap is passed to the shortest path search and the algorithm continues its iterations. To reduce cyclic behaviour in roadmap expansion, significant edges are not used multiple times. If a significant edge has been defined once, it is not utilized by expansion phases over future iterations.

3.5 Lazy Bridge Tests (LBT)

Bridge testing is a proven method of effectively navigating complex regions and narrow passages [11]. The bridge test operates by testing pairs of samples, within a given distance of one another, continually until both samples are found to be clashing. Once this criteria is met, the midpoint of these samples is tested and only if this configuration is C_{free} is it then placed in Ģ. Bridge testing ensures that samples are placed in narrow regions of the configuration space, however the method requires that three successive clash tests of a specific criteria must be met in order to place only one single node in Ģ.

In order to increase the efficiency of the traditional bridge test, a lazy method of bridge testing is implemented in LSEA. It operates by first collecting all nodes in Ģ that failed their clash test, or lie in C_{forbid}. For each pair of nodes in this set (within distance D_{bridge} of each other) a sample is generated at the midpoint. If the mid-point lies in C_{free}, we have satisfied the bridge criteria and this sample is added to Ģ. By utilizing previously determined C_{forbid} samples, the number of clash checks required to produce a bridge sample in Ģ is reduced. The Lazy Bridge test requires a sample of previously clash checked nodes, so it is implemented after each iteration of the expansion phase. To avoid cycles, once a bridge between two C_{forbid} samples has been found, it is never used again. In certain instances, LBT is not needed. At the end of each iteration of the main algorithm, if a certain ratio of failed samples to good samples produced is not met, LBT is not carried out.

4. Experiment and Results

4.1 Experimental Setup

Three path planning problems, each featuring varying levels of complexity, were created to test LSEA. To evaluate performance, LSEA was benchmarked against the Lazy-PRM path planner [3]. Both algorithms were run 100 times in each environment and the results were averaged. Both algorithms utilize the same set of variables that control the operation of the planner. For each environment, these variables were optimized (see Table 1.) and implemented before the testing process. Both planners made use of the *component-n* [16] neighbour connection strategy, which provided better results for both planners in each of the given environments. The effectiveness of lazy bridge testing techniques was carried out separately. In the *medium-scatter* environment, LBT was compared to traditional bridge testing as a means to effectively sample inside narrow passages. All algorithms and environments were developed and tested in the MATLAB programming language.

Each of the three environments is a 2D plane scattered with randomly distributed objects to navigate. The robot is a planar robot with three degrees of freedom (*x, y* and *rotation Rz*). The *sparse-scatter* environment (Figure 5. detail i) features oddly shaped objects, sparsely distributed about the configuration space. The robot is

relatively free to move about this environment and the robots orientation is not critical in order to pass between many of the objects. The second environment, *medium-scatter*, has more objects dispersed throughout. Their positions create several narrow passages the robot has to navigate in order to solve the given path planning query. The final environment has many smaller clash objects *densely-scattered* at random. The density of the clash objects drastically restricts the robot's motion and the orientation of the robot is critical in order to navigate through many sections of the workspace.

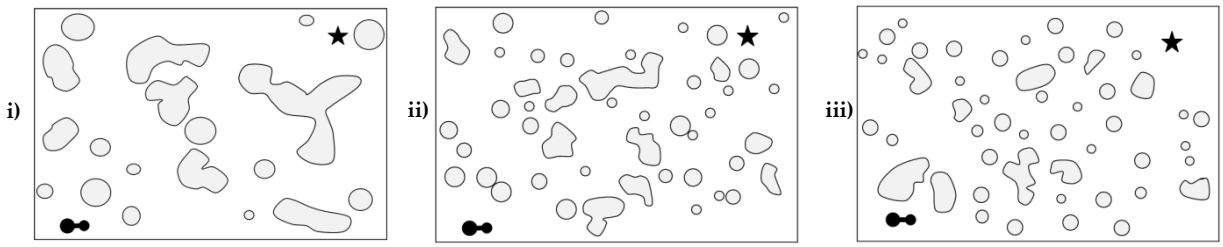


Figure 5. The testing environments, the robot is shown in the bottom left corner, and the goal is the black star in the top right. Detail: i) Sparse-Scatter ii) Medium-Scatter iii) Dense-Scatter

4.2 Results

4.2.1 Results of LSEA

The box plots in Figure 6 display the times taken per query for both planners. The whiskers encapsulate the entire spread of the data, the shaded box represents the lower quartile and the lighter shaded box above shows the upper quartile of the data. The boundary between these boxes represent the median time taken to complete the tests. Using the box plots, we can see that LSEA outperforms the traditional lazy planner in every environment. The improvement was smaller in the sparse scatter environment, but became more apparent in the medium and densely scattered environments.

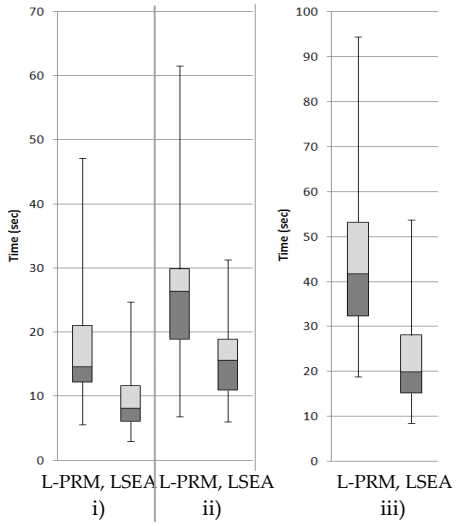


Figure 6. Box plots of results. i) Sparse-scatter ii) Medium-scatter and iii) Dense-scatter.

In the sparsely-scattered environment LSEA provided some degree of improvement. It was observed that the sparseness of the obstacles meant that both planners were able to solve the majority of the problems with the initial roadmap alone, few expansion iterations were required. Both algorithms use the same methods for generating $\mathcal{G}_{init}$, so their results are fairly similar. Regardless, the average time to find a solution was reduced by 40% and the spread of the results was also reduced somewhat. The advantages of LSEA became more apparent in the remaining environments. In the *medium-scatter* test, LSEA reduced the average time to reach a solution by 52% and the spread of data was reduced significantly. This result can be attributed to LSEA distributing its sampling workload towards areas that had not effectively been navigated by the robot. The traditional Lazy algorithm was observed redundantly sampling regions already navigated, slowing the average time to reach a solution. This same phenomenon was observed to a greater degree in the *densely-scattered* environment. The average time taken to solve the problem was reduced by 57%. LSEA was more intelligently able to discern which regions had been cleared by the robot and then push the sampling strategy away from these areas. Results from the testing are presented in Figure 7. The reduction in time for LSEA can be directly attributed to the reduced number of clash checks done in order to solve each path planning query. The roadmaps generated to solve the problems had fewer nodes and the percentage of roadmap nodes that were clash checked (shown in the brackets) was also lower, providing a more 'lazy' solution.

4.2.2 Results of LBT

To test the effectiveness of Lazy Bridge testing, another testing scheme was created using the *medium-scatter* environment. This environment was used as it features scattered obstacles, as well as narrow passages which need to be navigated, providing a good environment to test the overall performance of LBT. To gain both an

absolute and relative understanding of LBT effectiveness, two tests were created. In the first, the *medium-scatter* environment was used to test LSEA with and without LBT. LBT was found to improve the time taken to solve the query by 33.1%. When the planner is using LBT, fewer samples are required, because they are more effectively used during the process. The second test compared the effectiveness of LBT against traditional bridge testing techniques. Once again, the lazy method showed improved performance over the traditional method. A 17.8% reduction in time was observed and on average, 2934 fewer clash checks were required to reach a solution. These results can be attributed to the fact that LBT utilizes previously calculated clash results, reducing the overall amount of clash tests required to generate valid bridge samples. Granted, LBT has to substitute many more distance checks between nodes. However, the relative cost of these distance checks is small compared to the cost of the extra clash checks required when carrying out the traditional bridge testing. If LBT was employed on a more complex system, which requires a longer time to check the clash status of samples, a greater reduction in time would be observed.

100 tests. Sparse.	Lazy-PRM	LSEA
AveTime (s)	15.2	9.1
Ave Clash Checks	66898	44525
Nodes Used (% CC)	308 (78%)	186 (75%)

i)

100 tests. Medium.	Lazy-PRM	LSEA
AveTime (s)	28.8	13.9
Ave Clash Checks	120712	65348
Nodes Used (% CC)	385 (84%)	241 (80%)

ii)

100 tests. Dense	Lazy-PRM	LSEA
AveTime (s)	43.1	18.3
Ave Clash Checks	177903	83522
Nodes Used (% CC)	444 (90%)	249 (85%)

iii)

Figure 7. Tabulated results for each environment: i) Sparse-scatter ii) Medium-Scatter iii) Dense-Scatter

5. Discussion

LSEA was developed as an improvement over the traditional Lazy-PRM planner [3]. From a high level standpoint, both algorithms operate in a similar fashion. The main operational difference between LSEA and Lazy-PRM comes from the roadmap expansion phase. Lazy-PRM carries out its roadmap expansion by adding additional samples to regions deemed likely to be at the boundary of clash objects. LSEA takes this expansion method a step further by adding its samples to regions nearby clash objects that have not yet been navigated by the robot. By doing this a solution can be discovered faster, as no time is wasted by continuing to sample about a clash object that has already been cleared.

The trade off to this approach is that whilst a solution can be found faster, there is the possibility that the optimality of the returned path may not be as good. If, for instance, the LSEA planner navigates a certain obstacle poorly, the likelihood of adding more samples to this region and improving this local path over future iterations is low. Whereas, if the Lazy-PRM navigates a certain object poorly, there is a greater chance that future iterations of the algorithm will continue to place samples about this object, which could improve the local path. This effect can be reduced considerably via post process optimization of the path solution.

The observed performance advantage LSEA has over Lazy-PRM is directly attributed to the reduced amount of clash checks required to solve a path planning query. In the 2D test environments used, these clash checks are relatively simple and can be carried out very rapidly (~0.0001 sec). If LSEA planner was implemented in a more complex system that requires a much longer time to process a clash check, we can expect the performance advantage of LSEA to be even higher.

Variable	Value
N_{Init}	70,150,150
N_{Neighb}	5,5,5
# Seeds/SE's per iteration	10,10,10
N_{samp}	2,2,2
N_{rand}	10,10,10

Table 1. Variables used for each of the three environments (sparse, medium, dense) for both LSEA and Lazy-PRM during testing.

6. Conclusions and Future Work

In this paper, a new path planning algorithm that utilizes connectivity information to bias the expansion of the roadmap was presented. The algorithm was tested in three environments featuring different levels of complexity. Its performance was benchmarked against a traditional Lazy-PRM planner. In each test LSEA outperformed its counterpart considerably, despite having a similar method of operation. It was able to solve path planning queries faster, utilizing fewer samples to achieve its solution. In more difficult environments, it was able to process a valid solution twice as fast as its traditional counterpart. A lazy method of bridge testing was also proposed and tested. The lazy method was observed to sample narrow passages more effectively, utilizing fewer clash checks in order to generate valid bridge samples.

Future work regarding LSEA involves adapting its operation to plan paths for 6DOF articulated manipulators operating in a 3D environment. In such a system, clash checking algorithms place an even larger burden on computational resources. It is envisaged that

when LSEA is implemented on such a system, the reduction in required clash checks will further increase the performance advantage LSEA has over traditional Lazy-PRM methods. The LSEA algorithm is still in early stages of development. Many performance modifications are being investigated and further tuning of the algorithm for increased performance is to be carried out

7. Acknowledgments

This work was supported by the Defence Materials Technology Centre (DMTC), which was established and is supported by the Australian Government's Defence Future Capability Technology Centre (DFCTC) initiative.

8. References

[1] Tsianos K, Sucan I, Kavraki L (2007) Sampling-Based Robot Motion Planning: Towards Realistic Applications. Computer Science Rev. 1: pp 2-11.

[2] Kavraki L, Svestka P, Latombe JC, Overmars M (1996) Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. IEEE Trans. On Robotics and Automation. 12: pp 566-580.

[3] Bohlin R, Kavraki L (2000) Skin Path Planning Using Lazy PRM. Proc. IEEE Int. Conf. on Robotics & Automation. California, USA.

[4] Jaillet L, Simeon T (2004) A PRM-based Motion Planner for Dynamically Changing Environments. IEEE Int. Conf. on Intell. Robots & Systems. 2: pp 1606-1611, Sendai, Japan.

[5] Thomas U, Iser R (2010) A new Probabilistic Path Planning Algorithm for (Dis)assembly Tasks. ISR 41st International Symp. On Robotics. Munich, Germany.

[6] Lanteigne E, Jnifene A (2011) Small Tree Probabilistic Roadmap Planner for Hyper-Redundant Manipulators. Lec Notes in Comp. Science. Vol 6752. pp 11-20.

[7] S.A. Wilmarth, N.M. Amato, P.F. Stiller (1999) MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. IEEE 1999 Int. Conf. on Robotics and Automation, pp 1024-1031.

[8] Boor V, Overmars M, Stappen F.V.D (1999) The Gaussian Sampling Strategy for Probabilistic Roadmap Planners. Proc. of the 1999 IEEE Int. Conf on Robotics & Automation. USA.

[9] Rantanen M (2011) A Connectivity-Based Method for Enhancing Sampling in Probabilistic Roadmap Planners. J. Intell Robot Syst. 64: pp 161-178.

[10] Klasing K, Wollherr D, Buss M (2007) Cell-based Probabilistic Roadmaps (CPRM) for Efficient Path Planning in Large Environments. ICAR. 13th Int. Conf. on Adv. Robotics. pp 1075-1080.

[11] Sun Z, Hsu D, Jiang T, Kurniawati H, Reif J (2005) Narrow Passage Sampling for Probabilistic Roadmap Planning. IEEE Trans. On Robotics. 21: pp 1105-1115.

[12] Kuffner J, LaValle S (2000) RRT-Connect: An Efficient Approach to Single-Query Path Planning. Proc. IEEE Int. Conf. on Robotics & Automation. pp 995-1001.

[13] Sanchez G, Latombe JC (2003) A single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking. Springer Tracts in Adv. Robotics. 6: pp 403-417.

[14] Cheng P, Shen Z, LaValle S (2000). Using randomization to find and optimize feasible trajectory for nonlinear systems. Allerton Conf. on Communications, Control, Computing, pp 926-935

[15] Nieuwenhuisen D, Overmars M, (2004) Useful Cycles In Probabilistic Roadmap Graphs. Proc. IEEE Int. Conf. Robotics & Automation. 1: pp 446-452

[16] Geraerts R, Overmars M (2005) Sampling and node adding in probabilistic roadmap planners. Robotics and Autonomous Systems. 54: pp 165-173.

[17] Latombe JC (1991) Robot Motion Planning. Kluwer, Boston, USA.

2013 IEEE/ASME International Conference on
Advanced Intelligent Mechatronics (AIM)
Wollongong, Australia, July 9-12, 2013

Path Planning for Industrial Robots; Lazy Significant Edge Algorithm (LSEA)

Joseph Polden, Zengxi Pan, and Nathan Larkin

Abstract — **This paper presents a new sampling based path planning algorithm, called the Lazy Significant Edge Algorithm (LSEA). LSEA utilises roadmap connectivity information to bias its sampling strategy towards objects in a robots workspace that have not yet been navigated by the robot. This allows LSEA to avoid redundant sampling of configuration space. The robotic system used in this paper to test LSEA consists of an articulated industrial manipulator mounted on a linear rail. LSEA was tested on this system with a series of different path planning problems in order to judge its overall effectiveness. When compared to a number of other popular sampling based path planning algorithms, it was concluded that LSEA had the best overall performance. It was observed to solve the various path planning problems more quickly than its counterparts, utilising fewer clash checks in order to reach the various solutions.**

The full article below removed for copyright reason, please refer to:
Polden, J., Pan, Z. & Larkin, N. (2013). Path planning for industrial robots; Lazy Significant Edge Algorithm (LSEA). *2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)* (pp. 979-984). United States: IEEE.
doi: 10.1109/AIM.2013.6584221

Adaptive Partial Shortcuts: Path Optimization for Industrial Robotics

Joseph Polden[*,1, a], Zengxi Pan[1,b], Nathan Larkin [1,c], Stephen van Duin [1,d]

[1]Faculty of Engineering, University of Wollongong

Northfields Avenue, Wollongong NSW 2522, Australia

[a]jwp973@uowmail.edu.au +6142218132,

[b]zengxi@uow.edu.au, [c]nlarkin@uow.edu.au, [d]svanduin@uow.edu.au

* Corresponding Author

Abstract. The quality of a path generated from an automated motion planning algorithm is of considerable importance, particularly when used in a real world robotic application. In this work a new path optimization algorithm, called the Adaptive partial shortcut algorithm, is presented. This algorithm optimizes paths as a post process to motion planning, and was designed specifically for use on an industrial manipulator. The algorithm optimizes robot degrees of freedom individually allowing it to produce manipulator paths of particularly high quality. Moreover, this new algorithm utilizes an adaptive method of selecting the degree of freedom to optimize with each iteration, giving it a high level of efficiency. Tests conducted in this paper prove the effectiveness of the algorithm: Over a range of different test paths, the adaptive algorithm was able to generate solutions with a 61% reduction in collision checks than the original partial shortcut approach.

Keywords: Optimization, Probabilistic Roadmap, Post Process, Robotic Manipulator.

1. Introduction

The use of robotic manipulators for industrial applications, such as welding, painting or assembly tasks, has increased considerably in recent years [1]. Whilst the cost of industrial robotic hardware is decreasing, programming these systems remains time consuming and costly [2]. For more complex industrial applications, Offline Programming (OLP) methods have proven effective. OLP utilizes CAD models of the robotic system and its surrounding environment to plan and simulate robot programs before uploading them into the real world robotic system for use. One particular benefit of this virtual approach is that many of the programming steps can be automated. For example, weld torch orientations and robot positioning can be automatically assigned and optimized, which saves time and improves the quality of the generated robot program.

Motion planning is one component of OLP that has been automated by use of computer algorithm. A motion planning algorithm generates a series of motions for a robot so that it can navigate the surrounding environment without collision. However for an industrial robotic system, featuring 6 Degree of Freedom (DOF) articulated manipulators and high dimensional environments, the general motion planning problem becomes difficult to solve. Over the years a number of different approaches have been developed in attempts to solve these problems. Early approaches utilized explicit mathematical models of the surrounding environment, so that the robot can then navigate about it. A number of these approaches were effective for simplistic robotic systems, but in high dimensional cases they were found to be either slow or unable to solve the given problem. A more recent approach, the Probabilistic Roadmap Method (PRM) [3], was developed as an efficient and effective means to solve difficult problems. Even the most basic PRM variants were able to rapidly generate paths for a robotic manipulator moving about high dimensional configuration spaces [3]. A common drawback of this approach is the quality of the generated paths, which are often too long and exhibit jerky or redundant motions. If these paths are to be used in real world robotic applications, optimization is required.

The Partial Shortcut (P-Sc) algorithm, developed by [4], is particularly effective at optimizing PRM paths generated for robotic manipulators. Paths are optimized one DOF at a time, resulting in reduced path length and positional overshoot. Its reliance on probabilistic methods removes any need for complex mathematical representation of the surrounding environment. The disadvantage of the P-Sc algorithm is the increased computational effort required to optimize each robot DOF individually. In this paper a new partial shortcut algorithm, which adaptively selects robot DOFs to optimize, is presented. This new algorithm improves upon the original approach by reducing the number of collision checks required to optimize a given path.

1.1 Related Work

Probabilistic sampling based algorithms, such as the PRM planner, operate by generating a sample of random robot configurations about the surrounding free space. As collision free configurations are found, they are added as nodes to a network graph, often referred to as the roadmap. A local planner, which tests robot motions for collision, then attempts to link nearby nodes together. Collision free motions are added to the roadmap graph as edges between respective nodes. The graph is incrementally built until a suitable level of coverage has been reached. Graph searching algorithms can then trace a path through the roadmap from one configuration to another to solve a given motion planning problem. By utilizing this simplified representation of the free space, a complex motion planning problem in high dimensional configuration space can be resolved without using an explicit mathematical expression of the environment. The drawback however, is the quality of the path produced. The measurement of path quality is dependent on the robotic system used. Typically, an optimization algorithm will attempt to improve path quality by reducing overall distance. However in certain applications, other factors must also be considered. Examples include; maximizing clearance between a given robot and its surrounding environment [5], reducing overshoot [4], controlling orientations [6] or improving the dynamic qualities of non-holonomic motion [7].

The process of optimizing robot paths can generally be classified into two approaches; optimization embedded in the motion planning process, and optimization as a post-process to motion planning. A popular approach to embedded optimization involves the modification of roadmap graphs to store extra information about the environment as the roadmap is constructed. In [8], the definition of an edge cost was expanded to include weighted values relating to minimum clearance and changes in direction between successive edges. A modified Dijkstra's graph searching algorithm then evaluates the multiple criteria when searching for the minimum cost path. In [9] additional edges are selectively sampled into a tree based roadmap graph during the construction phase, and in [6] the growth of the network graph is controlled to ensure end effector orientations are kept within a set range. In [10], optimization is performed by continuing roadmap construction processes if a path of unsuitable quality is returned. Other variants of this concept were explored in [7] and [11], where a suboptimal path is used to define a reduced region of space to refine the motion planning process. Gradient decent techniques which combine motion planning and optimization were explored in [12] and [13], where explicit mathematical models are used to guide a robotic agent towards a goal configuration.

The path pruning and shortcut techniques shown in Figure 1 are examples of post-process optimization. They are simple in their implementation and operation, making them easily adapted to different robotic systems. In [14] and [15] redundant nodes are removed from a path by searching for valid routes that bypass them. In [16] the same process is carried out, but over larger segments of the path which are selected at random. Reference [17] used a similar approach by shortcutting between mid-points of edges, and in [18] this approach is extended by adding a number of additional configurations to the path, creating more potential for an optimal solution. Other approaches use one or more of these techniques in conjunction with some form of path smoothing in order to achieve a short path with smooth transitions between motions [19]. Both [4] and [20] optimized clearance by retracting the input path to a local medial axis before applying shortcut techniques. In [4] a novel adaptation to these generic shortcut methods, called the partial shortcut (P-Sc) algorithm, was developed. This approach is similar to the multiple segment shortcut; however it only interpolates a shortcut to one movable degree of freedom at a time, as represented diagrammatically in Figure 1, detail e).

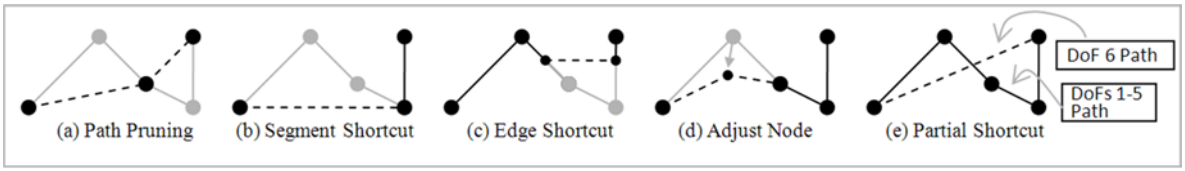


Figure 1. Examples of Generic Shortcut Techniques

1.2 Motivation

A robotic manipulator will typically only need to move a small portion of its joints in order to navigate an arbitrary obstacle. Motion of any of the other manipulator joints is redundant, as they are not required for this navigation. Whilst generic path pruning and shortcut approaches are effective at shortening the overall length of a manipulator's path, the optimized results often still exhibit redundant joint motion. This is because the shortcuts used are interpolated over all of the manipulators joints simultaneously. The P-Sc method, on the other hand, applies its shortcuts to single manipulator joints at a time. This approach provides an effective means to reduce this redundant joint motion. In doing this, harsh changes in joint trajectories between successive motions are also reduced. The P-Sc algorithms' probabilistic approach and use of roadmap style paths also simplifies is operational details, making it easy to understand and implement into an existing OLP system.

The approach of optimizing single robot DOFs at a time, however, means that more computational effort is required in the optimization process. Far more collision checks are needed to reduce the path length to a suitable level. This disadvantage becomes more prominent in industrial OLP applications where the large number of obstacles leads to a slower collision processing time; reducing overall usefulness of the partial shortcut approach. To address this particular weakness a new adaptive approach to partial shortcuts, designed specifically for use in industrial applications, was devised. The goal was to develop this algorithm so that it operates more efficiently by reducing the amount of collision checks required to optimize a given path.

The following paper first presents a summary of relevant notation. The partial shortcut approach is then described, and the Adaptive Partial Shortcut algorithm is introduced. Chapter 4 evaluates the effectiveness of this new algorithm with a series of tests on an industrial robotic manufacturing system. Chapter 5 features a discussion on the partial shortcut method and its effects on a manipulator robot, which is followed by the papers conclusion.

2. Relevant Notation

Sampling based motion planning and optimization algorithms operate on the premise of a robot moving about its surrounding workspace *W*, without colliding with any obstacles. To properly characterize this foundation, some relevant notation is described. A robot configuration π_i is defined by the set of numbers representing values of the individual robot DOFs needed to specify the robot position in *W* i.e. $\pi_i = [q_1, \dots, q_{q_n}]$ where q_n is the number of DOFs of the robot. The f^{th} value of configuration π_i can be referenced as $\pi_i[f] = q_f$.

The configuration space of a robot C_{space}; is the set of all possible robot configurations. By sampling robot configurations with a collision checking algorithm, two distinct subsets of C_{space} are defined: The free configuration space C_{free}; the set of all robot configurations that are not in collision with any surrounding obstacles, and the forbidden configuration space C_{forb}; the set of all robot configurations that are in collision.

A local path Π_i is a single motion from one individual robot configuration; π_i, to another; π_{i+1} where $\Pi_i \in C_{free}$.

$$\Pi_i = \{\pi_i, \pi_{i+1}\}$$

The motion of Π_i is defined using a local path planner, usually a linear interpolation in either Joint or Cartesian space. A local path that is kinematically feasible and resides completely within C_{free} is said to be valid.

A path $\overline{\Pi}$ is an ordered series of *n* local paths, i.e. $\overline{\Pi} = \{\Pi_1, \dots, \Pi_n\}$. A path cannot feature discontinuities i.e. a robotic device must be able to move from the initial configuration π_1 of Π_1 to the final configuration π_{n+1} of Π_n.

For an industrial manipulator, the length of paths are described using a joint cost function $c_J(\dots)$, which produces an array containing the change in displacement used by each robot DOF to travel the path. This array is referred to as the joint cost array. For a local path Π_i, which features only one direct motion:

$$c_J(\Pi_i) = [|\pi_i[1] - \pi_{i+1}[1]|, \dots, |\pi_i[q_n] - \pi_{i+1}[q_n]|] \qquad (1)$$

The resultant cost array contains the change in displacement of each robot DOF used to travel from π_i to π_{i+1}.

For a path $\overline{\Pi}$, we sum the joint costs across the *n* local paths contained within it i.e.

$$c_J(\overline{\Pi}) = \left[\sum_{i=1}^{n} |\pi_i[1] - \pi_{i+1}[1]|, \dots, \sum_{i=1}^{n} |\pi_i[q_n] - \pi_{i+1}[q_n]|\right] \qquad (2)$$

The resultant cost array produces the change of displacement of each robot DOF used to travel from π_1 of Π_1 to π_{n+1} of Π_n.

In the tests conducted in chapter 4, the P-Sc algorithm is terminated using a total cost function $c_T(\dots)$. The function produces one single value and is a summation of each element in the joint cost array produced from $c_J(\dots)$;

$$c_T(\overline{\Pi}) = \sum c_j(\overline{\Pi}) = \sum \left[\sum_{i=1}^{n} |\pi_i[1] - \pi_{i+1}[1]|, \dots, \sum_{i=1}^{n} |\pi_i[q_n] - \pi_{i+1}[q_n]|\right] \qquad (3)$$

3. Adaptive Partial Shortcut Algorithm

3.1 The Partial Shortcut Algorithm

Path optimization algorithms such as the path pruning and shortcut methods reduce the overall path cost by bypassing robot configurations wherever possible. The P-Sc method, shown in Algorithm I, instead operates by decoupling the robot DOFs, and optimizing path configurations one DOF at a time.

The P-Sc algorithm starts by selecting a robot DOF, *f*, randomly from a predefined weighted distribution. This distribution is spread across all available robot DOFs, and is used to skew this random selection towards particular DOFs of interest. The input path $\overline{\Pi}$ is then split into three segments denoted $\overline{\Pi}', \overline{\Pi}'', \text{and } \overline{\Pi}'''$, using the randomly generated integers a and b, where *n* is the number of local paths belonging to $\overline{\Pi}$:

$$1 < a + 1 < b \leq n \text{ and}$$

$$\overline{\Pi}' = \{\Pi_1, \dots, \Pi_a\}, \overline{\Pi}'' = \{\Pi_{a+1}, \dots, \Pi_b\}, \overline{\Pi}''' = \{\Pi_{b+1}, \dots, \Pi_n\}$$

For the random DOF f, all configurations $\pi_i''|i = 1 \dots m+1$ in $\overline{\Pi}''$ are replaced by the equivalent value linearly interpolated between 1 and $m+1$, where: $m = b - a$ is the number of local paths in $\overline{\Pi}''$. If the f^{th} value of configuration π_i'' is $\pi_i''[f]$, then

$$\pi_i''[f] = {}^{int}\pi_i''[f] \mid i = 1 \dots m + 1 \qquad (4)$$

Where ${}^{int}\pi_i''[f]$ is the i^{th} interpolated value of the f^{th} DOF between $\pi''_1[f]$ and $\pi''_{m+1}[f]$. If the modified segment is found to be valid, i.e.

$$\overline{\Pi}'' \in C_{free}$$

then the path section $\overline{\Pi}''$ is inserted back to the original path.

$$\overline{\Pi} = \overline{\Pi}' \cup \overline{\Pi}'' \cup \overline{\Pi}'''$$

Otherwise, $\overline{\Pi}''$ generated from this iteration is discarded. This process is then repeated iteratively until a termination criterion is met.

The P-Sc algorithm does not add any additional samples to, or around, the input path. Nor are any removed. P-Sc relies solely on modifying the configuration of path nodes. Since the algorithm only applies shortcuts to a single robot DOFs at a time, it proves less efficient at reducing overall path length than other shortcut methods. Isolating the interpolation to a single joint, however, provides an effective means to remove excessive joint overshoot.

Algorithm I PARTIALSHORTCUT ($\overline{\Pi}$)

input $\overline{\Pi}$ – Path for optimisation
output $\overline{\Pi}$ – The optimised path
1: $n \leftarrow length(\overline{\Pi})$
2: **loop**
3: $f \leftarrow$ a random degree of freedom
4: $a, b \leftarrow$ two random indices: $1 < a + 1 < b \leq n$
5: $\overline{\Pi}' \leftarrow \overline{\Pi}\{\Pi_1, \dots, \Pi_a\}$
6: $\overline{\Pi}'' \leftarrow \overline{\Pi}\{\Pi_{a+1}, \dots, \Pi_b\}$
7: $\overline{\Pi}''' \leftarrow \overline{\Pi}\{\Pi_{b+1}, \dots, \Pi_n\}$
8: $m \leftarrow (b - a)$
9: **for** $i = 1$ **to** m
10: $\pi_i''[f] \leftarrow \text{INTERPOLATE}\left(\pi_i''[f], \pi_{m+1}''[f], i/(m+1)\right)$
11: **end**
12: VALIDATE PATH($\overline{\Pi}''$)
13: **if** $\overline{\Pi}'' \in C_{free}$ **then**
14: $\overline{\Pi}'' \leftarrow \overline{\Pi}' \cup \overline{\Pi}'' \cup \overline{\Pi}'''$
15: **end**

3.2 Adaptive Partial Shortcuts

It was found that when operating on an industrial robotic system, the P-Sc algorithm spends the majority of its computation time, up to 95%, processing collision checks. Therefore, reducing the number of collision checks required to optimize a path provides an effective means to improving the overall efficiency of the algorithm. This reduction will be more effective in an industrial manufacturing environment, where collision checks are more computationally expensive to process. In this chapter a new adaptive partial shortcut algorithm is presented. The new approach is based on the original P-Sc algorithm described in section 3.1, and features a dynamic method of selecting partial shortcuts. The pseudo-code for the approach is given in Algorithm II, and its operational details are presented below.

Algorithm II ADAPTIVE-PARTIALSHORTCUT($\overline{\Pi}$)

input $\overline{\Pi}$ – Path for optimisation
output $\overline{\Pi}$ – The optimised path
1: $n \leftarrow length(\overline{\Pi})$
2: $\Pi_{ref} = \overline{\Pi}\{\pi_1, \pi_n\}$
3: **loop**
4: $AWD \leftarrow \text{genAWD}(\overline{\Pi}, \Pi_{ref})$
5: $f \leftarrow \text{selectDOF}(AWD, nDoF)$
6: $a, b \leftarrow$ two random indices: $1 < a + 1 < b \leq n$
7: $\overline{\Pi}' \leftarrow \overline{\Pi}\{\Pi_1, \dots, \Pi_a\}$
8: $\overline{\Pi}'' \leftarrow \overline{\Pi}\{\Pi_a, \dots, \Pi_b\}$
9: $\overline{\Pi}''' \leftarrow \overline{\Pi}\{\Pi_b, \dots, \Pi_n\}$
10: $m \leftarrow (b - a)$
11: **for** $i = 1$ **to** m
12: $\pi_i''[f] \leftarrow \text{INTERPOLATE}(\pi_i''[f], \pi_m''[f], i/(m))$
13: **end**
14: **if** $c_T(\overline{\Pi}'') < c_T(\{\Pi_a, \dots, \Pi_b\})$
15: VALIDATE PATH($\overline{\Pi}''$)
16: **if** $\Pi'' \epsilon\, C_{free}$ **then**
17: $\overline{\Pi}'' \leftarrow \overline{\Pi}' \cup \overline{\Pi}'' \cup \overline{\Pi}'''$
18: **end**
19: **end**

3.2.1 *Adaptive Weighting Distribution*

On line 3 of Algorithm I, a single robot DOF *f* is selected from a weighted distribution. For example [4] used a weighting of [6,6,6,2,2,2] to bias the selection of *f* for an articulated industrial robot with 6 DOF towards the robot's major joints (J1, J2, J3) over the minor joints (J4, J5, J6). Generally, this distribution works well as the optimization goal is to minimize total Cartesian motion. However there are certain situations where this fixed distribution may have a negative impact on the overall performance of the P-Sc algorithm; for instance when trying to optimize a path with excessive joint motion on joints J4, J5 and J6. An adaptive approach to bias the selection of *f* is proposed to address this issue.

The *Adaptive Weighting Distribution* (AWD), shown in Algorithm III, is a distribution that is updated at each iteration of the P-Sc algorithms main loop. The AWD method aims to generate a distribution that will skew the selection of *f* towards the joints that deviate furthest from the optimal solution, Π_{ref}. For the input path $\overline{\Pi}$, the reference path Π_{ref} is a local path travelling directly from the start configuration π_1 to the goal π_{n+1}, it assumes that no obstacles are present in the problem:

$$\Pi_{\text{ref}} = \{\pi_1, \pi_{n+1}\} \text{ where } C_{\text{space}} = C_{\text{free}}$$

Used as a reference, Π_{ref} is the most optimal path possible for a particular motion planning problem, as the local planner directly joins the start and goal configurations without any of the excess motion needed to avoid obstacles about C_{space}.

In Algorithm III, joint cost arrays for the local path Π_{ref} and the current path being optimized, $\overline{\Pi}$, are calculated using the joint cost functions (1) and (2). The AWD is then produced from the difference between resultant joint cost arrays;

$$\text{AWD} = \left[\left|\left[c_J(\overline{\Pi})\right] - \left[c_J(\Pi_{\text{ref}})\right]\right|\right] \quad (5)$$

When used to select *f*, the resultant AWD will bias the selection towards the DOFs that deviate the furthest from the optimal solution. As the optimization process continues, the outliers in the AWD are incrementally reduced, and the bias becomes more evenly spread to the other remaining robot DOFs. In chapter IV, the effectiveness of the proposed AWD method is tested alongside several static weighting distributions.

Algorithm III genAWD$(\overline{\Pi}, \Pi_{ref})$

input $\overline{\Pi}$ – Current Path. Π_{ref} – Reference Path.
output AWD – The adaptive weight distribution
1: $AWD = [|[c_J(\overline{\Pi})] - [c_J(\Pi_{ref})]|]$

Algorithm IV selectDOF(AWD, $nDoF$)

input AWD – Current Weight Distribution. $nDoF$ – number of DOFs to select
output *f* – DOFs to partially interpolate.
1: **if** *nDOF* **is** *a random selection* **then**
2: *n* = random integer, specified by *nDOF*
3: **else**
4: *n* = *nDOF*
5: **end if**
6: $f = \{f_1, \ldots, f_n\}$
7: **for** $i = 1$ **to** *n*
8: f_i = a random selection from AWD
9: **end**

3.2.2 *Optimal Number of Joints to Interpolate*

The original P-Sc approach carries out shortcuts on single robot DOFs. If the partial shortcut is instead carried out simultaneously on a number of DOFs, say two or three, the efficiency of the algorithm may be improved.

To test this concept, and to find an optimal number of DOFs to shortcut, the Adaptive P-Sc algorithm includes the function *selectDOF*. This function, called on line 5 of Algorithm II and shown in Algorithm IV, facilitates the selection of manipulator joints to be partially interpolated. Two variables are passed into this function; *nDOF*, which specifies how many robot DOFs are to be selected, and AWD; the current adaptive weighted distribution. *nDOF* robot DOFs are then selected from the AWD, in the same manner as in the original P-Sc algorithm, and the partial shortcut is carried out simultaneously on this group. If a random selection is used for *nDOF*, the random number of joints to select is evaluated each time the function *selectDOF* is called.

A series of tests were proposed to find an optimal value for *nDOF*, the candidates shown in Table I. These tests are carried out and summarized in chapter IV.

TABLE I
INPUTS FOR *SELECTDOF* TESTS

nDOF	DOFs to Select Per Iteration
1	One DOF
2	Two DOFs
3	Three DOFs
Rand(1/2)	Randomly select either one or two
Rand(2/3)	Randomly select either two or three
Rand(1-3)	Randomly select between one and three

3.2.3 *Pre-Test Potential Improvement*

Due to the random nature in which path segments and robot DOFs are selected by the P-Sc algorithm; there exists a probability that a particular partial shortcut will be duplicated at a later stage in the optimization process. This probability is higher in shorter paths, and was also observed to increase as the P-Sc algorithm nears completion. If a particular partial shortcut is duplicated, a significant amount of computational effort will be wasted; as collision checks will be carried out to validate a path segment that provides no benefit to the cost of the overall path. A method to avoid this duplication of partial shortcuts is to evaluate the potential benefit a particular shortcut could

possibly provide, before carrying out the computationally expensive collision checking procedures. This is done on line 14 of Algorithm II, immediately before the collision checking of the modified path $\overline{\Pi}''$ is done. Only if the modified path $\overline{\Pi}''$ offers a reduction in total cost over the original segments, is the path then checked for collision. Otherwise the algorithm immediately jumps to the next iteration, eliminating the chance that a particular partial shortcut is carried out more than once. This feature of the Adaptive P-Sc algorithm is evaluated in chapter 4, below

4. Experimental Results

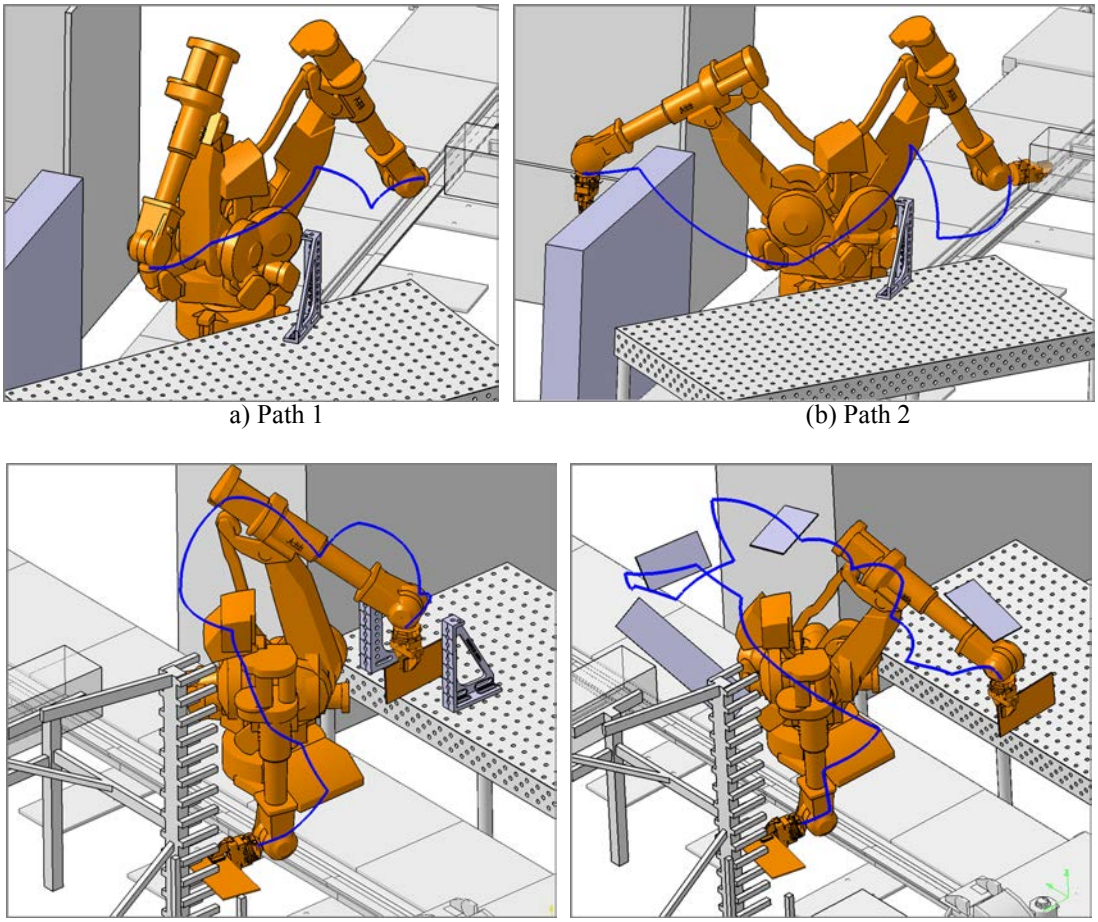
4.1 Setup

A series of tests were conducted to assess the performance of the Adaptive P-Sc algorithm. To do this four sample paths were generated from the OLP systems probabilistic motion planner. These paths feature large variances in length and complexity, and are indicative of paths typically encountered by the industrial manipulator system.

Test path 1 consists of four individual robot motions travelling from a home position to a tool rack, where a gripping end effector is loaded. The required distance for the robot to travel is short and no tool is mounted. The resultant path returned from the PRM planner only featured a small degree redundant motion. Path 2 simulated the robot moving from the tool rack, with a gripper now attached, to a handling bay on the other side of its configuration space. This path traced a similar region of space to the first, but was extended somewhat. Some additional obstacles were present, and the attached end effector increases the difficulty. Path 3 simulates the robot picking up a metal plate from a rack, and moving to position the plate on a workbench. The path length is longer than previous tests and intricate motions are required to both pick up and position the metal plate. The large size of the plate being transported also complicates the path planning problem. The resultant path is long, with many more nodes present. The path also features a lot of redundant joint motions, which are required to navigate the obstacles present in the problem. The fourth and final test path was similar to the third, however the path length was extended, and extra obstacles were scattered about the workspace to increase the difficulty of the problem. The purpose of this path is to test the performance of the adaptive P-Sc algorithm in a difficult scenario. Details of each of these test paths are given in Table II and Figure 2. To determine the effectiveness of the adaptive P-Sc algorithm, the various functions that distinguish it from the original P-Sc algorithm (presented in chapter section 3.1) were tested independently of one another. These tests consisted of optimizing the four paths until the optimal path cost, shown in Table II, was achieved. All results presented are averaged over 70 individual tests, and are given in terms of the number of calls to the collision checking algorithm. The results are presented graphically in the results section, and the raw data is tabulated in the appendix. The optimal path costs, used to terminate the optimization process, were determined in a separate series of tests by exhaustively running the P-Sc algorithm on each path until it was optimized to a level deemed to be suitable for use in a real world robotic system. The OLP system and subsequent motion planning and optimization algorithms used in these experiments were all developed and run in the MATLAB programming environment.

TABLE II
TEST PATH DATA

Path	#	C_J	C_P	Optimal C_P
1	5	[130,50,104,312,119,85]	681	330
2	8	[200,167,120,288,222,333]	1330	550
3	15	[280,264,210,712,447,608]	2521	720
4	19	[305,255,284,784,538,868]	3034	1010



a) Path 1 (b) Path 2

(c) Path 3 (d) Path 4

Figure 2. Start and goal configurations of the four test paths. TCP trace included.

4.2 Results

The first test conducted aimed to find the optimal probability distribution to bias the selection of *f*. The AWD approach, described in chapter section 3.2.1, was evaluated alongside several static probability distributions. The original distribution used in [4] was included, along with some other static distributions that were found to perform well over the four test paths.

The results of these tests are shown in Figure 3. It was found that the adaptive method performed the best in each of the four paths. On paths 1 and 2, it was found to reduce the average number of collision checks required by 15% over the best static distribution, and 32% less than the distribution specified in [4]. The AWD approach was found to perform better on the more complex paths 3 and 4; where it used 30% fewer collision checks than its closest competitor and 38% less than the distribution specified in [4]. When averaged over the four test paths, the number of collision checks used by the AWD approach lies 1.94 standard deviations below the sample average, further highlighting the general effectiveness of the approach. The raw data from these tests is included in the appendix.

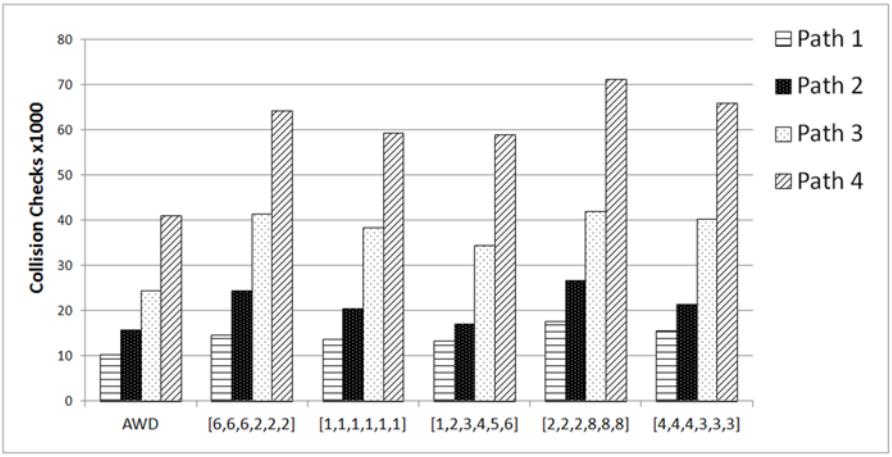


Figure 3. Results from the weight distribution tests. See Appendix A for data

The next test was designed to assess the effectiveness of the *selectDOF* algorithm introduced in section 3.1.2 , and to also find the optimal value to use for the input; *nDOF*. In these tests, the *AWD* method was used. The results, summarized in Figure 4, indicate that randomly selecting either 2 or 3 joints per iteration was the most effective. When averaged over the four test paths, selecting Rand(2/3) joints per iteration was found to reduce the number of required collision checks by 24%, 11%, 38%, 15% and 10% over the 1-DOF, 2-DOF, 3-DOF, Rand(1/2) and Rand(1-3) variants. In the simpler problems, which were easier to navigate, the 3-DOF option performed well. However on paths 3 and 4, the 3-DOF approach was outperformed considerably by the random selection variants. This was attributed to the high rate of failure when attempting to partially shortcut 3 robot DOFs simultaneously in more complex regions of space.

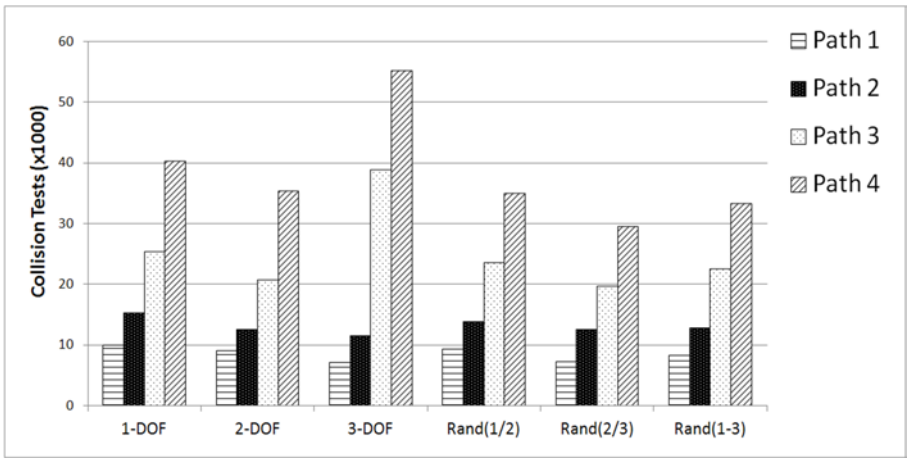


Figure 4. Results from the *nDOF* tests. See Appendix A for data

The next series of tests assessed the Adaptive P-Sc algorithms use of a check that ensures partial shortcuts are not carried out multiple times, as presented in chapter 3.2.3. As expected, this check provides an overall benefit to the speed of the P-Sc algorithm. By checking for potential improvement; the number of collision tests used to reach an optimal solution was reduced by 7%, 11%, 23% and 29% for each of the four respective problems. On average, the total number of required collision tests was reduced by 18%. Both P-Sc variants used in this phase of testing utilized AWD, and were also modified to randomly select either two or three joints per iteration. The data from these tests is presented in Figure 5.

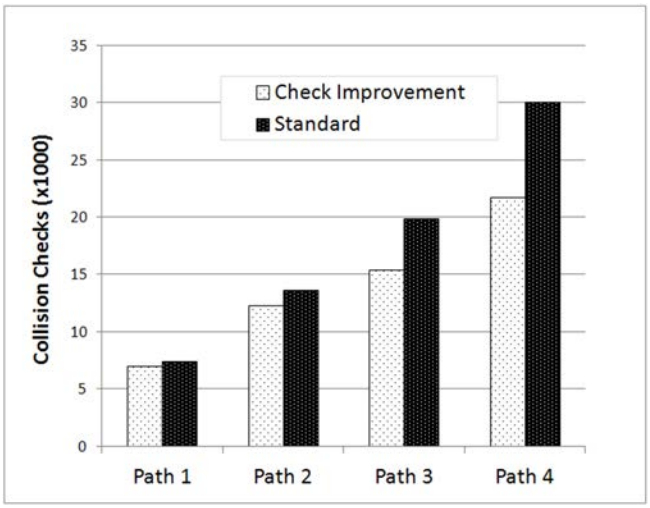


Figure 5. Results from checking potential improvement

To determine the overall effectiveness of the Adaptive P-Sc algorithm, its performance was compared to the original P-Sc variant presented in [4]. Both algorithms' configurations, and results of the subsequent tests are presented in Table III. Overall the adaptive algorithm performed significantly better than the original variant. On average, it reduced the number of collision checks required to generate an optimal path by 61%. The Adaptive algorithm showed improved performance across all tests, and is particularly suited to longer and more complex test paths. The advantage of the Adaptive P-Sc approach is shown in Figure 6, which details the optimization process of both variants during one of the optimization tests carried out on path 3. The figure shows the adaptive approach is more successful at finding valid partial shortcuts than the original variant; far fewer main loop iterations are required to reach the optimal solution.

TABLE III
COMPARISON OF ADAPTIVE AND ORIGINAL P-SC ALGORITHMS

	Original P-Sc	Adaptive	
Configuration			
Weight	[6,6,6,2,2,2]	DWD	
nDOF/iteration	1	Rand(2/3)	
Check improve?	No	Yes	
Tests			% Reduction
Path 1	14,193	7,127	49.8
Path 2	23,120	12,443	46.2
Path 3	45,961	17,061	63
Path 4	60,527	19,877	67.2
Average	**35,950**	**14,116**	**60.7**

* Results are given in number of collision checks

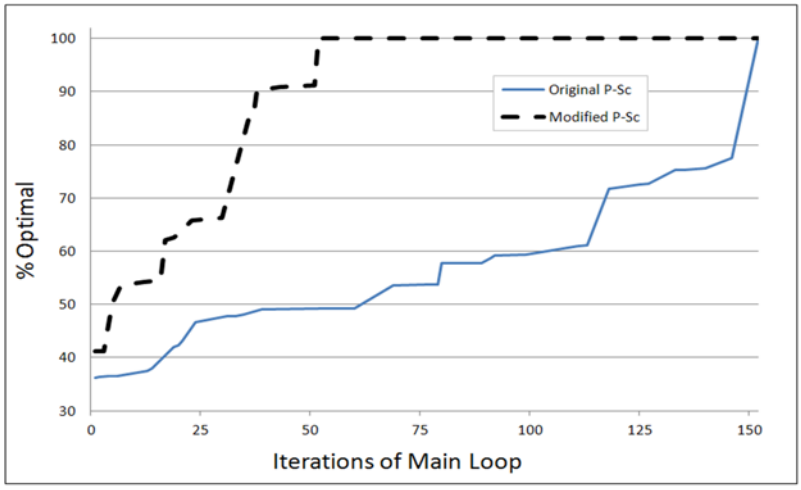
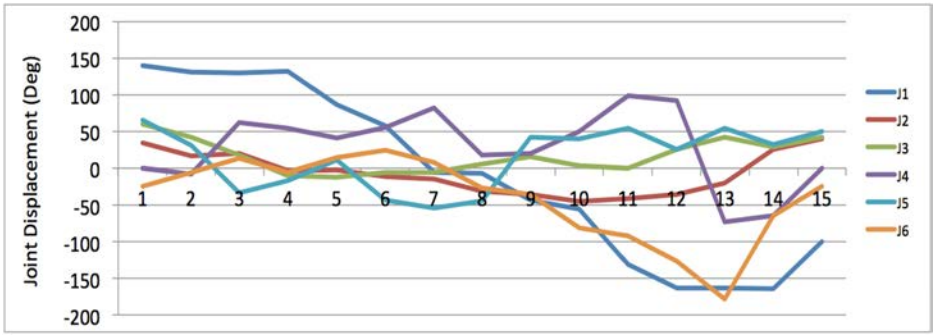


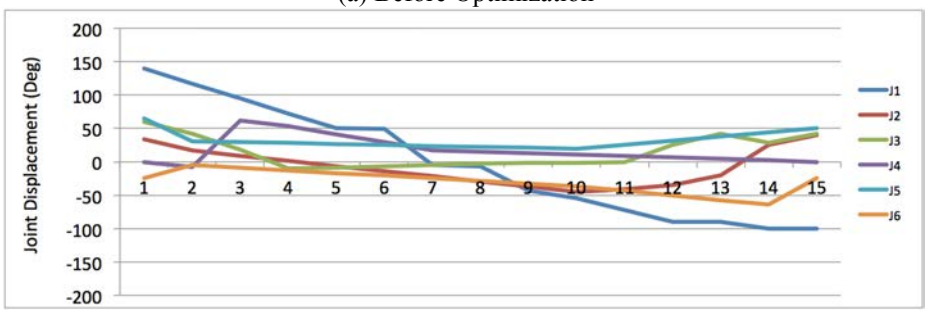
Figure 6. Comparison of Adaptive and Original P-Sc algorithms

5. Discussion

As a by-product to the way in which it reduces path length, the P-Sc algorithm also reduces redundant joint motion and overshoot. This phenomenon is represented graphically in Figure 7, which plots the trajectories of the six individual joints of the manipulator as it travels path 4, both before and after optimization is carried out. The optimization process reduced the number of individual changes in joint trajectory from 90 down to 25, indicating that redundant motion has been removed. Whilst changes in joint trajectory still exist in the optimized path, the degree in the severity of these changes has been significantly reduced; producing a smoother and more aesthetic series of manipulator motions.



(a) Before Optimization



(b) After Optimization

Figure. 7. Joint trajectories for path 3, before and after optimization

Another area of importance relates to an effective means of terminating the P-Sc algorithm. Until this stage, all tests have been terminated once the P-Sc algorithm has optimized the sample path to a pre-determined value. It is important to note that in a real world application, this optimal value will

not be known; introducing the need for an effective termination protocol for the algorithm. The simplest methods of terminating the optimizer are to end the process after a set number of iterations, or an amount of time. The drawback to these methods is that they do not take into account the complexity of the path, which can vary widely for our system. Paths that are highly complex may not be optimized enough within the set timeframe. Conversely, simple paths may be fully optimized within a few iterations and the remainder of the P-Sc iterations will be redundant. To address this issue, a specialized termination protocol was devised.

The termination protocol operates by counting consecutive failures of partially interpolated paths. The general approach is to terminate the algorithm if η successive partial shortcuts return a negative result from the collision test. To determine a suitable value of η, consecutive failures of the collision testing algorithm were recorded over 100 tests on each of the four queries. Over the 400 tests carried out, the longest run of consecutive failures encountered before an optimal solution was achieved was 9. By setting η to a conservative value of 12, we are ensuring that the P-Sc algorithm will terminate suitably. Naturally, if significant changes to the robotic system, or its surrounding environment were to occur, this value of η would need to be recalculated.

6. Conclusion

Whilst the partial shortcut algorithm is an effective method for the post process optimization of manipulator paths, it is not without its limitations. When applied to more complex problems, such as a robotic manipulator moving about high dimensional configuration spaces, a high number of collision checks are required to optimize the given path. In industrial applications where large and complex environments are present, the processing time for collision is slowed considerably, further decreasing the overall speed of the algorithm. In this paper, an Adaptive Partial Shortcut algorithm was presented. This adaptive approach utilizes an optimal reference path to dynamically select suitable robot DOFs to optimize. When tested on a variety of sample paths from a typical industrial robotic system, this adaptive P-Sc approach was found to reduce the number of collision checks required for optimization by up to a third of the original P-Sc algorithm.

7. Appendix A

TABLE A.1
RAW DATA FROM WEIGHTED DISTRIBUTION TESTS (FIG. 3.)

	Path 1	Path 2	Path 3	Path 4
AWD	10235	15732	24475	41025
[6,6,6,2,2,2]	14584	24515	41510	64322
[1,1,1,1,1,1]	13641	20592	38354	59390
[1,2,2,4,5,6]	13315	17174	34484	59063
[2,2,2,8,8,8]	17619	26762	42068	71169
[4,4,4,3,3,3]	15681	21481	40348	65930

* Results given in number of collision checks

TABLE A.2
RAW DATA FROM *NDOF* TESTS (FIG. 4.)

	Path 1	Path 2	Path 3	Path 4
1-DOF	9953	15249	25439	40238
2-DOF	9085	12559	20752	35406
3-DOF	7126	11572	38807	55157
Rand(1/2)	9393	13911	23588	35021
Rand(2/3)	7283	12563	19679	29522
Rand(1-3)	8306	12890	22572	33315

* Results given in number of collision checks

8. References

[1] Interational Federation of Robotics, Germany. 2012. “History of Industrial robots, From the first installation until today”, [online]. Available: http://www.ifr.org/fileadmin/user_upload/downloads/forms___info/History_of_Industrial_Robots_online_brochure_by_IFR_2012.pdf

[2] Z. Pan , J. Polden, N. Larkin, S. Van Duin and J. Norrish, “Recent progress on programming methods for industrial robots,” *Robotics and Computer-Integrated Manuf.*, Vol. 28, no. 2, pp. 87-94, 2012.

[3] L. E. Kavraki, P. Svestka, J. C. Latombe and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces*,” IEEE Trans Robot. Autom.*, Vol. 12, no. 4, pp. 566-580. 1996.

[4] R. Geraerts and M. H. Overmars, “Clearance based path optimization for motion planning.” In *Proc. IEEE Intl. Conf. on Robot. Autom.,* Vol. 3, pp. 2386-2392. February 2004.

[5] R. Guernane and N. Achour, “An Algorithm for Generating Safe and Execution-Optimized Paths.” *Fifth Intl. Conf. Autonomic and Autonomous Sys.,* pp. 16-21. April 2009.

[6] C. Fragkopoulos and A. Gräser, "A RRT based path planning algorithm for Rehabilitation robots." *41st Intl. Symp. & 6th German Conf. on Robotics (ROBOTIK)*, 2010.

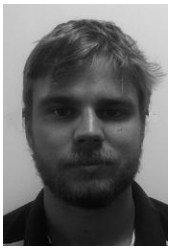
[7] S. Sekhavat, P. Svestka, J. P. Laumond and M. H. Overmars, “Multilevel path planning for nonholonomic robots using semiholonomic subsystems.” *The intl. j. rob research*, Vol 17, no. 8, pp 840-857, 1998

[8] J. Kim, R. A. Pearce and N. M. Amato, “Extracting optimal paths from roadmaps for motion planning.” In *Proc. IEEE Int. Conf. Robot. Autom* Vol. 2, pp. 2424-2429. September 2003.

[9] D. Nieuwenhuisen and M. H. Overmars, “Useful cycles in probabilistic roadmap graphs,” *Proc. IEEE Int. Conf. Robot. Autom* Vol. 1, pp. 446-452, April 2004

[10] K. Klasing, D. Wollherr and M. Buss, “Cell-based probabilistic roadmaps (CPRM) for efficient path planning in large environments,” In Proc. Int. Conf. on Advanced Robotics, August 2007.

[11] R. Guernane and N. Achour, "Generating optimized paths for motion planning," *Rob. and Autonomous Sys.*, Vol 59, no. 10, pp 789-800, 2011
[12] N. Ratliff, M. Zucker, J. A. Bagnell and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning." *Proc. IEEE Int. Conf. Robot. Autom* , pp. 489-494, May 2009.
[13] C. S. Lin, P. R. Chang, and J. Luh, "Formulation and optimization of cubic polynomial joint trajectories for industrial robots." *IEEE Trans. on Automatic Control*, Vol 28, no.12 , pp. 1066-1074, 1983
[14] P. Isto, "Constructing probabilistic roadmaps with powerful local planning and path optimization," In *IEEE/RSJ Intl. Conf. on Intel. Rob. and Systems.* Vol. 3, pp. 2323-2328, 2002.
[15] M. Kallmann, A. Aubel, T. Abaci, and D. Thalmann, "Planning Collision-Free Reaching Motions for Interactive Object Manipulation and Grasping," In *Computer Graphics Forum*," Vol. 22, No. 3, pp. 313-322, November 2003.
[16] G. Sánchez, and J. C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," *Robotics Research*, pp 403-417, 2003
[17] R. Guernane and M. Belhocine, "A smoothing strategy for PRM paths application to six-axes MOTOMAN SV3X manipulator," *In IEEE/RSJ Intl. Conf. Intel. Rob. Sys.,* pp 4155-4160, August 2005.
[18] D. Hsu, J. C. Latombe, and S. Sorkin, "Placing a robot manipulator amid obstacles for optimized execution," In *Proc. IEEE Intl. Symp. Assembly and Task Planning*, pp. 280-285. 1999.
[19] M. Stilman, "Global manipulation planning in robot joint space with task constraints," *IEEE Trans. Rob.*, Vol. *26* no. 3, 576-584. 2010
[20] C. Holleman, and L. E Kavraki, "A framework for using the workspace medial axis in PRM planners." In *Proc. Intl Conf. on Robotics and Automation.,* Vol. 2, pp. 1408-1413. 2000.



Joseph Polden received a B.Eng (Mech), from the University of Wollongong, NSW, Australia in 2008. He is currently working toward a Ph.D. degree in mechatronics, also at the University of Wollongong. His current research interests relate to the development of Automated Offline Programming systems for industrial applications.

Dr. Zengxi Pan received the B.S. and M.S. degree in mechanical engineering from Tsinghua University, Beijing, China, in 1998 and 2000, and the Ph.D. degree in Robotics from Stevens Institute of Technology Hoboken, NJ, USA in 2005. He is currently a Senior Research Fellow in the Faculty of Engineering at University of Wollongong, Wollongong, NSW, Australia.