Robot programming using augmented reality

Chong, Jonathan Wun Shiung

2006

Chong, J. W. S. (2006). Robot programming using augmented reality. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/60515>

<https://doi.org/10.32657/10356/60515>

ROBOT PROGRAMMING USING AUGMENTED REALITY

By

Jonathan Chong Wun Shiung

S.M. in Innovation in Manufacturing Systems and Technology (IMST)
Singapore-MIT Alliance, 2003
B. Eng. in Mechanical Engineering
Universiti Teknologi Malaysia (UTM), 2002

SUBMITTED TO SMA OFFICE IN PARTIAL
FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY
IN INNOVATION IN MANUFACTURING SYSTEMS AND TECHNOLOGY
(IMST)
AT THE
SINGAPORE-MIT ALLIANCE

2006

Signature of Author: _____

Jonathan Chong Wun Shiung
IMST Programme
21 May 2007

Certified by: _____

Professor Andrew Nee Yeh Ching
SMA Fellow, NUS
Thesis Advisor

Certified by: _____

Professor Kamal Youcef-Toumi
SMA Fellow, MIT
Thesis Advisor

Accepted by: _____

Professor Yue Chee Yoon
Programme Co-Chair
IMST Programme

Accepted by: _____

Professor David E Hardt
Programme Co-Chair
IMST Programme

To
my Lord and Saviour who is omnipresent,
my father and mother back home in Kuching, and my brother here in Singapore,
and my beloved Belinda who I pray will be here soon.

ACKNOWLEDGEMENTS

Above all, I thank God for the strength, perseverance and inspiration I needed to complete this research over the past three years. It has been by far, the period of my life that has stretched my faith to its limits, and taught me a great deal about myself.

I am indebted to my supervisors, Prof. Andrew Nee Yeh Ching from NUS and Prof. Kamal Youcef-Toumi from MIT for giving me the opportunity to pursue this degree. Prof. Nee has truly been like a 'father' to me during this period of time. He encouraged me whenever I felt like quitting and allowed me to go through challenges that were necessary to build my character, even when I did not realise it! I also have to mention that Prof. Nee is a true gentleman, evident in how he deals with the pressure of research and holding important places in office. I would also like to thank Prof. Kamal for providing me with the 'MIT experience', his insightful observations and critique have profoundly changed the way I solve problems. During my stay at MIT, I enjoyed the conversations and discussions we had about research and life in general. Prof. Kamal is also the only professor I know personally who uses Skype and MSN Messenger to converse with his colleagues and students, truly impressive!

I would like to thank all those in the AR research group in NUS. Firstly, I am indebted to Prof. Ong Soh Khim of NUS for constantly challenging me to conduct better research. Her advice and constructive criticism are a major reason why I could complete this work on time. I would also like to highlight her amazing attention to detail which is something that I need to practise more in my own work. Special thanks also goes out to my lab mates, Poh, Pang Van, Dr. Yuan, Zhang lie, Shen Van, and

Louis for making this experience memorable. Poh and Pang Van especially have been there through the ups and downs, supporting me in my research and personal life.

I would also like to thank Prof. Marcelo H. Ang Jr. from the mechanical engineering Department at NUS, Prof. Lee Wee Sun and Prof. David Hsu from the SMA-CS programme, and Dr. Mervyn Fathianathan who have been so gracious with their time when I sought advice. Special thanks to Greg Belote and Duncan Ma from MIT for taking interest and spending all those hours discussing my work, and Simon Zhao and Steven Halim from NUS for their kind help with algorithms and programming.

I have made many friends who have enriched my life and supported me in a non-academic way. To Mikhail Shapiro of MIT, the genius who plays the guitar in the most logical way, thanks for being such a wonderful friend. I miss those nights hanging out just talking about life. Thank you also to my church friends who have provided me a lifeline when I had no place to run to, and for keeping me in your prayers. To Ivan, Danny, Kevin, Michelle, CK, Jeff, and Merv, thank you for the music and friendships.

My heart also goes out to my beloved family for their understanding and love. I hope that I have made you proud, thank you for believing in me and teaching me about integrity and excellence. To Belinda, thank you for your unselfish love. Being away is painful and tough, but you came into my life at just the right time. Without you, this work would never have been completed. You are my inspiration and source of strength.

Last but not least, I would like to thank SMA for funding this research for the past three years. To Mei Ling, Hasinati, Thatchi, Vino and Michelle, thanks for everything!

TABLE OF CONTENTS

Title Page	
Acknowledgements	ii
Table of Contents	v
List of Figures	x
List of Tables	xiv
List of Algorithms	xv
List of Abbreviations	xvi
List of Symbols	xvii
Abstract	xxi
 CHAPTER 1: INTRODUCTION	 1
1.1 Chapter Introduction	
1.2 Current and Future Trends in Robotics	
1.3 Human-Robot Interaction	4
1.4 Motivation for the Work	4
1.5 Objectives of the Work	5
1.6 Organisation of Thesis	6
 CHAPTER 2: ROBOT PROGRAMMING AND AUGMENTED REALITY	 9
2.1 Chapter Introduction	9
2.2 Robot Programming	9
2.2.1 Existing Robot Programming Methods	9
2.2.2 Programming by Demonstration	14

2.3	Augmented Reality	19
2.3.1	Technical Issues in AR	21
2.3.1.1	Optical or Video Head-Mounted Display	21
2.3.1.2	Tracking and Sensing Methods	22
2.3.1.3	Registration	24
2.3.1.4	Hardware	25
2.3.1.5	Cognitive Issues	26
2.3.2	Applications of AR	26
2.4	Robot Programming using AR	29
2.4.1	Benefits and Potential of AR for Robot Programming	32
2.4.2	Targeted Applications for RPAR	33
2.4.3	Registration of Virtual Robot	35
2.4.4	Programming Large Robots using RPAR	36
2.5	Chapter Summary	37
CHAPTER 3: RPAR SYSTEM DESCRIPTION		39
3.1	Chapter Introduction	39
3.2	ARToolkit Marker-based Tracking Method	39
3.3	Relationships between Main Coordinate Systems	40
3.4	Robot Kinematics Model	42
3.5	RPAR System Architecture	43
3.6	Chapter Summary	46
CHAPTER 4: RPAR METHODOLOGY FOR CLASS I TASKS		49
4.1	Chapter Introduction	49

4.2	Overview of Path Planning Methods	49
4.2.1	Classical Path Planners	49
4.2.2	Sampling-based Algorithms	51
4.3	Proposed RPAR-I Methodology for Class I Tasks	52
4.4	Human Input from Demonstrations	55
4.4.1	Collision-Free Volume for RPAR-I	55
4.4.2	CFV Check	58
4.4.3	Selection of Start and Goal Configurations	60
4.5	Beam Search Strategy for Generating Collision-Free Paths	61
4.5.1	Path Representation	61
4.5.2	Path Cost	63
4.5.3	Heuristic Beam Search	65
4.6	Results and Discussions for Class I Tasks	74
4.6.1	Scenario 1: The neighbourhoods for the greedy path do not hit the boundary of the CFV	74
4.6.2	Scenario 2: The neighbourhoods for the greedy path hit the boundary of the CFV	76
4.6.3	Scenario 3: The search is stalled	80
4.6.4	Other Possible Scenarios	80
4.7	Chapter Summary	83
 CHAPTER 5: RPAR METHODOLOGY FOR CLASS II TASKS		85
5.1	Chapter Introduction	85
5.2	Proposed RPAR-II Methodology for Class II Tasks	85
5.3	Output Curve Representation and Parameterization	88

5.3.1	Parametric Curve Representation	88
5.3.2	Data Collection and Parameterization	88
5.3.3	Discussions on Curve Representation and Parameterization	93
5.3.3.1	Curve Representations	93
5.3.3.2	Parameterization	94
5.4	Curve Learning and Generation	95
5.4.1	Proposed Bayesian Neural Network and Reparameterization Algorithm	95
5.4.2	Reparameterization and Error Definitions	98
5.4.3	Discussions on Curve Learning and Generation	99
5.4.3.1	Learning, Fitting and Modelling of Curves	99
5.4.3.2	Neural Networks	101
5.4.3.3	Bayesian Framework	104
5.4.3.4	Reparameterization	107
5.5	Collision-Free Volume for RPAR-II	108
5.6	End-Effector Orientation Planning	108
5.6.1	Simple Case using Linear Interpolation	108
5.6.2	Complex Case using Space Curve	111
5.7	Results and Discussions for Class II Tasks	116
5.7.1	Curve Learning Results and Discussions	116
5.7.1.1	Limitations of ARToolkit	117
5.7.1.2	Experimental Procedure	118
5.7.1.3	Curve Learning Case Studies	119
5.7.1.4	Effects of the PLP Algorithm, Reparameterization and ARToolkit Limitations	122

5.7.1.5	Effects of the Number of Demonstrations and Different Curve Learning Methods	126
5.7.2	End-Effector Orientation Planning Case Studies	139
5.7.2.1	Simple Case Study	139
5.7.2.2	Complex Case Studies	141
5.8	Chapter Summary	148
CHAPTER 6: CONCLUSIONS AND RECOMMENDATIONS		150
REFERENCES		155
APPENDIX A: COMPARISON BETWEEN TRACKING SYSTEMS		165
APPENDIX B : BEAM SEARCH PSEUDOCODE		166
APPENDIX C : BAYESIAN NEURAL NETWORKS		168
APPENDIX D : CALCULATING THE PROJECTION OF THE TRACKED PROBE TIP ONTO THE NORMAL PLANE		172
APPENDIX E: CONTENTS OF ATTACHED CD-ROM		173

LIST OF FIGURES

Figure 1.1: Drivers for the advancement of robotics	2
Figure 2.1: Traditional robot programming methods [7]: (a) lead-through programming, (b) walk-through programming, and (c) offline programming	10
Figure 2.2: VR system by Natonek <i>et al.</i> [9]	12
Figure 2.3: VR system by Aleotti <i>et al.</i> [10]	12
Figure 2.4: VR for robot programming [11]: (a) actual robotic work cell, and (b) VR environment in a desktop-PC	13
Figure 2.5: Principal architecture of a robot PbD system [19]	16
Figure 2.6: Summary of robot PbD research	17
Figure 2.7: PbD system using a teaching gripper [21]	19
Figure 2.8: Real-Virtuality (RV) continuum (adapted from [32])	20
Figure 2.9: A comparison of Human-Computer Interaction (HCI) styles [34]: (a) human-computer interaction isolated from human-real world interaction, (b) user immersed in computer world and isolated from real world, (c) user interacts with real world and computers placed in the real world, and (d) user uses a computer to aid in his/her interaction with the real world (AR)	20
Figure 2.10: Technical issues to be considered when developing AR systems	21
Figure 2.11: (a) Optical HMO, and (b) video HMO [33]	22
Figure 2.12: Personal Interaction Panel (PIP) input device using a tracked pointer [37]	23
Figure 2.13: Applications of AR: (a) maintenance [48], (b) mock tumour biopsy [33], (c) wire bundle assembly [51], (d)-(e) factory layout planning [52], and (f) human workspace in manual assembly station [52]	27
Figure 2.14: Telerobotic control using AR [53]	28
Figure 2.15: (a) Planning of a robot task and visual feedback, and (b) execution of the task [53]	28
Figure 2.16: Flow of information in an immersive AR environment	29
Figure 2.17: Virtual robots in real environments: (a) miniature generic-purpose robot in a miniature kitchen, (b) miniature airplane washing robot, and (c) full-scale robot model	31
Figure 2.18: Class I tasks: a) pick-and-place for disabled [56], b) pick-and-place for industrial applications [57], c) spot welding [57], and d) bridge inspection [58]	34
Figure 2.19: Class II tasks: a) airplane washing [59], b) arc welding [60], and c) laser cutting [61]	35
Figure 2.20: Registering the virtual robot in the real environment	36
Figure 2.21: Scaling and rescaling for programming large robots using RPAR, for the example of airplane washing robots	37
Figure 3.1: Relationships between camera and markers	41
Figure 3.2: (a) Handheld probe for general interaction, and (b) handheld probes with different end-effectors	41
Figure 3.3: Six dof robot model and an alternative end-effector (bottom right)	42
Figure 3.4: RPAR system architecture (excluding the physical robot controller)	45
Figure 3.5: HMO used for visualization	45

Figure 3.6:	(a) PointGrey FireFly camera [65] and (b) i-glasses [66]	46
Figure 3.7:	Tools developed for the path planner in this thesis	46
Figure 4.1:	Roadmap examples [13]: (a) visibility graph, and (b) Voronoi diagram	50
Figure 4.2:	Cell decomposition [13]	50
Figure 4.3:	An energy landscape for potential functions [13, 67]	51
Figure 4.4:	A guided EST focuses the search and uses a smaller search space (edges) [71]: (a) a standard EST, and (b) a guided EST for the same start and goal as (a)	52
Figure 4.5:	Proposed RPAR-I methodology	53
Figure 4.6:	(a) Sphere attached to a probe with marker, and (b) generating a CFV over a rectangular block	57
Figure 4.7:	Interpolating between spheres does not guarantee that an object within the CFV is collision-free	58
Figure 4.8:	Bounding cylinder and its characteristic points within the CFV	59
Figure 4.9:	Forces due to gravitational moments for (a) rotary joints, and (b) linear joints	64
Figure 4.10:	Full search space with the start configuration as the root node	66
Figure 4.11:	Beam search concept where a heuristic guides the search to determine which portions of a current slice should be kept [74]	66
Figure 4.12:	Filter (3) prunes revisited nodes/states and their corresponding paths ($\rightarrow 4 \rightarrow 5 \rightarrow 1$), and avoids cycles within a path ($\rightarrow 1 \rightarrow 2 \rightarrow 1$)	67
Figure 4.13:	Filter (4) deletes duplicate nodes (Node 5 for path $\rightarrow 3 \rightarrow 4 \rightarrow 5$) and selects only the lowest cost path to that particular node	68
Figure 4.14:	Example of the beam search approach in a 2D robot C-space	73
Figure 4.15:	Scenario 1: (a) a well-defined compact CFV, and (b) the final path (search steps = 86, <i>total-path-cost</i> = 106.0423 J, and final <i>dist-to-goal</i> = 0.0075 rad)	76
Figure 4.16:	Solution path for example in Scenario 1 with resolution for all joints = 0.5°	76
Figure 4.17:	Jagged movements along the CFV boundary may exist if the CFV is not compact enough, <i>i.e.</i> , boundaries are not smooth	77
Figure 4.18:	Scenario 2: (a) Example I: when the end-effector hits the boundary of the CFV, (b) the solution path showing the circled portion along the boundary of the CFV, and (c)-(d) Example II	78
Figure 4.19:	Solution path for Example I in Scenario 2 with resolution for all joints = 0.5° ($k = 10$, search steps = 134, <i>total-path-cost</i> = 59.3368 J, and final <i>dist-to-goal</i> = 0.0074 rad)	79
Figure 4.20:	Effects of increasing k on the path costs for both examples in Scenario 2	79
Figure 4.21:	Scenario 3: (a) CFV with defect, and (b) path found for $k=20$ (search steps = 139, <i>total-path-cost</i> = 135.1606 J, final <i>dist-to-goal</i> = 0.0107 rad, and resolution for all joints = 0.5°)	80
Figure 4.22:	Planning curvy paths around pins using multiple paths (three sequential paths), for three different CFVs and sets of start-goal configurations	81
Figure 4.23:	General pick-and-place task: (a) generating a CFV, (b) selecting start configuration, (c) selecting intermediate goal configuration, (d) selecting final goal configuration, and (e) final paths found	82

Figure 4.24: Full-scale general pick-and-place task: (a) generating a CFV, (b) selecting start configuration, (c) selecting goal configuration, and (d) final path found	83
Figure 5.1: Constructing a virtual car roof for aesthetic evaluation [76]	86
Figure 5.2: Curve learning problem and proposed approach	86
Figure 5.3: Proposed RPAR-II methodology	87
Figure 5.4: Parametric curve	88
Figure 5.5: Data collection modes: (a) Point-by-Point (PbP) mode, and (b) tracing mode	89
Figure 5.6: Definitions for PLP algorithm	90
Figure 5.7: 20 example of parameterization using PLP algorithm	92
Figure 5.8: Different representations of a planar circle [77]	94
Figure 5.9: Proposed Bayesian neural network structure with one hidden layer	97
Figure 5.10: (a) Single perceptron, and (b) multilayered feedforward neural network	102
Figure 5.11: Evidence of Occam's razor [82-85]	106
Figure 5.12: Simple case using linear interpolation and unconstrained end-effector orientation selection	110
Figure 5.13: Local surface planes defined by tracing rough curves on both sides of the output curve	III
Figure 5.14: Defining the curve coordinate system at its start (and the corresponding end-effector direction) using the user-defined surface point	113
Figure 5.15: Coordinate systems involved for calculating end-effector transformations	113
Figure 5.16: Case Study 1- miniature planar 'AR' curve: (a) desired curve and robot base, (b) real drawn curve, (c) piecewise linear approximation of desired curve obtained from Step 3, and (d) desired curve using piecewise linear approximation from Step 1(iii)	120
Figure 5.17: Case Study 2- miniature non-planar curve on mouse: (a) desired curve and robot base, (b) real drawn curve, (c) piecewise linear approximation of desired curve obtained from Step 3, and (d) desired curve using piecewise linear approximation from Step 1(iii)	121
Figure 5.18: Case Study 3- full-scale non-planar curve on bent sheet metal: (a) desired curve and robot base, (b) real drawn curve, (c) piecewise linear approximation of desired curve obtained from Step 3, and (d) desired curve using piecewise linear approximation from Step 1(iii)	121
Figure 5.19: Output curve and final parameterized data (ten demonstrations) for Case Study 1: (a) Method A, (b) Method B, (c) Method C, and (d) Method 0	123- 124
Figure 5.20: Output curve and final parameterized data (ten demonstrations) for Case Study 2: (a) Method A, (b) Method B, (c) Method C, and (d) Method D	124
Figure 5.21: Output curve and final parameterized data (ten demonstrations) for Case Study 3: (a) Method A, (b) Method B, (c) Method C, and (d) Method 0	125
Figure 5.22: Example of 3D data points parameterized using the PLP algorithm according to a piecewise linear approximation	125
Figure 5.23: Case Study 1: (a) average error, (b) maximum error, and (c) variance of error	128

Figure 5.24: Case Study 2: (a) average error, (b) maximum error, and (c) variance of error	128
Figure 5.25: Case Study 3: (a) average error, (b) maximum error, and (c) variance of error	129
Figure 5.26: Output curves overlaid on drawn desired curve for Case Study 1: (a) one demonstration, (b) five demonstrations, and (c) ten demonstrations	130-132
Figure 5.27: Output curves overlaid on drawn desired curve for Case Study 2: (a) one demonstration, (b) five demonstrations, and (c) ten demonstrations	133-135
Figure 5.28: Output curves overlaid on drawn desired curve for Case Study 3: (a) one demonstration, (b) five demonstrations, and (c) ten demonstrations	136-138
Figure 5.29: Planning the end-effector orientation and a successful collision-free path for the simple case study using linear interpolation	140
Figure 5.30: Angle profiles for the collision-free path for the simple case study	141
Figure 5.31: Planning the end-effector orientation and a successful collision-free geometric path for Case Study 1	142-143
Figure 5.32: Angle profiles for the collision-free path for Case Study 1	143
Figure 5.33: Planning the end-effector orientation and a successful collision-free geometric path for Case Study 2	144-145
Figure 5.34: Angle profiles for the collision-free path for Case Study 2	145
Figure 5.35: Planning the end-effector orientation and a successful collision-free geometric path for Case Study 3	146-147
Figure 5.36: Angle profiles for the collision-free path for Case Study 3	147

LIST OF TABLES

Table 2.1: Scenarios for robot and environment entities	30
Table 2.2: Comparison between RPAR and traditional robot programming approaches	33
Table 5.1: Comparison between modelling, fitting and learning	100
Table 5.2: Four methods used for curve learning	117
Table A.1: Comparison between ARToolkit and optical tracking systems	165

LIST OF ALGORITHMS

Algorithm 4.1:	<i>CFV Check</i> algorithm	60
Algorithm 4.2:	Proposed beam search algorithm	66
Algorithm 5.1:	Proposed PLP algorithm	90-91
Algorithm 5.2:	Proposed Bayesian neural network and reparameterization algorithm	97
Algorithm 5.3:	Algorithm for calculating end-effector transformations along the output curve	114-115
Algorithm B.1:	Detailed beam search pseudocode	166-167

LIST OF ABBREVIATIONS

HRI	Human-Robot Interaction
AI	Artificial Intelligence
HCI	Human-Computer Interaction
UN	United Nations
JETRO	Japan External Trade Organisation
VR	Virtual Reality
AR	Augmented Reality
SCARA	Selective Compliant Assembly Robot Arm
PbD	Programming by Demonstration
RPAR	Robot Programming using Augmented Reality
OSHA	Occupational Safety & Health Administration
VRPL	Virtual Reality and Rapid Prototyping Laboratory
CAVE	Cave Automatic Virtual Environment
C-Space	Configuration Space
RV	Real-Virtuality
MR	Mixed Reality
HMD	Head-Mounted Display
PIP	Personal Interaction Panel
VE	Virtual Environment
ARVIKA	Augmented Reality for Development, Production and Servicing
AREAS	Augmented Reality System for Evaluating Assembly Sequences
ABB	Asea Brown Boveri
CAD	Computer-Aided Design
SME	Small and Medium Enterprise
dof	degrees-of-freedom
DH	Denavit-Hartenberg
ISS	International Space Station
CFV	Collision-Free Volume
IK	Inverse Kinematics
FK	Forward Kinematics
PRM	Probabilistic RoadMap
EST	Expansive-Spaces Tree
fps	frames per second
PLP	Piecewise Linear Parameterization
PbP	Point-by-Point
LM	Levenberg-Marquardt
SSE	Sum of Squared Errors
SSW	Sum of Squared Weights
CMM	Coordinate Measurement Machines
ODF	Orthogonal Distance Fitting
EE	End-Effector

LIST OF SYMBOLS

CHAPTER 3

x, y	2D image coordinates
X, Y, Z	3D Euclidean space coordinates
ρ	Arbitrary factor
A	Matrix of camera intrinsic parameters
${}^{(1)}_{(2)}M$	Transformation matrix relating coordinate system (2) with respect to coordinate system (1)
${}^{(1)}_{(2)}R$	Rotation matrix relating coordinate system (2) with respect to coordinate system (1)
${}^{(1)}_{(2)}T$	Translation matrix relating coordinate system (2) with respect to coordinate system (1)
$\mathbf{q}$	Vector of generalised coordinates for a robot
$\boldsymbol{\theta}$	Vector of angular joints for a robot

CHAPTER 4

S_i	Virtual spheres
c_i	Coordinates for the centre of S_i
r_0	Radius of virtual spheres
$\mathcal{R}^3$	3D Euclidean space
r	Radius of physical sphere
$\Delta \mathbf{r}_{robot_error}$	Maximum positional error of the physical robot within its workspace
$\Delta \mathbf{r}_{tracking}$	Maximum error of the tracking system
$\Delta \mathbf{r}_{modelling}$	Virtual robot modelling error
p_j	Bounding cylinder characteristic points
θ_k, z_l	Bounding cylinder parameters
r_{cyl}	Radius of bounding cylinder
h_{cyl}	Height of bounding cylinder
$L_{j,i}^{p_i \rightarrow c_i}$	Distance between characteristic point P_i and virtual sphere centre c_i
ε_j	Perturbation magnitude for <i>joint</i> j
$\Delta q_{j,t}$	Perturbation value for <i>joint</i> j at search step t
$\Delta \mathbf{q}_{c,t}$	Vector of perturbation values for combination c at search step t
$\mathbf{q}_t$	Robot configuration at search step t
$\mathbf{u}$	Vector of joint generalized forces
$u_{j,t}$	Generalized force at <i>joint</i> j at search step t
$\tau_{j,t}$	Torque at <i>joint</i> j at search step t
$F_{j,t}$	Linear force at <i>joint</i> j at search step t

$F_{N,j,t}$	Normal force at joint j at search step t
$F_{T,j,t}$	Tangential force at joint j at search step t
r_j	Rotation radius for joint j (distance between rotation centre and tangential force)
μ_j	Bearing coefficient of friction at joint j
$\Delta\theta_{j,t}$	Angular perturbation at joint j at search step t
$\Delta d_{j,t}$	Linear perturbation at joint j at search step t
$\bar{u}_{j,t}$	Average generalized force at joint j at search step t
b	Branching factor for beam search
k	Beam size/width
h	Height of beam search tree
S	Start node for beam search
G	Goal node for beam search
V	Total number of vertices/nodes possible in the free C-space of the robot
W	Number of nodes removed from neighbourhood region after filtering

CHAPTERS

$\mathbf{a}$	Vector of model parameters
$\mathbf{b}$	Vector of model parameters
t	Location parameter for parametric curve
$\hat{u}_j$	Unit vector for piecewise linear line j
L_j	Length for piecewise linear line j
p_j	Vertices for piecewise linear approximation of curve j
d_i	3D data points
$v_{i,j}$	Vector between start point P_j of piecewise linear line j and data point d_i
a_j	Projection of $v_{i,j}$ in the direction of $\hat{u}_j$
$b_{i,j}$	Point in the direction of $\hat{u}_j$ where d_i is closest to
$L_j^{(1) \rightarrow (2)}$	Distance between points (1) and (2)
t_i	Parameter for d_i
$E(\mathbf{W})$	Neural network objective function with $\mathbf{W}$ being the vector of weights
w_i	Network weights
F	Space curve function
G	Space curve function
β	Regularization parameter for sum of squared errors in neural network objective function
α	Regularization parameter for sum of squared weights in neural network objective function
E_D	Sum of squared errors
E_w	Sum of squared weights
$\mathbf{H}$	Hessian matrix
res	Resolution parameter for parametric curve

l_i	Shortest Euclidean distance between a measurement point i and a curve that consists of a set of points
$\mathbf{X}_i$	3D points $[x, y, z]$
$\mathbf{X}_j^F$	Sets of 3D points for function F
$\mathbf{X}_j^G$	Sets of 3D points for function G
e_i	Errors between output and desired curve
$\bar{e}$	Average error
$e_{\max}$	Maximum error
σ^2	Variance of error
y_o	Output for logistics function
$\mathbf{w}$	Vector of weights for logistics function
$\mathbf{x}$	Input vector to logistics function
b	Bias for logistics function
D_i	Desired output for neural network
y_i^L	Output for unit i for layer L (output layer)
u_i^l	Activity for unit i for layer l
w_{ij}^l	Weight i for layer l connected to neuron j from layer $l-1$
$h_j^{(l-1)}$	Output from unit j from layer $l-1$
b_i^l	Bias i for layer l
s_i^l	Sensitivity i for layer l
λ	Fixed learning rate for neural network
$\mathbf{W}_{MP}$	Most likely weights at the maximum of posterior
$\Delta\mathbf{W}$	Difference between $\mathbf{W}$ and $\mathbf{W}_{MP}$
D	Data set for Bayesian neural network
$\sigma_{w D}$	Standard deviation of posterior distribution after data is presented
σ_w	Standard deviation of posterior distribution before data is presented
M	Model under consideration in the Bayesian framework
$\bar{L}$	Straight line vector between start and end points for linear interpolation
q	Control parameter for determining resolution of linear interpolation
$\mathbf{I}$	Identity matrix
γ	Angle between end-effector vector/direction and surface plane
$\hat{X}_i, \hat{Y}_i, \hat{Z}_i$	Unit vectors for coordinate systems' axes
ϕ_i	Rotation angle about $\hat{Z}_{i+1}$ axis
α_i	Rotation angle about $\hat{X}_{i+1}$ axis

APPENDICES

$N_{i,p}$	Node i for path p in beam search
$N_{H,p}$	Head node for path p in beam search
$Z_D(\beta)$	Normalization constant for $P(D W, \{3, M\})$
$Z_w(\alpha)$	Normalization constant for $p(w a, M)$
$Z_F(\beta, \alpha)$	Normalization constant for $p(w D, \{3, a, M\})$

β_{MP}	Most probable β regularization parameter
α_{MP}	Most probable α regularization parameter
N	Total number of available parameters
γ'	Number of effective parameters used by the network
$\mathbf{H}_{MP}$	Most probable Hessian matrix of objective function
λ	Learning parameter for neural network
e	Residual
$\mathbf{J}$	Jacobian matrix containing sensitivities of residuals to network weights
$[a, b, c]$	Components of unit vector $\hat{Y}_1$ from Chapter 5
u	Constant used for calculating projected point on normal surface
$[x_0, y_0, z_0]^T$	Coordinates of the start point of the curve
$[x_p, y_p, z_p]^T$	Coordinates of projected point on normal surface
$[x_{tracked}, y_{tracked}, z_{tracked}]^T$	Coordinates of tracked probe tip

ROBOT PROGRAMMING USING AUGMENTED REALITY

By

Jonathan Chong Wun Shiung

Submitted to the Singapore-MIT Alliance (SMA) Office
In Partial Fulfilment of the Requirements for the
Degree of Doctor of Philosophy in Innovation in Manufacturing Systems and
Technology (IMST)

ABSTRACT

Current robot programming approaches lack the intuition required for quick and simple applications. As new robotic applications are being identified, there is a greater need to be able to programme robots safely and quickly. Recently, an approach known as Programming by Demonstration (PbD) has emerged primarily addressing intuitiveness in Human-Robot Interaction (HRI). This approach is complemented by another emerging technology known as Augmented Reality (AR) which is the environment where computer-generated 3D objects are blended (registered) onto a real world scene, to enhance the interaction of a user with the real world. The work in this thesis marries the two concepts above by proposing a new approach for immersive robot programming known as Robot Programming using AR (RPAR), where the user directly moves a virtual robot amongst real objects in an *unknown* environment. The first part of this work is the development of a new RPAR methodology aimed at applications such as pick-and-place tasks, where there are a number of possible path solutions for a given start and goal configuration pair. The ability of the user to make sense of an unknown environment is utilized by enabling him/her to intuitively define subsets of the free space that are relevant to his/her goals. Collision-free paths for user-defined start and goal configurations are then generated using a beam search strategy. The second part of this work is the development of a new RPAR methodology aimed at applications such as arc welding and laser cutting, where the end-effector of the robot is constrained to follow a user-defined 3D path at a certain orientation, consistently with respect to that path. The first step requires the user to perform a number of demonstrations to obtain data points of the path/curve to be followed. These data points are processed using a number of algorithms to produce an output curve that best represents the demonstrated curve, *i.e.*, learning the human intention. The Piecewise Linear Parameterization (PLP) algorithm and a curve learning method based on Bayesian neural networks and reparameterization are proposed. After the output curve is generated, the user is required to interactively define the free space along the curve. The orientation of the end-effector can then be planned intuitively using visual feedback from AR, and a final collision-free path can be obtained. The two RPAR methodologies developed in this research have been applied to a number of case studies and the various related issues have been discussed.

Keywords:

Robot programming, Augmented Reality, methodology, Collision-Free Volume, beam search, curve learning, parameterization, reparameterization, Bayesian neural networks, end-effector orientation planning

Thesis Advisors:

1. Professor Andrew Nee Yeh Ching, SMA Fellow, National University of Singapore (NUS)
2. Professor Kamal Youcef-Toumi, SMA Fellow, Massachusetts Institute of Technology (MIT)

CHAPTER 1

INTRODUCTION

1.1 Chapter Introduction

Chapter 1 begins with a discussion on the current state of robotics and the expected directions that will be undertaken by researchers and the industry. A major issue is that the ever increasing role of robots in our daily lives will require improved Human-Robot Interaction (HRI) interfaces, and this will in turn affect the way robots are programmed. The motivation for the work and the objectives are then presented. Lastly, the organisation of the thesis is described.

1.2 Current and Future Trends in Robotics

Advancements in the field of robotics have gained much momentum in recent years as researchers and the industry strive to set goals and directions for the future. This is a result of a number of drivers, summarized in Figure 1.1. Major breakthroughs in engineering, Artificial Intelligence (AI), Human-Computer Interaction (HCI), and Human-Robot Interaction (HRI) have made it possible to design more advanced robotic systems for a wider range of applications. The demands for higher quality of life will most likely be met by the integration of robots in our everyday lives to perform tasks that are uncomfortable, monotonous and/or even dangerous. Collective efforts to involve various researchers from different fields, for the undertaking of multi-disciplinary projects, are also more frequently observed [1-4]. These

collaborations are vital to ensure that research is carried out more efficiently and focused, with an effective pooling of resources and expertise. In the industrial context, higher labour costs and lower robot costs (from competitive manufacturing) will lead to higher demands for more intelligent robotic systems to replace human experts. Shortage of labour is also a major concern for countries such as Japan where low birth rates and an ageing population are expected to deprive the manufacturing sector of skilled workers [2].

Figure 1.1: Drivers for the advancement of robotics

Traditionally industry-focused, robotic systems are increasingly being used outside the factory floor, evident by the growing presence of service robots in personal environments. According to the United Nations (UN), robots can be classified into three categories; industrial robots, professional service robots, and personal service robots [4]. Industrial robots have long been used to increase productivity and efficiency through the automation of manufacturing processes, such as welding and

assembly. These robots have very little (or no) autonomy and are often programmed to perform repetitive tasks in structured environments. Professional service robots are similar to industrial robots because they also manipulate the environment they are in. However, these robots are more focused on the execution of specific tasks outside the factory floor that would otherwise be inconvenient and/or dangerous for humans. Examples of these tasks include landmine defusing and hazardous waste disposal. Lastly, personal service robots aid humans in everyday situations or environments. Examples of these robots include guide robots for museums, and serial manipulators for pumping fuel into vehicles and for helping the elderly or disabled perform rudimentary daily tasks in personal environments. In actual fact, researchers had identified these potential service applications over a decade ago and in a relatively short time, service robots have started to invade our environments more and more [5].

Of the three categories, the expected growth in demand is largest for personal service robots, followed by professional service robots and industrial robots. A report by the Japan External Trade Organisation (JETRO) reveals that in Japan (the largest market for robots in the world) alone, the sales of service robots is expected to grow from 10 billion Yen (~87 million USD) in 2006 to 55 billion Yen (~480 million USD¹) in 2010 [2]. In the near future, Japanese companies also predict that service robots adapted from industrial robots for performing specific tasks such as cleaning, will dominate the market. Although the growth in demand for industrial robots is not expected to be as large as in the case for service robots, the market share for industrial robots is still expected to be worth a significant 241 billion Yen (~2 billion USD) for the Japanese domestic market, and 487 billion Yen (~4.2 billion USD) outside Japan in 2010 [2].

¹ Based on an exchange rate of 1 USD = 114.877 Yen

1.3 Human-Robot Interaction

For industrial robots, HRI issues were largely confined to finding better ways to reconfigure or program robots. The interaction between human and robot is minimal where a master-slave relationship is often preferred. For example, teaching pendants are used to programme the robots without having to physically interact with them. More advanced interaction modes include Virtual Reality (VR) and Augmented Reality (AR) which are used to facilitate robot programming and task planning in a safer and more intuitive way. Professional and personal service robots on the other hand, require more sophisticated HRI modes that allow bi-directional interaction through mutual learning, and clearly defined task and autonomy guidelines [1, 4].

Despite advancements in robotic task automation, humans will always be a vital component in the robot programming process because of superior intelligence and flexibility. Therefore, HRI researchers are constantly seeking better user interfaces to improve existing mature methods of programming robots. These interfaces are expected to be very different as more enabling technologies are made available [4]. For example, for industrial-related tasks, more reliable and accurate measurement systems are needed to help users provide meaningful input and make better assessments of robot actions.

1.4 Motivation for the Work

In light of the reported trends, there is currently a pressing need to identify new methods for programming robots safely, efficiently and more intuitively. These

methods should address HRI issues for both service and industrial robots. Therefore, the focus of this research is on the development of new robot programming methodologies that are:

1. Faster, more efficient and more convenient.
2. More intuitive (as opposed to teaching pendants and traditional offline programming).
3. Flexible to different and/or unknown environments.
4. Safer for the user when present in the working environment of the robot.
5. Cost effective.

The potential of AR as an environment to facilitate these new methodologies will be explored. This is because AR has been successfully applied to other human-related tasks such as maintenance and assembly operations.

1.5 Objectives of the Work

The objectives of the work are as follows:

1. To propose the application of an AR environment for facilitating immersive robot programming and to discuss the benefits of AR over conventional robot programming methods.
2. To propose a new robot programming methodology using AR for tasks that have a number of possible path solutions for a given start and goal

configuration pair, such as pick-and-place tasks. These tasks are categorised as *Class I Tasks*.

3. To propose a new robot programming methodology using AR for tasks where the end-effector of the robot is constrained to follow a user-defined 3D path at a certain orientation with respect to that path, such as for arc welding and laser cutting. These tasks are categorised as *Class II Tasks*.

These two classes of tasks were selected because they encompass a wide range of industrial and service robotic applications. The physical manipulation of objects for Class I tasks are not considered or are assumed to have been successfully performed prior to planning the collision-free path, whereas Class II tasks are non-contact tasks.

1.6 Organisation of Thesis

Chapter 2 begins with a review on existing robot programming methods and a relatively recent concept known as Programming by Demonstration (PbD). A review of AR technology and a number of successful applications is also made. The potential and benefits of an AR environment for improving and facilitating PbD are then discussed. An approach known as Robot Programming using AR (RPAR) where the user directly interacts with a virtual robot is proposed.

Chapter 3 describes the RPAR system proposed and implemented in this research. The relationships between the main coordinate systems, the robot kinematics model and the overall system architecture are presented.

Chapter 4 proposes a first robot programming methodology for Class I tasks as defined in Section 1.4. The ability of the user to make sense of an unknown environment is utilized by enabling him/her to intuitively define a subset of the free space which is relevant to his/her goals. Collision-free paths for user-defined start and goal configurations are then generated using a beam search strategy. A number of pick-and-place case studies for the proposed methodology are then presented. The effects of the defined free space and beam search parameters on the final collision-free path are discussed.

Chapter 5 proposes a second robot programming methodology for Class II tasks as defined in Section 1.4. There are two major sections in the methodology, curve learning and end-effector orientation planning. The first step in curve learning requires the user to perform a number of demonstrations to obtain data points of a path/curve to be followed. These data points are processed by a learning system to produce a 3D output curve that best represents the human demonstrations/intentions. After the curve is found, the user is required to interactively define the free space along the curve. The orientation of the end-effector can then be planned intuitively using visual feedback from AR and a final collision-free path for the robot can be obtained. The results for a number of curve learning experiments and a number of case studies to demonstrate the overall methodology are then presented. The effects of the number of demonstrations and different curve learning methods on the quality of the output curve are discussed.

Chapter 6 summarizes the thesis by presenting the key contributions of the research and future research opportunities.

CHAPTER 2

ROBOT PROGRAMMING AND AUGMENTED REALITY

2.1 Chapter Introduction

Chapter 2 begins with a literature review on existing robot programming methods and AR-related issues. The primary contribution of the chapter is the proposal of an AR environment for immersive robot programming where the user directly moves a virtual robot. This is a departure from existing systems that are focused on telerobotics, and separating the human and robot. The benefits of an AR approach over traditional robot programming methods and a number of implementation issues are also discussed.

2.2 Robot Programming

Robots are usually designed to be able to perform a wide range of tasks. A popular definition of a robot by Russell and Norvig is "a reprogrammable, multifunctional manipulator designed to move materials, parts, tools, or specialized devices through various programmed motions for the performance of a variety of tasks" [6].

2.2.1 Existing Robot Programming Methods

Traditionally, there are three methods to generate a robot programme as defined by the Occupational Safety & Health Administration (OSHA) which is under the U.S Department of Labour [7]. These are:

1. **Lead-through Programming:** This is performed manually using a teaching pendant which will provide tasks commands for the robot (Figure 2.1(a)). The movements and configurations of the robot are recorded and 'played-back' for verification. For this method, the user might be present within the working environment of the robot and this raises safety issues.
2. **Walk-through Programming:** This is performed manually when the user moves the robot using physical contact (Figure 2.1(b)). Often, this method is undesirable due to safety issues. The power supply of the robot has to be turned off and the joint brakes need to be controlled carefully to avoid any accidents. Therefore, the method is useful only for Selective Compliant Assembly Robot Arm (SCARA) robots that do not 'collapse' after the brakes are disengaged.
3. **Offline Programming:** The robot programme is generated using a remote computer console and is subsequently transferred to the actual robot controller (Figure 2.1(c)). This method requires detailed knowledge of the work cell and robot models. Although the most complex among the three, offline programming provides the greatest flexibility in evaluating robotic solutions [8].

Figure 2.1: Traditional robot programming methods [7]: (a) lead-through programming, (b) walk-through programming, and (c) offline programming

The first two methods in Figure 2.1, also known as online programming, have a number of disadvantages. There are safety concerns for the user present within the working environment of the robot. The lead-through method using a teaching pendant is also slow and unintuitive. As mentioned earlier, the walk-through method is restrictive because of physical and mechanical constraints. In general, online programming of robots is slow and leads to lengthy downtime, and is risky for the human and robot. Physical robots may injure the human if not controlled with care, and collisions between the robot and obstacles will damage both.

Virtual Reality (VR) is an example of an offline programming method aimed at increasing the intuitiveness of the robot programming task for the human in a safe environment. Natonek *et al.* [9] described a VR system used for the training of robots for object manipulation in environments that are known *a-priori* (Figure 2.2). The characteristics of the system are: (1) provides the user with VR interaction devices to control the robot using a desktop-PC-based visualization system (at a separate location) that provides a *semi-immersive* environment, and (2) recognizes simple objects in the robot environment using *a-priori* knowledge that is constantly updated in the VR environment. In the VR system proposed by Aleotti *et al.* [10], a set of pre-determined classes for simple robot motions was constructed and identified based on hand gestures using a data glove (Figure 2.3). These identified motions were processed and transferred to a real manipulator for execution. However, as these VR systems use desktop-PC-based visualization and the user is not able to directly interact in the working environment of the robot, the level of intuitiveness is still low. In addition, the need to recognise known real objects restricts the flexibility applicable to different and/or unknown environments.

Figure 2.2: VR system by Natonek *et al.* [9]

Figure 2.3: VR system by Aleotti *et al.* [10]

Another example of an application of VR to robot programming and task planning is the work carried out in the Virtual Reality and Rapid Prototyping Laboratory (VRPL) at the University of Missouri [11]. The project was initiated by a local company who needed to verify if a robotic work cell design was viable. Although 20 drawings of the work cell and robot were provided, it was necessary to go down to the actual work cell. After measurements were taken, the work cell entities and layout were modelled in a virtual environment. Figure 2.4 shows the actual and simulated environments. Robot programming and task planning was then performed via simulation on the PC. The

major motivation for this approach was that despite extensive analysis in 2D, an extension to 3D was necessary to aid in the verification of the robot paths and other work cell activities. This was justified, when a collision in an initially-planned path which was overlooked by the company in its initial 2D analyses, was detected. It is important to note that the company was not ready to invest in a robotic solution until there was some way for them to be confident that the planned robot paths were free of collisions. The ability to evaluate various robotic options is a major advantage of VR, *i.e.*, offline programming.

Figure 2.4: VR for robot programming [11]: (a) actual robotic work cell, and (b) VR environment in a desktop-PC

The approach reported in [11] has the same disadvantages as the work in [9-10]. Extensive modelling of the environment is required. Therefore, the method is restrictive and inflexible to different and/or unknown environments. In addition, careful calibration and fine tuning is needed when replicating simulated environments. Although the user is provided with a *semi-immersive* feel where he/she is able to navigate in a virtual environment, the reported systems essentially use desktop-PC-based visualization. This reduces the level of intuitiveness for the user. On the other hand, *totally-immersive* environments like Cave Automatic Virtual Environment (CAVE) are complex and not cost-effective for simple applications. Typically, offline

programming packages also require the user to master complex programming languages that are software-specific and thus, non-generic. A recent and comprehensive review on VR applications in robotics can be found in [12].

2.2.2 Programming by Demonstration

Robot motion planning has been the subject of extensive research over the past three decades, involving an inter-disciplinary effort from largely mechanical engineers, mathematicians, and computer scientists [13]. The research is directed towards the creation of robots that possess the intelligence to plan and execute tasks successfully on their own using high level task descriptions. This turns out to be very complex, especially for applications that involve physical interaction between the robot and entities in environments that are unknown. In general, the task of robot programming is time-consuming and unintuitive [14-15]. A simple task that can be performed easily by a human (such as manoeuvring an arm through obstacles or holding a teapot) is often difficult to replicate using a computer-controlled robot. This is because humans possess the ability to perform complex manipulations without having to consciously perceive the detailed motion plans [16]. Aided with intelligence and sensory information such as vision and touch, the human is the 'near-perfect' robot system that is highly versatile and robust.

Recently, an approach known as Programming by Demonstration (PbD) has emerged primarily addressing intuitiveness in HR!. In this approach, the robot system *interprets*, and *improves* or *duplicates* actions based on human demonstrations [17]. Therefore, the human provides learning data and in some cases becomes the driving intelligence

behind the planning process. The PbD approach provides the user with an intuitive and quick way of programming robots by allowing the task to be demonstrated manually and leaving the robot system to observe, follow and learn from sensory information. A PbD system could potentially be as flexible as a human.

There is a common understanding that a PbD system needs to be generalized and not restricted to a single application [18]. However, development into realizing this goal is still at its infancy owing to the complexity of creating robust and adaptive learning systems. A more practical approach in designing a learning system would be to create interactive systems where humans provide appropriate domain knowledge using a generic interface. Figure 2.5 is a good illustration of a PbD system architecture which has two levels [19]. The top level involves *learning on the symbolic level* where the system either generalizes robot programmes completely or partially. Interpretation of user actions is carried out using pre-determined rules or rules acquired from processing experimental data. The lower level involves *learning on the sub-symbolic level* where basic skills such as the generating of end-effector trajectories from freehand motions, are learnt. Examples of geometric robot trajectory generation using human demonstrations are reported by Delson and West [20-22].

The involvement of the human in providing data for PbD applications raises the issues of data acquisition and data processing. The number of times that a user needs to demonstrate a particular task to provide sufficient information for the learning system needs to be determined. Multiple demonstrations are more practical when the planned task is to be executed many times and has higher performance requirements. On the other hand, single demonstrations are more practical for simple and quick applications

[23]. Data obtained through direct human demonstrations may not be optimal and unwanted data has to be removed [17]. Kaiser *et al.* [24] discussed the sources of sub-optimality that might occur with human demonstrations and the need for a PbD system to provide online adaptation for task refinement. These ideas were then tested for two tasks, inserting a peg into a hole and opening a door [25].

Figure 2.5: Principal architecture of a robot PbD system [19]

Researchers have also addressed the practical methods by which demonstration data can be collected. The choice of sensors depends on the process requirements, such as accuracy and reliability. Force sensors and position sensors have been used for pick-and-place operations [20-22] and data gloves for grasping operations [26-27]. Chen and Zelinsky [28] used video captures of a simple assembly process where time stamps were introduced for sampling frames. The data associated with these frames include force and position data which were used to derive a representation of the task geometry. The latter part of the work involved the generation of noise-free paths from noisy

demonstrated paths using the derived Configuration-space (C-space) [29] of the task. Alternatively, manipulations that would not be feasible in the real world because of physical constraints can be executed in VR environments, and inaccuracies of sensor-based approaches do not need to be critically taken into account [30].

Figure 2.6: Summary of robot PbD research

Figure 2.6 summarizes the robot PbD work carried out by various researchers. Despite the advancements in PbD, there are still a number of limitations in these reported works. It is still very difficult to realize PbD for complex contact manipulations, such as grasping assembly actions. However, a useful and successful application of PbD is in path planning where humans are able to navigate a robot through obstacles in unplanned environments. Delson and West addressed the sub-optimality issue in human demonstrations through identifying a satisfactory robot path that was free of 'wiggles', using a number of repeated human demonstrations and the inherent human variation [20-22]. A shortest Euclidean distance path can be identified from a range of allowable, obstacle-free demonstrations for a 2D planar space [20]. A path that is not necessarily shortest but depends on a proposed collision-free region for a 3D space can be identified as well [21]. The teaching gripper used by Delson and West is shown in Figure 2.7. Delson and West [22] also differentiated human variations from workpiece variations for constrained motions. However, their work does not provide any other means of evaluating collisions, between parts of the robot other than the end-effector, and the obstacles. The approach reported in [20-22] has further limitations in that multiple demonstrations of a similar nature are needed and the method is limited to a 2D or 3D C-space [29]. The number of similar demonstrations required for defining a satisfactory collision-free area is also unclear. In addition, it has been shown that the shortest Euclidean distance path for the end-effector does not necessarily mean the shortest joint travel for an articulated arm [31]. Most PbD literature also reports the use of data gloves, tracked grippers, etc., to obtain the data of the end-effector, so as to interpret the actions of the user. However, as mentioned earlier, it should not be assumed that other parts of the robot are also collision-free in the environment.

Figure 2.7: PbD system using a teaching gripper [21]

Another observation is that current PbD work separates the human and robot rather than allowing the human to directly interact with the robot. Even if the human demonstrations can be broken down to meaningful tasks that can be interpreted at a higher level, the successful execution of the task is dependent on whether these commands can be adapted to the particular manipulator concerned. This causes current PbD systems to be inflexible. On the other hand, direct interaction with the physical robot is usually impossible because of safety and mechanical constraints. A possible HRI interface that will be explored in this thesis is the enabling of the user to move and control a virtual robot, and intuitively evaluate whether the robot is able to successfully execute a task. The user will be able to directly demonstrate configurations of the robot in a manner similar to the walk-through programming method without physical constraints and safety concerns.

2.3 Augmented Reality

AR is an environment where computer-generated 3D objects are blended (registered) onto the real world view to enhance the user's real time interaction with the real world [32-35]. Figure 2.8 shows the Real-Virtuality (RV) Continuum described in [32].

Figure 2.8: Real-Virtuality (RV) continuum (adapted from [32])

AR is a subset of Mixed Reality (MR) or a combination of real and virtual entities in a scene. It does not aim to replace the real world but rather, supplement it with information that a human would otherwise be unable to have access to using his/her own senses [33]. AR involves interaction between the human-computer, and the human-real environment. Figure 2.9 describes the role of AR and other environments in facilitating these interactions.

Figure 2.9: A comparison of Human-Computer Interaction (HCI) styles [34]: (a) human-computer interaction isolated from human-real world interaction, (b) user immersed in computer world and isolated from real world, (c) user interacts with real world and computers placed in the real world, and (d) user uses a computer to aid in his/her interaction with the real world (AR)

2.3.1 Technical Issues in AR

The major technical issues that need to be considered when designing an AR system are shown in Figure 2.10. These will be discussed in the following sub-sections.

2.3.1.1 Optical or Video Head-Mounted Display

There are two major technologies that characterize the Head-Mounted Display (HMD) in AR, optical and video as shown in Figure 2.11. These HMDs allow the user to have

a view of the real world with virtual objects superimposed onto it. The selection of the appropriate technology depends very much on the application requirements. Various trade-offs need to be made between resolution, latency, safety, cost and so forth [33, 35]. For example, in an optical see-through HMD, the user has a direct view of the real world. Therefore, slight delays in registration of the virtual objects can easily be detected by the human eye. In video see-through systems, the user has no direct view of the real world and has to rely on video cameras. Therefore, streaming of the real world video can be delayed to match the registration latency of the virtual objects.

Figure 2.11: (a) Optical HMO, and (b) video HMO [33]

2.3.1.2 Tracking and Sensing Methods

Tracking and sensing are closely related to registration. Accurate registration requires positional information of the head of the user and real objects in the environment, obtained from tracking [33, 36]. The three requirements of a good tracking system according to [36] are:

1. Accurate both in terms of orientation (degrees) and position (mm).

2. Low combined latency (delay) of the system.
3. Operable at long ranges (especially for outdoor applications)

There are two major tracking approaches namely, *inside-out* and *outside-in* tracking. The former refers to the tracking of objects (static or moving) using a camera(s) attached on a moving object such as a HMO whereas the latter is the tracking of objects (static or moving) using mounted cameras that are stationary. Both methods are used to calculate the pose of the head (or view) relative to real world objects, so that virtual objects can be registered correctly. An example is the Personal Interaction Panel (PIP) input device shown in Figure 2.12, developed using the Studierstube software platform [37]. The PIP consists of a tracked pointer that is used to manipulate virtual controls registered onto a real flat board.

Figure 2.12: Personal Interaction Panel (PIP) input device using a tracked pointer [37]

Sensing can be carried out using a single or hybrid scheme. Sensors may operate based on time-frequency measurement, spatial scan, inertial sensing, mechanical linkages, and direct-field sensing [38]. For vision-based systems, markers are used extensively in realizing AR systems. The most popular software is the ARToolKit developed at the University of Washington [39]. For ARToolKit, image processing techniques are used

to identify marker features. The major weakness of marker-based systems is that tracking cannot be performed when these markers are occluded (not fully in the camera view) unless there is some form of marker redundancy.

2.3.1.3 Registration

Registration is the proper alignment of virtual and real objects relative to each other. Good registration ensures that the user perceives virtual objects to coexist with real objects. Unlike virtual environments, the demands for accurate registration in AR are higher [36]. Again, the question of 'how accurate?' depends on the application. For example, more accurate registration is required for medical applications which involve the overlay of images for surgery, compared to simple maintenance tasks which usually involve only text annotations.

There are two types of errors associated with registration, static and dynamic errors. Static errors are misalignments that occur even though the user and HMD remain perfectly still. Dynamic errors occur when the view point of the user changes, and are the major source of registration error [36]. An error model for AR systems which helps system architects determine sources of registration errors and the most significant contributors, sensitivity of the net registration error to input errors, nature of distortions caused by each type of input error, and the level of registration accuracy to expect as a function of the input errors, is described in [40-41].

Occlusion and collision detection are the two important issues in registration to ensure a realistic AR environment. When a virtual object is in front of a real object, the

section of the real object that should not be seen has to be occluded by the virtual object and vice versa. Collision detection prevents virtual objects from penetrating into each other or into real objects. For vision-based systems, both occlusion and collision detection are achieved using the 3D positional information of each pixel in the image.

2.3.1.4 Hardware

HMOs, wearable computers and input devices like pointers are used extensively in AR systems. Portability is a major hardware issue as most applications require movement by the user. Sufficient rigidity of HMDs minimizes static errors and is required for accurate tracking and registration. In addition, comfortable and ergonomically designed wearables are essential for optimal user performance. Wearable computing has been actively researched by groups such as the Wearable Group at Carnegie Mellon University [42]. The group has reported successful wearable systems such as the VuMan3 and Navigator 2, Adtranz and C-130.

A number of additional points regarding wearable devices were highlighted by major industry players at the third Augmented Reality for Development, Production and Servicing (ARVIKA) forum; ARVIKA is a European consortium created to develop practical AR systems for the manufacturing industry [43]. The concerns raised include the unsuitability of current data goggles for whole day use, the need for improving the management of large quantities of data on small screens, the importance of developing language control for user acceptance, and the increasing demand for smaller wearable computers that are able to handle more graphic intensive applications.

2.3.1.5 Cognitive Issues

Cognitive science is the study of the human brain's ability to analyse, reason and make decisions. As mentioned earlier, the primary goal of AR is to aid the user in his/her interaction with the real world. Therefore, an appropriate AR scheme for any particular application has to be designed based on an acceptable cognitive framework. The basic question that needs to be asked is what, when and how to display the relevant information. A framework on how to best design an AR scheme using cognitive theory was described in [44]. The work also reported that 50% of an operator's task time in manufacturing and maintenance is spent on cognitive activity, stressing its importance.

2.3.2 Applications of AR

AR has been widely applied to the areas of maintenance [44-48], assembly [49-51], computer-assisted surgery [33], and factory planning [52]. For assembly tasks, the user is guided by a reasonable amount of intelligence via an expert system that contains information of all feasible assembly states. The Augmented Reality System for Evaluating Assembly Sequences (AREAS) that guides the user in planning assembly sequences and warns the user of infeasible assembly states is described in [50]. An example of a successful industrial application of AR to assembly is the wire bundle assembly project by Boeing [51]. In these applications, annotations are used to provide instructions to the user in an environment that is known *a-priori* or information may be interactively retrieved by the user from the computer system. AR has also been used for computer-assisted surgery, where virtual images obtained from scans carried out on the patient are overlaid onto the body of the patient to guide surgeons [33]. The

application of AR for layout planning has also seen considerable success, most notably of the work carried out by Volkswagen where high level qualitative evaluations (subject to refinement) of a particular layout are performed [52]. In addition, this manufacturing planning system is able to overlay virtual human workspaces at the actual manual assembly station. Figure 2.13 shows a few examples of the AR applications described above.

Figure 2.13: Applications of AR: (a) maintenance [48], (b) mock tumour biopsy [33], (c) wire bundle assembly [51], (d)-(e) factory layout planning [52], and (f) human workspace in manual assembly station [52]

Compared to the applications in Figure 2.13, relatively little work has been reported on the application of AR in robot programming. One of the first robotic applications using AR was for telerobotic control [53]. Due to limitations in communication channels, there is a lag between the input of the operator, and the visual feedback and actual execution of the task. An operator is provided with visual feedback of a virtual wireframe robot superimposed over an actual physical robot located at its remote working environment. The virtual robot will execute a task for evaluation by the

operator, and if it is satisfactory, the task will be transferred to the real robot. The system is shown in Figures 2.14-2.15.

Figure 2.14: Telerobotic control using AR [53]

Figure 2.15: (a) Planning of a robot task and visual feedback, and (b) execution of the task [53]

AR has also been used for the programming of painting robots [54]. In this system developed by Asea Brown Boven (ABB), the handheld spray gun is tracked as the user performs the task and the corresponding virtual spray results are displayed to the user through a HMD. This feedback for evaluation is obtained by tracking the object to be sprayed using markers, and displaying results from a pre-defined relationship between the paint and the object model. The movements of the user are then generated into a robot programme.

2.4 Robot Programming using AR

AR functions as the interface that facilitates the flow of information between the computer system and real world. This process is controlled partially or fully by the human, who perceives the real world as AR. Partially here refers to the case where the computer system handles certain tasks automatically, such as when and how to extract sensor information. The information flow is illustrated in Figure 2.16. In the real world, information such as the partial model of the environment is extracted via sensors, and gestures, speech signals and input devices such as keyboards may be used for capturing user intentions. The computer system responds by rendering appropriate computer graphics in the view of the user. This includes annotations or instructions, and 3D virtual objects such as a virtual robot. Audio instructions on how to proceed with a task are also possible. Therefore, the computer system is responsible for processing real world input and producing results that may be used by virtual and real devices. For example, the results from a path planner can be used as input for a virtual robot model, a real robot or a real haptic device. The vital feedback from the real world to the computer system is unavailable in conventional Computer-Aided Design (CAD) and offline programming systems. Hence, the case for AR in robot programming is a promising one.

Figure 2.16: Flow of information in an immersive AR environment

A number of scenarios for robot programming involving real and virtual entities may exist, as shown in Table 2.1. However, only Scenarios 1-2 are considered as they are under the AR class.

Table 2.1: Scenarios for robot and environment entities

Scenario	Robot	Environment Entities	Class
1	Real	Virtual	AR
2	Virtual	Real	AR
3	Virtual	Virtual	VR
4	Real	Real	Reality

In Scenario 1, the major concern is whether the real robot is able to perform a particular task constrained by the layout of the work environment. For industrial applications, the workpieces and fixtures in a particular robot work cell would not vary too greatly in terms of size or type. In addition, the designer of the workpiece would have considered issues such as whether welding features, for example, are reachable by a welding torch held by a robot. The redesign of fixtures at the robot programming stage would be difficult to carry out in AR because of its inherent inaccuracies. Since the model of the real robot is available, a thorough fixture design process involving more rigorous analyses can be carried out at an early stage on a computer (offline).

There are a number of options for robot programming in Scenario 1. If the real robot is available, traditional online programming methods may be employed. However, these methods have various disadvantages as explained earlier. It is also difficult to perform programming on a virtual workpiece without any kind of force feedback for the tool the user is handling. It would be a challenge, for example, to select a point in space by visual means alone. Although haptic feedback can be employed, this would require

partial or complete modelling of the environment for collision detection, and the use of complex and expensive equipment. This form of programming would also be confined to robots and workpieces that are relatively small and reachable by the human, because it is performed in the actual full-scale work environment with a real robot. Hence, the role of AR in this scenario is limited to improving visualization that enables a high level check on the reachability of the robot in the environment. This is similar to the work at Volkswagen in [52].

This thesis proposes the application of an AR environment as defined in Scenario 2, for *immersive* robot programming where the user directly interacts and controls a virtual robot in a real environment. This is a departure from existing AR approaches that are more focused on telerobotics, or separating the robot and human [53-54]. An AR environment is suitable for developing interactive robot programming and complements the notion of PbD. This is because the human is allowed to participate in the path planning process, to perform certain tasks that are difficult to be simulated in a computer, such as recognizing and interpreting an unknown environment. AR also provides the flexibility to augment the view of a user with *visual guidance*. A virtual robot model can be superimposed over the real entities in a real environment, as shown in Figure 2.17.

Figure 2.17: Virtual robots in real environments: (a) miniature generic-purpose robot in a miniature kitchen, (b) miniature airplane washing robot, and (c) full-scale robot model

2.4.1 Benefits and Potential of AR for Robot Programming

The benefits of the proposed approach, Robot Programming using AR (RPAR), over traditional robot programming approaches are summarized in Table 2.2. Using a virtual robot means that a programming task can be carried out more safely than traditional online programming methods. An AR environment can also be created in the actual work cell, providing the flexibility and adaptability to different environments when an *in-situ* approach is desired. Hence, there is no need to extensively model the environment entities, which is a major disadvantage of YR. An *in-situ* approach also does not require careful calibration between the various entities in the environment which is the case when physically replicating a 'perfectly' modelled VR environment. Without having to transport a real robot to the actual work cell, RPAR allows the evaluation of various robotic options before the selection is made. The user is able to evaluate simulations of planned paths for different virtual robots. If a robot has been selected, programming can still be performed during the shipment period of the physical robot to the actual work cell. The approach proposed in this research is similar to the intuitive and quick walk-through method as it enables a human to directly move a virtual robot when planning a path. Therefore, the level of intuitiveness is high compared to lead-through and non-immersive offline programming. As mentioned in earlier sections, lead through programming requires the use of a teaching pendant and offline programming typically requires the user to master complex programming languages. Another possible application of an AR environment with a virtual robot is the programming of large robots where the walk-through method is infeasible (such as airplane washing robots as shown in Figure 2.17(b)) using scaled-down models of both the environment and the virtual robots.

The advantages of RPAR make it especially suitable for Small and Medium Enterprises (SME) where robots need to be re-programmed in a dynamic environment where tasks are frequently changed.

Table 2.2: Comparison between RPAR and traditional robot programming approaches

Method	Robot Type	Safety Issue	Environment Entities	Evaluation	Intuitiveness
RPAR	Virtual	No	Real	Flexible	High
Lead-Through	Real	Yes	Real	Inflexible	Low
Walk-Through	Real	Yes	Real	Inflexible	High
Offline (VR)	Virtual	No	Virtual	Flexible	Low

2.4.2 Targeted Applications for RPAR

The methodologies that will be proposed in this research are focused on geometric path planning (no consideration of velocities and accelerations) for unknown environments that are static, *i.e.*, entities are stationary. Therefore, a decoupled trajectory planning approach is adopted where the geometric path is first planned without considering the suitable velocities and accelerations needed to achieve the desired motions [55]. Robot dynamics may be included later but this is beyond the scope of this thesis. The goal is not to develop better dynamic models but to demonstrate how AR is able to improve the methods by which collision-free paths are obtained. The targeted tasks are divided into two classes as described earlier in Section 1.4 and are repeated here for convenience:

1. *Class I Tasks*: there exists a number of possible path solutions for a given start and goal configuration pair; such as for general pick-and-place, spot welding and inspection tasks.

2. *Class II Tasks:* the end-effector is constrained to follow a defined path, consistently at a certain orientation with respect to that path; such as for arc welding and laser cutting.

A number of examples of Class I tasks are shown in Figure 2.18. Figure 2.18(a) shows robot arms used for assisting the disabled [56]. Paralyzed patients that need to be taken care of in hospitals can benefit from the use of pre-programmed robot arms that are able to carry out tasks such as administering medicine. Figure 2.18(b) shows an industrial pick-and-place robot and Figure 2.18(c) shows an industrial spot welding robot [57]. Lastly, an experimental snake-like robot for bridge inspection is shown in Figure 2.18(d) [58]. The task of programming these hyper-redundant snake robots using conventional methods such as teaching pendants can be daunting.

Figure 2.18: Class I tasks: a) pick-and-place for disabled [56], b) pick-and-place for industrial applications [57], c) spot welding [57], and d) bridge inspection [58]

A number of examples of Class II tasks are shown in Figure 2.19. Figure 2.19(a) shows the SKYWASH airplane washing robot [59]. These robots are used to replace manual labour that is slow and dangerous for human workers who have to work in elevated areas. Figure 2.19(b) shows an industrial arc welding robot [60] and Figure 2.19(c) shows an industrial laser cutting robot [61].

Figure 2.19: Class II tasks: a) airplane washing [59], b) arc welding [60], and c) laser cutting [61]

2.4.3 Registration of Virtual Robot

The location of the base of the robot determines the final collision-free path in C-space that will be generated, as illustrated in Figure 2.20. Using RPAR, the user is able to

evaluate several possible base locations in the unknown environment (relationships between the entities and robot base are unknown) by moving the marker(s) that define the base, and verifying if the robot is indeed able to reach all sections of the environment relevant to the task. This verification can be performed by moving the virtual robot via its end-effector, or observing the superimposed workspace of the robot. In this research, the coordinate system of the base is used as the world coordinate system.

Figure 2.20: Registering the virtual robot in the real environment

2.4.4 Programming Large Robots using RPAR

As mentioned in Section 2.4.1, if scaled-down real physical models of the environment entities are available, AR can be used for programming large robots (using a scaled-down virtual robot) where the walk-through method would otherwise be impossible or uncomfortable to execute. Uncomfortable refers to the case where the robot has a relatively long reach and large workspace, which would require the user to move from one location to another, or stretch to reach different sections of the workspace. Examples are the bridge inspection robot in Figure 2.18(d), the airplane washing robot

in Figure 2.19(a), and large pick-and-place robots which are uncomfortable to directly manipulate in a full-scale AR environment. A scaled-down AR environment would still provide a more realistic experience for the user compared to a VR one.

For a scaled-down environment, a number of factors need to be taken into account as illustrated in Figure 2.21. When the path has been planned, it needs to be rescaled considering the potential errors. Safety factors and tolerances also need to be set to account for other uncertainties and to increase the level of confidence that the planned path is indeed reproducible in a full-scale environment. As the actual full-scale robot executes the path, online adaptations using real-time sensors can also be employed as reported in the SKYWASH [59].

Figure 2.21: Scaling and rescaling for programming large robots using RPAR, for the example of airplane washing robots

2.5 Chapter Summary

A thorough review on existing robot programming methods and Programming by Demonstration (PbD) was made. A new approach, known as Robot Programming

using AR (RPAR), where the user directly controls a virtual robot in a manner similar to the walk-through method, was also proposed. The benefits of RPAR over existing methods were discussed and examples of the two classes of tasks targeted in this thesis were presented. Lastly, a number of practical issues regarding RPAR were addressed. These were the registration of the virtual robot in the real environment, and scaling and rescaling for programming large robots. Overall, this chapter has presented the case or motivation for RPAR and the necessary background for the development of RPAR methodologies in later chapters.

CHAPTER 3

RPAR SYSTEM DESCRIPTION

3.1 Chapter Introduction

Chapter 3 presents the RPAR system used in this research. Firstly, the ARToolkit tracking method is briefly described [39]. The relationships between different coordinate systems and the robot kinematics model are then defined. Lastly, the system architecture and tools used in the AR environment are presented.

3.2 ARToolkit Marker-based Tracking Method

The ARToolkit method is based on the identification of 20 square markers. Each marker has a unique pattern printed on its centre that is asymmetric in at least one direction to avoid ambiguity. Tracking of the markers involves the following image processing techniques: (1) segmentation of the entire image and identification of the marker vertices, and (2) template matching of the pattern to determine the orientation of the marker. The ARToolkit algorithms are based on a pinhole camera model,

$$\rho \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = A [R | T] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (3.1)$$

where x, y are the 20 image coordinates corresponding to the 30 coordinates X, Y, Z , ρ is an arbitrary factor, A is the matrix of camera intrinsic parameters, and the matrix

$[R|T]$ of extrinsic parameters consists of a rotation matrix, R and translation matrix, T that relates the marker coordinate system to the camera coordinate system.

The ARToolkit method has three major disadvantages:

1. Inaccuracy: ARToolkit does not provide the level of accuracy required for metrology applications in manufacturing.
2. Jittering: Virtual objects shake unnaturally due to noise in the tracking data.
3. Occlusion: Rendering fails when the flat ARToolkit markers are blocked from the camera's view unless there is some form of marker redundancy.

Despite the limitations above, ARToolkit is sufficient for the purpose of demonstrating the methodologies developed in this research. More sophisticated tracking systems that are accurate, stable and robust towards marker occlusion can be employed. A comparison between ARToolkit and optical tracking systems [62-63] can be found in Appendix A.

3.3 Relationships between Main Coordinate Systems

There are three main Euclidean coordinate systems in the RPAR system namely, the camera, the base of the robot, and the wrist of the robot which is a probe held by the user to control and move the robot. This setup is based on the inside-out tracking approach as defined in Section 2.3.1.2. As an example in Figure 3.1, the B marker is used as the base and the A marker as the wrist. The relationships between the camera (X_C, Y_C, Z_C), base (X_B, Y_B, Z_B) and wrist (X_W, Y_W, Z_W) coordinate systems are obtained

using the marker detection method in ARToolkit, while the relationship between the wrist and the end-effector reference point $(X_{W'}, Y_{W'}, Z_{W'})$ coordinate systems is known. The end-effector reference point can also be the handheld probe tip used for the general interaction as shown in Figure 3.2(a). Figure 3.2(b) shows examples of handheld probes with different miniature end-effectors. The matrix M relates any two coordinate systems and consists of the R and T matrices. Therefore, the transformation from the base, B to the end-effector reference point, W' coordinate systems is given by,

$$\begin{aligned} {}^B M_{W'} &= {}^B M_c [{}^W M_c]^{-1} {}^W M_{W'} \\ &= {}^B M_c {}^c M_w {}^w M_{W'} \end{aligned} \quad (3.2)$$

Figure 3.1: Relationships between camera and markers

Figure 3.2: (a) Handheld probe for general interaction, and (b) handheld probes with different end-effectors

3.4 Robot Kinematics Model

The configuration of a robot can be characterized by its generalized coordinates, $q = [q_1, q_2, \dots, q_n]$ where n is the number of degrees-of-freedom (dot). To illustrate the RPAR methodologies, a six dof revolute joint robot (similar to the PUMA 560 industrial robot with three axes intersecting at the wrist [64]), as shown in Figure 3.3, is used. The angles of the joints are denoted as $q = \theta = [\theta_1, \theta_2, \dots, \theta_6]$. Figure 3.3 also shows an example of a rendered gripper-type end-effector corresponding to the physical probe with the same end-effector type. It is important to note that other than the PUMA 560-type robot, other models can also be used and this provides flexibility for the evaluation of various robotic options. The matrix B_M relates the end-effector reference point to the base and is the kinematics chain of the robot that needs to be solved.

Figure 3.3: Six dofrobot model and an alternative end-effector (bottom right)

In brief, the kinematics analysis can be divided into forward and inverse kinematics. The forward kinematics analysis is straightforward; given the configuration of the

robot, the unique position and orientation of the end-effector can be determined using the Denavit-Hartenberg (DH) convention [64]. The inverse kinematics of a six-dof robot is much more complex because there are multiple solutions how the robot is able to achieve a given end-effector orientation and position. However, setting allowable joint ranges or using the geometric approach (only one arm configuration type is picked at a time) avoids multiple solutions and unwanted configurations, such as singularities where the robot loses a dof. Avoiding multiple solutions at anyone time is essential to ensure that the process is intuitive for the user who is moving the robot. If the end-effector is moved out of the workspace of the robot, the robot will stop and remain at the last feasible configuration prior to the workspace violation. Both forward and inverse kinematics solutions used in the RPAR system have been reported by Craig [64].

3.5 RPAR System Architecture

The RPAR system architecture is shown in Figure 3.4. The system uses a Pentium 4, 2.4 GHz, processor with 512 MB RAM PC and was implemented using the video-based tracking method in ARToolkit and written in the C programming language. For each video frame, display of the virtual robot and other virtual elements is provided to the user through the HMO (Figure 3.5) using the OpenGL Renderer. The HMO of the RPAR system consists of a single IEEE Firefly camera [65] shown in Figure 3.6(a) and an i-glasses video display goggle [66] shown in Figure 3.6(b). The virtual display serves as feedback that is useful for interaction and the evaluation of outcomes from user actions.

The ARToolkit pose determination module calculates the pose of the robot wrist marker, or the pose of an input probe marker used for general interaction. Examples of general interaction are the generation of a Collision-Free Volume (CFV), and the collection of data points for learning a curve to be followed by the end-effector (Chapter 5). The robot models are represented by the Inverse Kinematics (IK), Forward Kinematics (FK) and torque/force modules. The IK module calculates the necessary generalized coordinates of the virtual robot based on the obtained end-effector pose from ARToolkit (for real-time user-controlled movement), or the calculated end-effector transformations from the path planner. The FK module calculates the coordinates of the robot components (*e.g.*, end-effector reference point) based on the generalized coordinates. This information is used for torque/force calculations and collision-free verification. The torque/force module calculates the amount of work or effort used by the robot for a particular movement, which is used to calculate the path cost. It is important to note again that for simplicity, a decoupled trajectory planning approach is adopted, where the dynamics of the robot are not considered during the geometric path planning stage [55].

The path planner consists of tools used for planning and generating collision-free paths. In this thesis, the path planner is made up of collision-free space generation, collision-free verification for robot components, path search, path generation, curve learning and end-effector orientation planning algorithms. Figure 3.7 shows the inputs and outputs to the path planner sub-modules and each of these components will be discussed in greater detail in their respective chapters. After a collision-free path (in terms of robot generalized coordinates) is found, it is simulated by the OpenGL Renderer for evaluation by the user. If the path is satisfactory, it will then be

transferred to the physical robot controller for further processing and execution (the physical robot in Figure 3.4 is a mock-up for illustrative purposes). The physical robot controller is not part of the current RPAR system.

If a full-scale robot and environment are used and the field-of-view (fov) of the camera is insufficient, the RPAR system can be modified to use larger markers and desktop-PC-based visualization. In this scenario, the user still directly interacts with the virtual robot in the real environment. However, if more sophisticated tracking systems are available, an outside-in tracking approach can be employed.

Figure 3.4: RPAR system architecture (excluding the physical robot controller)

Figure 3.5: HMD used for visualization

Figure 3.6: (a) PointGrey FireFly camera [65] and (b) i-glasses [66]

Figure 3.7: Tools developed for the path planner in this thesis

3.6 Chapter Summary

In this chapter, the RPAR system used for developing the methodologies in later chapters was described. The basic system architecture is generic and can serve as a basis for building a better and more comprehensive RPAR system. For example,

libraries for different robot models and path planning algorithms can be developed. Sophisticated tracking systems that are more robust and accurate can also be used to replace the ARToolkit pose determination module; this will greatly increase user acceptance of RPAR methodologies.

CHAPTER 4

RPAR METHODOLOGY FOR CLASS I TASKS

4.1 Chapter Introduction

The primary contribution of Chapter 4 is the proposal of a new robot programming methodology for Class I tasks as defined in Section 1.4, using the RPAR approach. The chapter begins by reviewing existing path planning methods, describing how the free space is defined and how path searches are conducted. Next, the overall proposed RPAR-I methodology is described and compared to existing path planning approaches, followed by more detailed explanations of its various sections. These consist of the role of the human in generating the free space relevant to the task, *i.e.*, the Collision-Free Volume (CFV), the demonstration/selection of the start and goal configurations in an intuitive way, and the beam search strategy used to generate the collision-free paths. A number of scenarios illustrating the methodology are then presented and discussed.

4.2 Overview of Path Planning Methods

4.2.1 Classical Path Planners

The three classical path planners are based on roadmaps, cell decomposition and potential functions [16]. Roadmaps are representations of the C-space in the form of nodes that correspond to distinct features (such as intersections in obstacles) and edges that represent the relationships between the nodes. The three popular roadmaps are

visibility graphs, Voronoi diagrams and silhouettes. Examples of a visibility graph and Voronoi diagram are shown in Figure 4.1. Search algorithms (*e.g.*, Dijkstra's algorithm) are used on these roadmaps to find shortest paths between start and goal configurations in the C-space. Secondly, cell decomposition is the division of the free space of the robot into cells that are adjacent to one another where the union of these cells (without overlapping) equals the entire free space. An example is the trapezoid decomposition of a planar C-space as shown in Figure 4.2. Here, the shortest path comprises of connections between the midpoints of the cells selected from the adjacency graph. Lastly, potential functions are a means of constructing a gradient field in C-space using differentiable real-valued functions (Figure 4.3). The simplest potential functions consist of the attractive and repulsive potentials. The former is a gradient guiding the search towards the goal which is the global minimum in the energy landscape, and the latter operates in the vicinity of obstacles (elevated sections of the energy landscape) by repelling the robot away. An example of a complete potential function planner was reported by Khatib [67]. The system consists of a mobile planar robot that uses sensors to measure distances between the robot and obstacles in real time.

Figure 4.1: Roadmap examples [13]: (a) visibility graph, and (b) Voronoi diagram

Figure 4.2: Cell decomposition [13]

Figure 4.3: An energy landscape for potential functions [13,67]

4.2.2 Sampling-based Algorithms

Sampling-based algorithms are more powerful alternatives to the classical planners which require the explicit representation of the entire C-space. A variety of techniques are used to obtain samples or points in the C-space where a robot is collision-free. For the Probabilistic RoadMap (PRM) planner, a coarse sampling resolution is used to obtain roadmap points before finer resolutions are used to construct the edges connecting these points [68]. Multiple queries (selection of start-goal configuration pairs) can be answered once a roadmap that represents the connectivity of the free C-space is found. Single query planners such as the Expansive-Spaces Tree (EST) planner use an exploratory approach that grows trees from defined configurations and focuses on solving a query as fast as possible instead of exploring the entire free space [69-70].

Sampling can be performed randomly, or deterministically using a grid. For an EST planner, a configuration in the neighbourhood of the node to be expanded is selected randomly from a uniform distribution and connected if it is valid, *i.e.*, collision-free. Guiding the sampling randomly as opposed to a greedy approach is aimed at avoiding

the search from being stuck at local minima; the greedy approach maintains only the best solution throughout the search. ESTs use sampling strategies that focus the search towards unexplored areas of the free space relevant to reaching the goal. A variant of the EST is the guided EST planner [71]. The guided EST has advantages over conventional ESTs when higher dimensional kinodynamic constraints are involved and the cost of the path is important. Kinodynamics is the consideration of dynamic constraints simultaneously when planning the geometric path. The applications presented in [71] were the docking of a space shuttle at the International Space Station (ISS) and the movement of a blimp in a factory environment. Guided ESTs incorporate more complicated neighbourhood sampling methods to create focused ESTs (compare Figures 4.4(a) and 4.4(b) for the same start and goal) and to search for a low cost path. A local path gradient descent is used to further optimize this path.

Figure 4.4: A guided EST focuses the search and uses a smaller search space (edges) [71]: (a) a standard EST, and (b) a guided EST for the same start and goal as (a)

4.3 Proposed RPAR-I Methodology for Class I Tasks

The proposed RPAR-I methodology is shown in Figure 4.5. In the first step, a user familiarizes himself/herself with the robot and the environment through moving the

robot in the environment. In the next step, the user manually generates a Collision-Free Volume (CFV) using the swept volume of a sphere attached to a probe. The details of the CFV will be presented in Section 4.4.1. The path(s) are planned through interactively selecting valid start and intermediate goal configurations in a sequential manner. A beam search algorithm is used to automatically generate the corresponding collision-free path(s) for the robot joints. This information is used to simulate the virtual robot performing the path(s) in the environment, for the user to visually evaluate the smoothness of the path(s) and for any potential collisions. If the results are unsatisfactory, the user may choose to repeat the process.

Figure 4.5: Proposed RPAR-I methodology

The goal of the methodology is to generate smooth paths consistently and more reliably as compared to these paths being directly demonstrated by a human. Therefore, the use of a *single* direct demonstration [23] (using the end-effector probe to move the robot along a path) as the final path, is undesirable. This is because the resolution and smoothness of the generalized coordinate profiles for the resulting path will be inconsistent due to either one or a combination of: (1) inconsistency in the human hand movements, (2) inconsistency in the sampling rate, and (3) a low sampling rate. Although smoothing can be performed as a post-processing step, it is difficult to determine the appropriate level of smoothness so as to guarantee that the path is collision-free [20].

The proposed methodology incorporates the ability of a human to make sense of a new environment and plan collision-free paths quickly. The CFV, which defines a subset of the entire free space, is manually generated in a manner similar to an artist 'painting on a 3D canvas'. Therefore, this approach is different from the use of multiple direct demonstrations of the same nature to define the free space as reported by Delson and West [20-22]. In addition, unlike classical offline planners described earlier, the CFV avoids the need to model the entities in the environment because this information is implicit in the volume. Therefore, any part of the robot would avoid obstacles if it remains within the volume.

A beam search strategy is adopted to generate paths for an n -dof robot. The beam search is appealing for its simplicity and has been used to solve problems in job scheduling [72] and assembly line balancing [73]. The beam search algorithm is similar to automated sampling-based algorithms, such as the EST [69-70], which use

sampling strategies that focus the growth of trees towards unexplored areas of the free space that are relevant to reaching the goal. In the RPAR-I methodology, the CFV focuses the search further by defining the boundaries for the path generation, and the explicit construction of the free C-space is not required. The beam search algorithm in this research also uses a different sampling strategy (this will be explained in Section 4.5.3) from the EST planners to maintain a number of unique paths during the search.

In practice, Class I tasks such as pick-and-place tasks require the robot to manipulate objects such as by gripping them. The RPAR-I methodology begins only *after* the manipulation of an object concerned is assumed to be successful, and the object is to be brought from the current start position to a goal position in the environment. Therefore, the space that is occupied by the object at the start configuration is in fact a part of the CFV; this is because the object to be gripped is no longer considered an obstacle. Before the selection of the start configuration in Figure 4.5, the following actions are taken: (1) the object is moved so that it does not occupy its original space, (2) a CFV is then generated at this previously occupied space, and (3) the object is placed back at its original position after the entire CFV has been generated.

4.4. Human Input from Demonstrations

4.4.1 Collision-Free Volume for RPAR-I

The CFV is generated from the swept volume of a sphere and represented as a set of virtual spheres intersecting each other, and displayed to the user through the HMO. With the CFV being generated and displayed to him/her in real-time, the user manually

generates the CFV in areas which he/she wants the robot to move to through dragging a physical sphere attached to a probe with a marker, as shown in Figure 4.6(a). There are two radii involved in the generation of the CFV, the virtual sphere radius, r_o and the physical sphere radius, r . The CFV is a set of n virtual spheres defined by,

$$CFV = \{ S_i(c_i, r_o); i = 1, \dots, n \} \in \mathbb{R}^3 \quad (4.1)$$

where S_i is a virtual sphere, characterized by a centre point, c_i and a radius, r_o .

The 3D coordinates of these centre points are determined with respect to the world coordinate system, which is the coordinate system of the base of the robot (B marker in Section 3.1). The choice of the physical sphere radius, r depends on a number of factors. In practice, the physical sphere should be larger than the virtual sphere, which is the basic unit of the CFV. Therefore, r is defined by,

$$r = r_o + (|\Delta r_{robot_error}| + |\Delta r_{tracking}| + |\Delta r_{modelling}|) \quad (4.2)$$

where Δr_{robot_error} is the maximum positional error of the physical robot within its workspace, which can be obtained from the manufacturer specifications, $\Delta r_{tracking}$ is the maximum error of the tracking system, and $\Delta r_{modelling}$ is the modelling error. The modelling error refers to uncertainties that may exist with the virtual robot model.

It should be noted that r depends on the level of resolution required of the environment (whether the sphere is able to access small spaces) and a combination of different radii can be used. Here, resolution refers to the physical geometrical constraints of the environment and "how small a gap" the robot has to move its end-effector through.

Using an AR environment, various options for visual feedback can be employed. The display of the CFV is made semi-transparent to allow the user to view the overall environment at all times, and enable a visual inspection of the quality and compactness of the CFV generated. The virtual robot is rendered during the generation of the CFV as shown in Figure 4.6(b). This is to enable the user to observe whether the robot is able to reach all sections of the CFV based on its model, and is important to increase the chances of the search finding a solution path within the CFV. Intuitively, the CFV can be viewed as a rough and gross estimate of a more precise path within it. The visual feedback also helps the user visually verify for collisions between the other less critical parts of the robot (*e.g.*, links 1 and 2) with any obstacles in the environment.

Figure 4.6: (a) Sphere attached to a probe with marker, and (b) generating a CFV over a rectangular block

The quality of the CFV can be evaluated based on its compactness and regularity in shape. The issue of interpolating between samples should not be considered for the CFV generation even though the sampling rate for this video-based system is low. This is because, although interpolation may be useful to obtain a more condensed CFV, it does not guarantee the collision-free property as illustrated in Figure 4.7. There are other practical ways to obtain a good CFV, such as using a faster sampling rate or

having the user move the sphere at a slower velocity. An adequate CFV can be obtained at a low frame rate of 10 frames per second (fps) if the user avoids fast and sudden movements. Lastly, the user can perform multiple passes of the sphere through the CFV to generate a more condensed and regular volume.

Figure 4.7: Interpolating between spheres does not guarantee that an object within the CFV is collision-free

4.4.2 CFV Check

After the CFV has been defined, the *CFV Check* can be performed to determine whether critical parts of the robot are within the volume and hence collision-free. This check can be used in real time as the user moves the robot and for the filtering process during the path search (this will be explained in Section 4.5). The choice of critical parts depends on the targeted application. For the examples in this thesis, the focus is on the end-effector, as it is essentially the part of a robot most likely to collide with obstacles. Therefore, it requires a more thorough verification as compared to other parts of the robot, which can be visually inspected during the simulation. For more complex cases where multiple critical parts need to be considered, a number of separate CFVs can be generated, each handling a particular critical part(s) of the robot. The *CFV Check* is performed through verifying whether a bounding volume for the end-effector is within the CFV. The bounding volume should also consider the

dimensions of the object that is being gripped by the end-effector. In this work, a bounding cylinder is used because it is a tighter bound as compared to a bounding box; the volume ratio of a cylinder to a bounding box with same height is $\pi j^2 h / 4 j^2 h \approx 0.79$. A simple method that verifies if a number of characteristic points on the discretized surface of the bounding cylinder are within the CFV is used (Figure 4.8). The coordinates of each characteristic point on the cylinder surface, P_i with respect to the base, B are given by,

$$\begin{aligned} {}^B T_{P_i} &= {}^B M_W {}^W T_{P_i} \\ &= {}^B M_W \begin{bmatrix} r_{cyl} \cos \theta_k \\ r_{cyl} \sin \theta_k \\ z_l \\ 1 \end{bmatrix}; \quad \in k, l \end{aligned} \quad (4.3)$$

where $\theta_k \in [0, 2\pi]$, $z_l \in [0, h_{cyl}]$, r_{cyl} is the radius of the bounding cylinder, h_{cyl} is the height of the cylinder and W is the wrist coordinate system.

Figure 4.8: Bounding cylinder and its characteristic points within the CFV

The coordinates, P_j are given by the translation matrix, ${}^B T_{P_j}$ in Equation (4.3). The number of points depends on the resolution of the discretized surface characterized by parameters θ_k and z_l . The *CFV Check* is described in Algorithm 4.1 where r_o is the radius of the CFV virtual sphere (from Equation (4.1)), there are $i = 1, \dots, n$ virtual spheres and $j = 1, \dots, m$ characteristic points.

Algorithm 4.1: CFV Check algorithm

-
1. Calculate the distances, $L_{j,i}^{p_j \rightarrow c_i}$ between each characteristic point, P_i and each CFV sphere centre, c_j .
 2. **If** $\left\{ \left((L_{1,1}^{p_1 \rightarrow c_1}) \leq r_0 \right) \text{ or } \left((L_{1,2}^{p_1 \rightarrow c_2}) \leq r_0 \right) \text{ or } \dots \text{ or } \left((L_{1,n-1}^{p_1 \rightarrow c_{n-1}}) \leq r_0 \right) \text{ or } \left((L_{1,n}^{p_1 \rightarrow c_n}) \leq r_0 \right) \right.$
and
 $\left((L_{2,1}^{p_2 \rightarrow c_1}) \leq r_0 \right) \text{ or } \left((L_{2,2}^{p_2 \rightarrow c_2}) \leq r_0 \right) \text{ or } \dots \text{ or } \left((L_{2,n-1}^{p_2 \rightarrow c_{n-1}}) \leq r_0 \right) \text{ or } \left((L_{2,n}^{p_2 \rightarrow c_n}) \leq r_0 \right)$
and
 $\vdots$
and
 $\left((L_{m-1,1}^{p_{m-1} \rightarrow c_1}) \leq r_0 \right) \text{ or } \left((L_{m-1,2}^{p_{m-1} \rightarrow c_2}) \leq r_0 \right) \text{ or } \dots \text{ or } \left((L_{m-1,n-1}^{p_{m-1} \rightarrow c_{n-1}}) \leq r_0 \right) \text{ or } \left((L_{m-1,n}^{p_{m-1} \rightarrow c_n}) \leq r_0 \right)$
and
 $\left. \left((L_{m,1}^{p_m \rightarrow c_1}) \leq r_0 \right) \text{ or } \left((L_{m,2}^{p_m \rightarrow c_2}) \leq r_0 \right) \text{ or } \dots \text{ or } \left((L_{m,n-1}^{p_m \rightarrow c_{n-1}}) \leq r_0 \right) \text{ or } \left((L_{m,n}^{p_m \rightarrow c_n}) \leq r_0 \right) \right\}$
(4.4)
 3. Entire bounding cylinder is within CFV, *i.e.*, collision-free.
 4. **else**
 5. Bounding cylinder is not entirely within CFV, *i.e.*, collision-free property not guaranteed.
-

Lemma 4.1: The complexity of Algorithm 4.1 is $O(mn)$.

Proof For each of the m characteristic points p_j , there are n distances to the virtual sphere centres c_i ; line 1 takes $O(mn)$. Line 2 performs $O(1)$ checks against r_0 for $O(mn)$ distances. Hence, the algorithm complexity is $O(mn)_n$

4.4.3 Selection of Start and Goal Configurations

After the CFV has been generated, the user selects start and goal configurations for the path(s) to be planned. This is done by moving the virtual robot using a probe with its tip resembling the end-effector (Figures 3.2(b) and 3.3), and recording the desired configurations. These configurations should be reachable by the robot (which can be verified using the kinematics model), and the critical part or the end-effector should be

within the CFV. To avoid the CFV from blocking the view of the end-effector, the virtual spheres are displayed as smaller dots to indicate the location of the CFV so as to guide the user during the selection process. If the end-effector is outside the CFV, the *CFV Check* will detect it, the colour of the dots will be changed to red and a warning will be shown. The proposed approach enables multiple queries to be carried out within the CFV generated, where a query to the system for a path is made every time a pair of start and goal configurations is selected. Therefore, this approach has the characteristics of both multiple-query planners [68], where multiple paths can be planned using the current CFV without having to generate a new one, and single query planners [69-70] where the focus is on finding a path for a given start and goal configuration pair without constructing the entire roadmap.

4.5 Beam Search Strategy for Generating Collision-Free Paths

The search problem can be stated as the determination of an optimal or good path in an n dof robot joint C-space that is characterized by a cost measure (this will be explained in Section 4.5.2), given the start and goal configurations. The path to be determined and generated should possess the following characteristics: (1) smoother and with fewer wiggles than when performed directly by a human where unnecessary movement causes noise [20-21], (2) collision-free for critical parts of the robot, *i.e.*, within the CFV, and (3) all configurations along the path are executable by the robot.

4.5.1 Path Representation

The first issue to be considered is the method of representing the path. At each current search step, a neighbourhood region for the current node or head node of a path is

generated. This region consists of all the possible perturbations on the generalized coordinates q of the robot. Each perturbation of any particular joint} can be expressed as either decreasing, maintaining or increasing the current value by a predefined magnitude $\varepsilon_j > 0$,

$$\Delta q_{j,t} = -\varepsilon_j \text{ or } 0 \text{ or } \varepsilon_j; \quad \in j = 1, \dots, n \quad (4.5)$$

$$\Delta \mathbf{q}_{c,t} = [\Delta q_{1,t}, \Delta q_{2,t}, \dots, \Delta q_{n,t}]; \quad c = 1, \dots, 3^n \quad (4.6)$$

where} represents the robot joints, and c the different perturbation combinations in radians. For n dof, there are 3^n (three possible ways to perturb each joint as shown by Equation (4.5)) possible perturbation combinations. In general, the perturbation magnitudes may vary for different joints depending on the actuator resolutions.

To obtain the neighbourhood region, these perturbations are added to the current nodes of the paths being maintained. For a particular path which consists of one configuration per search step,

$$\mathbf{q}_{t+1} = \mathbf{q}_t + \Delta \mathbf{q}_{c,t} \quad (4.7)$$

where $\mathbf{q}_t = [q_{1,t}, q_{2,t}, \dots, q_{n,t}]$ is the configuration at search step t and $\Delta \mathbf{q}_c$ is a perturbation combination c chosen by the search at search step t .

The final path returned as a solution consists of the start, intermediate and goal nodes, which are similar to many discrete waypoints in online programming. The start and goal nodes are selected by a user as described in Section 4.4.3. This path, consisting of $m + 1$ nodes, can be represented by,

$$\underbrace{(Start = \mathbf{q}_0) \rightarrow (\mathbf{q}_1 = \mathbf{q}_0 + \Delta \mathbf{q}_{c,0}) \rightarrow \dots \rightarrow (\mathbf{q}_{m-1} = \mathbf{q}_{m-2} + \Delta \mathbf{q}_{c,m-2}) \rightarrow (Goal = \mathbf{q}_m)}_{m \text{ search steps, } m+1 \text{ nodes}} \quad (4.8)$$

4.5.2 Path Cost

The path cost will be used by the search to compare between paths during the node filtering process, which is a local optimization step (this will be explained in Section 4.5.3). It is a measure that is used to evaluate the amount of work or energy needed by the joints to execute a path, and also an objective way of measuring the smoothness of a path. The path cost may depend on the robot dynamics model. However, the search procedure is independent of the path cost formulation because as mentioned earlier, a decoupled trajectory approach is adopted [55]. To demonstrate the procedure and for the purposes of simplicity, the path cost used in this work is formulated with the assumption that the accelerations and velocities are small and negligible. The path cost only considers the kinematics of the robot and quantifies the effects of gravitational forces (weights of links) on the configurations of the robot at each search step.

For the joint generalized forces, $\mathbf{u} = [u_1, u_2, \dots, u_n]$ there are two types of joint movements, rotational or linear. The more extended the configuration, the higher the torques or forces at the joints due to higher gravitational moments, compared to a less extended configuration. These gravitational moments are reflected through two force components for each rotational or linear joint, perpendicular (normal) and parallel (tangential) to the movement plane as shown in Figure 4.9. Therefore, for each joint j , the generalized force at search step t , is defined as,

$$u_{j,t} = \begin{cases} \tau_{j,t} = \mu_j F_{N,j,t} r_j + F_{T,j,t} r_j & \text{if joint } j \text{ is rotational} \\ F_{j,t} = \mu_j F_{N,j,t} + F_{T,j,t} & \text{if joint } j \text{ is linear} \end{cases} \quad \in j = 1, \dots, n \quad (4.9)$$

where $\tau_{j,t}$ and $F_{j,t}$ are the torques and forces respectively for joint j at search step t , $F_{N,j,t}$ and $F_{T,j,t}$ are the normal and tangential forces respectively for joint j at search step t , μ_j is the bearing coefficient of friction at joint j , and r_j is the distance between the rotation centre and the tangential force for rotational joints.

Figure 4.9: Forces due to gravitational moments for (a) rotational joints, and (b) linear joints

The path cost between two search steps is obtained by multiplying the average torque or force between the two steps, with the magnitudes of the perturbations. For rotational joints, the perturbation is angular, $\Delta q_{j,t} = \Delta \theta_{j,t}$ in radians, and for linear joints, the perturbation is the distance of travel, $\Delta q_{j,t} = \Delta d_{j,t}$ in metres. For a particular path, the *current-path-cost* is the cost from the start, $t = 0$ to a certain search step, $t = T$,

$$\begin{aligned} \text{current-path-cost} &= \sum_{t=1}^T \sum_{j=1}^n \frac{(u_{j,t} + u_{j,t-1})}{2} |\Delta q_{j,t-1}| \\ &= \sum_{t=1}^T \sum_{j=1}^n \bar{u}_{j,t} |\Delta q_{j,t-1}| \end{aligned} \quad (4.10)$$

where T is the latest or current search step at the current node of the path, $\bar{u}_t$, is the average generalized force for joint $\}$ at search step t , $\Delta q_{j,t}$ is the perturbation for joint $\}$ chosen by the search at search step $t-1$, and *current-path-cost* is in Joules (J).

For Equation (4.10), when $T = m$, the *current-path-cost* becomes the *total-path-cost* and both are always positive.

4.5.3 Heuristic Beam Search

The beam search algorithm uses a heuristic to guide the search towards the goal, and performs local optimization of the path cost along the way. Therefore, the primary objective of the search is getting to the goal configuration followed by improving the path cost during the search whenever possible. In the beam search algorithm, the search space is represented as a tree, and the root node as the start node (or state), S . Each node has a branching factor of $b = 3^n$ which is the number of possible perturbation combinations. The size of the search tree is $O(b^h)$, where h is the height of the beam or the length of the final path (Figure 4.10). The beam search approach prunes current nodes that are ranked worse than the best k nodes. The parameter k is the maximum number of nodes that are selected, and is a constant parameter known as the maximum *width of the beam* (Figure 4.11). The beam search algorithm is attractive because the memory requirements are linear $O(kh)$ rather than exponential as in the case of a full or exhaustive search. The beam search algorithm used in this work is shown in Algorithm 4.2. A more detailed version of the pseudocode can be found in Appendix B.

Figure 4.10: Full search space with the start configuration as the root node

Figure 4.11: Beam search concept where a heuristic guides the search to determine which portions of a current slice should be kept [74]

Algorithm 4.2: Proposed beam search algorithm

-
1. Set the first and only path (length = 1) to the start node, S.
 2. Set saved *goal-path* = NULL.
 3. Repeat
 4. Obtain neighbourhood region.
 5. Filter current nodes and delete corresponding unwanted paths.
 6. Select at most k best unique paths using heuristic and delete unwanted paths.
 7. **If** any current node reaches the goal, G and a current *goal-path* is found
 8. **If** saved *goal-path* = NULL or current *goal-path* cost < saved *goal-path* cost
 9. saved *goal-path* = current *goal-path*.
 10. Verify if the saved *goal-path* has the lowest cost among all paths.
 11. **If** other possibilities of reaching the goal at a lower cost, exist.
 12. Delete paths with higher costs than the saved *goal-path*.
 13. else
 14. Return saved *goal-path*.
 15. **Until** saved *goal-path* is returned.
-

Algorithm 4.2 maintains at most k unique paths during the search for the goal. Line 4 perturbs the head nodes or the current nodes of all the paths maintained, to produce $O(3^k k)$ neighbours, which are the 'frontline' of the search. Line 5 filters and deletes the nodes (and their corresponding paths) that are: (1) not within the CFV, (2) configurations that are unreachable or infeasible according to the robot model, (3) revisited nodes in order to maintain only one path to any particular node, and (4) duplicate nodes where more than one path reach the same neighbour or configuration.

Filter (3) avoids the search from maintaining subset paths of only the greedy path and thus stifling the exploration. The greedy path is when $k = 1$ and only the single best path is kept. This occurs when nodes are revisited although a previous path to that node already exists as illustrated in Figure 4.12. It also avoids cycling, *i.e.*, perturbation combinations that bring the path to a previous node already existing in the path (Figure 4.12). When there are duplicate nodes where two or more paths reach the same configuration at the same search step, the fourth filter chooses only the path with the lowest *current-path-cost* among those duplicate paths (Figure 4.13). This is a local optimization step.

Figure 4.12: Filter (3) prunes revisited nodes/states and their corresponding paths ($\rightarrow 4 \rightarrow 5 \rightarrow 1$), and avoids cycles within a path ($\rightarrow 1 \rightarrow 2 \rightarrow 1$)

Figure 4.13: Filter (4) deletes duplicate nodes (Node 5 for path $\rightarrow 3 \rightarrow 4 \rightarrow 5$) and selects only the lowest cost path to that particular node

Line 6 is the essence of the beam search algorithm, where the best k current nodes in terms of the smallest generalized coordinates distance to goal, *dist-to-goal* and their corresponding paths are selected, and the remaining unwanted nodes and their corresponding paths discarded. The *dist-to-goal* for a particular node at search step t is a heuristic defined by,

$$dist-to-goal = \sqrt{\sum_{j=1}^n (q_{j,t} - q_{j,G})^2} \quad (4.11)$$

where G represents the goal node.

Lines (4-6) are repeated until a goal is found, when for any current node,

$$|q_{j,t} - q_{j,G}| \leq \epsilon_j; \quad \in j = 1, \dots, n \quad (4.12)$$

For Equation (4.11), if a combination of rotational and linear joints exists, the joint values (and the perturbations) can be multiplied by weighting factors so that both rotational and linear joints are within the same order of magnitude.

Lines (7-14) verify whether (1) a current *goal-path* found has the lowest possible cost or (2) there are paths that have not reached the goal, but may have the potential of reaching the goal at a lower cost in the later search process. This is another local optimization step and is achieved by keeping the paths that have lower *current-path-costs* than the saved *goal-path*, and discarding those paths that have higher *current-path-costs*. When a new or current *goal-path* that has a lower cost than the previous best *goal-path* is found, the saved *goal-path* is updated with this better solution. Eventually, when a current *goal-path* has been found with the lowest cost, and there are no other possible paths that can be found with lower costs than this path in the later search process, this *goal-path* will be returned as the final solution (line 14).

Lemma 4.2: The complexity of Algorithm 4.2 is $O(3^n kV)$ where n is the number of dof of the robot, k is the beam size, and V is the total number of vertices/nodes possible in the free C-space of the robot that is discretized according to the perturbation resolutions of the robot',

Proof For each current node in the neighbourhood region of size $O(3^n k)$:

1. A *CFV Check* is performed and the current node deleted if it cannot be guaranteed to be collision-free. This takes $O(1)$ ³.
2. The current node configuration is checked whether it is reachable by the robot based on the kinematics model and deleted if it is invalid. This takes $O(1)$.
3. The *current-path-cost* and *dist-to-goal* for the current node are calculated. This takes $O(1)$.

² It is important to note again that the explicit construction of the C-space is not required in the beam search strategy (Section 4.3).

³ *CFV Check* is based on *constant* numbers of characteristic points and virtual spheres, which are fixed prior to the search. Hence in Algorithm 4.2, it takes $O(1)$.

4. The current node is compared with every node in the paths maintained, to check if it is a revisited or cycled node. This takes $O(kt)$ where t is the current search step or the current length of the paths maintained.
5. If two or more paths reach the same configuration at the same search step, only the path with the lowest *current-path-cost* is kept and duplicate paths deleted. This is performed by comparing the current node with every other current node in the neighbourhood region and takes $O(3^n k)$.

Steps (1-5) (or lines (4-5) in Algorithm 4.2) above take,

$$\begin{aligned} & O(3^n k) \left(O(1) + O(1) + O(1) + O(kt) + O(3^n k) \right) \\ &= O(3^n k) O(kt) + \left(O(3^n k) \right)^2 \end{aligned}$$

As $kt \rightarrow V$, the first term becomes more significant than the constant $\left(O(3^n k) \right)^2$ because $V \gg 3^n k$. Hence, steps (1-5) take $O(3^n kV)$.

In line 6 in Algorithm 4.2, the remaining current nodes are sorted according to *dist-to-goal* using Quicksort, and the best k nodes are selected for maintenance; this takes $O(3^n k \lg(3^n k))$ [75]. For lines (7-14), at most $k_{current}$ nodes maintained are checked for the existence of other lower cost paths to the goal, should *at least one goal path* be already found. This involves comparing the *current-path-costs* with a current saved *goal-path* and takes $O(k)$.

Hence, lines (6-14) in Algorithm 4.2 take $O(3^n k \lg(3^n k)) + O(k) = O(3^n k \lg(3^n k))$.

Therefore, the complexity of Algorithm 4.2 is,

$$\begin{aligned} & O(3^n kV) + O(3^n k \lg(3^n k)) \\ & = O(3^n kV), \quad V \gg \lg(3^n k) \end{aligned}$$

Lemma 4.3: Beam search is not complete, *i.e.*, it cannot guarantee that a solution will be found, if it exists. However, the probability of finding a solution path, if it exists, increases as the search steps t and beam size k increase.

Proof It is well known that local heuristic-based searches cannot guarantee completeness [74-75]. At each search step in Algorithm 4.2, the number of free and available vertices is $O(V - kt)$ where V is the total number of vertices possible in the free C-space of the robot, k is the beam size or number of unique paths maintained, and t is the search step. Therefore, the probability of a particular neighbourhood region containing vertices of a solution path(s), if it exists, is

$$\frac{O(3^n k)}{O(V - kt)}, \quad V \geq kt, \quad O(3^n k) \leq O(V - kt).$$

Noting that the numerator is a constant, as t increases and $kt \rightarrow V$, the probability of finding a solution increases. Increasing k also helps improve this probability through both the numerator and denominator. However, the search cannot guarantee finding a solution if a neighbourhood region cannot be generated, *i.e.*, the search is stalled.

The probability of finding *at least one* feasible neighbour at each search step is

$$\frac{O(3^n k) - O(W)}{O(3^n k)}, \quad O(W) \leq O(3^n k) \text{ where } O(W) \text{ is the number of nodes removed from}$$

the neighbourhood region after the filtering process. $O(W)$ is dependent on the

topology of the CFV and the robot model. As k and the number of dof for the robot increase, the probability that the search would be able to manoeuvre out of a tight spot and avoid getting stalled, increases. This improves the robustness of the search.

Lemma 4.4: Beam search cannot guarantee a globally optimal solution.

Proof Beam search cannot guarantee a globally optimal solution as with all other local heuristic-based searches [74-75]. For example, in Figure 4.11, there is a possibility that the globally optimal solution exists towards the right, outside the beam. In general, good heuristics and filtering schemes, and larger beam sizes are able to produce locally optimal solutions. In Algorithm 4.2, local optimization is performed through lines 5, and lines (7-14)

The heuristic from Equation (4.11) is the shortest distance between the start and goal configurations in the C-space of the robot irregardless of the presence of obstacles in that direction. Hence, the expansion of the path neighbourhood regions will be in the general direction of the heuristic. For the 2D robot C-space example in Figure 4.14, the path costs are assumed to be optimal in the direction of the heuristic".

Based on *Lemma 4.3*, if the search is stalled, a solution path may be found, if it exists, after many search steps. If only the greedy choice is kept ($k = 1$), this path will be very undesirable as the search tries to find a way out by backtracking if it cannot proceed in the initial heuristic direction due to the CFV boundary and/or constraints by the robot model. This is shown for the example path between the start, S and goal, G in Figure 4.14(a). According to *Lemmas 4.3 and 4.4*, increasing the beam size k improves the

⁴ Strictly speaking, the lower cost paths might not be *exactly* in the direction of the heuristic, depending on how the path cost is formulated.

probability of finding a solution, and this solution would be a locally optimal one. This is shown in Figure 4.14(b) where alternative paths avoid the search from getting stalled.

One practical solution for paths that are winding and not in a general direction towards the goal is to plan a sequence of multiple paths based on human-input as additional heuristics. Using the RPAR system, the human is able to define intermediate goals so that the beam search would only have to look for piecewise solutions that are in the general direction of the heuristic from Equation (4.11), and perform local optimization of each piecewise path in the sequence. The major drawback is that the path might only be locally optimal as shown in Figure 4.14(c), because the solution is dependent on where the intermediate goal G' is placed. However, with larger k 's, the path cost can be further improved (path moves closer to the globally optimal one) as shown in Figure 4.14(d). As discussed earlier, the trade-off is between the quality of the solution and the memory and computational costs of a fully exhaustive search.

Figure 4.14: Example of the beam search approach in a 2D robot C-space

4.6 Results and Discussions for Class I Tasks

This section presents the results and discussions of the experiments carried out for Class I tasks. The goal of these experiments is to observe the effects of generating different CFVs and varying the beam size, k . To demonstrate the proposed RPAR-I methodology, the six dof rotary robot model ($q = 9$) in Section 3.4 is used. The calculation of the path cost is based on the actual robot size which is 10 times larger than the scaled-down virtual model and for simplicity, the angle perturbations are set to be equal for all joints, *i.e.*, $\varepsilon_j = 0.5^\circ (\pi / 180)$ rad, $\in j = 1, \dots, n$. For any particular path planning process, there are a number of possible scenarios.

4.6.1 Scenario 1: The neighbourhoods for the greedy path do not hit the boundary of the CFV

In this scenario, the CFV is relatively compact and regular in shape, such as the example for the scaled-down robot shown in Figure 4.15. A query is made by recording a pair of valid start and goal configurations. The fastest and *final* solution returned is the greedy path solution ($k = 1$). Without loss of generality, take the example of moving from configuration or state $\langle 0, 0, 0 \rangle$ to $\langle 1, 1, 1 \rangle$ where the available perturbations at each node are $\langle +1, +1, +1 \rangle$, $\langle +1, 0, 0 \rangle$, $\langle 0, +1, 0 \rangle$, $\langle 0, 0, +1 \rangle$, and $\langle 0, 0, -1 \rangle$. If the neighbours *are always valid at every search step, i.e.*, the neighbourhood does not hit the boundary of the CFV, then the greedy choice would be $\langle +1, +1, +1 \rangle$ to reach the goal in *one search step* according to the *dist-to-goal* heuristic. This is indeed the fastest way of reaching the goal compared to using other neighbours which would take at least three steps (*e.g.*, $\langle 0, +1, 0 \rangle \rightarrow \langle +1, 0, 0 \rangle \rightarrow \langle 0, 0, +1 \rangle$).

$0, +1>$ or $<0, +1, 0> \rightarrow <+1, 0, 0> \rightarrow <0, 0, -1> \rightarrow <0, 0, +1> \rightarrow <0, 0, +1>$. When the neighbourhoods for the greedy path do not hit the boundary of the CFV throughout the search, the constraints set by the CFV did not influence the solution. In other words, every node in the final greedy path solution would have always been available for selection based on the *dist-to-goal* heuristic. Since the perturbation magnitudes for all joints are equal, the angle profiles for each joint in Figure 4.16 are linear with a fixed slope of $\pm \varepsilon_j / \text{time_step}$ prior to achieving their goal values (slope = 0). The final goal configuration was reached after 86 steps.

The greedy path solution is also locally optimal with respect to the path cost formulation in Section 4.5.2. As mentioned above, it is the first path to reach the goal. Subsequently, all other possibilities of obtaining a lower cost path solution from the remaining paths maintained that have yet to reach the goal, would be explored. The greedy path solution is locally optimal in terms of its path cost because it is the final solution returned, *after* all these other possibilities have been exhausted.

It was also observed that increasing k did not help improve the cost of the path solution. Intuitively, the path cost in Equation (4.10) is a function of joint travel and is always positive. Therefore, smoother and shorter paths (fewer search steps) like the greedy path solution which only requires $k = 1$, tend to have lower costs compared to longer, noisy or 'wiggly' ones as defined earlier in Section 4.5.

Figure 4.15: Scenario 1: (a) a well-defined compact CFV, and (b) the final path (search steps = 86, *total-path-cost* = 106.0423 J, and final *dist-to-goal* = 0.0075 rad)

Figure 4.16: Solution path for example in Scenario 1 with resolution for all joints = 0.5°

4.6.2 Scenario 2: The neighbourhoods for the greedy path hit the boundary of the CFV

This scenario occurs when the boundary of the CFV imposes non-linear constraints that force the joints to re-adjust before achieving their goal values as shown earlier in

Figure 4.14(b) and 4.14(d). The boundary can be described as the outer surface of the CFV separating the internal volume from the free space and/or obstacle(s). Hence, the quality of the solution is dependent on the quality of the CFV. When the end-effector hits the boundary of the CFV, there is a possibility of slightly jagged movements in the angle profiles if the boundary is not smooth as illustrated in Figure 4.17. In Figure 4.17, if the search chooses to move the end-effector from position 1 to position 2 and then from position 2 towards the goal G in the direction of the vector $1 \rightarrow 2$, the end-effector will hit the boundary of the CFV. Hence, the search has to alter its direction upwards and move the end-effector to position 3. However, the quality of the CFV can be improved by using higher sampling rates (higher than the 10 fps used in this work), moving the physical sphere slowly or using multiple passes of the sphere through the CFV. Using larger magnitudes for the perturbations also avoids the search from being drawn into small areas caused by a non-smooth CFV boundary. However, using lower resolutions will weaken the assumption that all the intermediate points between the larger steps are collision-free.

Figure 4.17: Jagged movements along the CFV boundary may exist if the CFV is not compact enough, *i.e.*, boundaries are not smooth

A number of experiments were carried out to study the effects of increasing k and two examples are shown in Figure 4.18. Re-adjustments for certain joints are needed when

the end-effector hits the boundary of the CFV. For example, Figure 4.19 shows that re-adjustments for joints 2 and 3 are needed for Example I, when they are unable to continue moving in the direction of the heuristic (the final goal configuration was reached after 134 steps). Small jagged movements caused by the boundary can also be observed for these joints. However, Figure 4.20 shows that increasing k improves the path costs for both Examples I and II. This is because for $k > 1$, the search considers paths that are further away from the boundary of the CFV, other than the greedy path which is close to the boundary. These paths are less jagged than the greedy path and have potentially lower path costs. When the search considers these options, local optimization of the path can be carried out and improved solutions can be obtained.

Figure 4.18: Scenario 2: (a) Example I: when the end-effector hits the boundary of the CFV, (b) the solution path showing the circled portion along the boundary of the CFV, and (c)-(d) Example II

Figure 4.19: Solution path for Example I in Scenario 2 with resolution for all joints = 0.5° ($k = 10$, search steps = 134, *total-path-cost* = 59.3368 J, and final *dist-to-goal* = 0.0074 rad)

Figure 4.20: Effects of increasing k on the path costs for both examples in Scenario 2

4.6.3 Scenario 3: The search is stalled

Occasionally, when the CFV contains defects, such as holes, the search may be stalled as shown earlier in Figure 4.14(a). These holes are 'artificial obstacles' that may occur when the CFV is not compact enough during the generation process. In such a scenario, most of the search time is spent on trying to find a way around the defect rather than reaching the goal. By increasing k , the availability of several alternative paths improves the search's chances of finding a solution and therefore, improves its robustness. For the example in Figure 4.21, a hole was introduced to the CFV on purpose and the search was stalled when $k = 1, 5$ and 10 but was able to avoid the defect entirely when $k > 20$. However, such a scenario is rare and can be avoided using the practical ways of improving the CFV described in Section 4.3.

Figure 4.21: Scenario 3: (a) CFV with defect, and (b) path found for $k=20$ (search steps = 139, *total-path-cost* = 135.1606 J, final *dist-to-goal* = 0.0107 rad, and resolution for all joints = 0.5° ,

4.6.4 Other Possible Scenarios

Besides Scenario 3, the search also fails to find a good path for cases where the path is long and relatively winding. However, this can easily be overcome through planning

multiple sequential paths as described earlier in Section 4.5.3. Two examples of scaled-down environments are shown in Figures 4.22-4.23. In Figure 4.22, three paths for three different CFVs to navigate the end-effector around a number of poles/pins are used. In Figure 4.23, two sequential paths are used to move the end-effector to a table to pick up a ball and place it on a conveyor. Figure 4.24 shows an example of a path planned for a full-scale robot and environment. In this full-scale environment, desktop-PC-based visualization was used as opposed to the HMD for miniature environments. However, the RPAR-I methodology was still applied in an immersive AR environment.

Figure 4.22: Planning curvy paths around pins using multiple paths (three sequential paths), for three different CFV s and sets of start-goal configurations

Figure 4.23: General pick-and-place task: (a) generating a CFV, (b) selecting start configuration, (c) selecting intermediate goal configuration, (d) selecting final goal configuration, and (e) final paths found

Figure 4.24: Full-scale general pick-and-place task: (a) generating a CFV, (b) selecting start configuration, (c) selecting goal configuration, and (d) final path found

4.7 Chapter Summary

In this chapter, a RPAR-I methodology for Class I tasks was proposed and compared with existing path planning approaches. The methodology establishes the roles of the humans and computers based on their individual capabilities. Tasks that can be performed easily by the human are the generation of a CFV and selection of start-goal configuration pairs, and tasks that are best handled by the computer are collision verifications, and the automatic generation of smooth and optimal paths. The

generation of paths is handled by a beam search algorithm constrained by a manually generated CFV.

A number of possible scenarios may occur during the application of the RPAR-I methodology. Scenario 1 occurs when the greedy search neighbourhoods do not hit the boundary of the CFV and the solution returned is the greedy path solution. When larger beam sizes, k were used, the path costs improved as shown in Scenario 2. From the experiments, it was discovered that the beam size does not need to be too large to achieve this. This is because the level of improvement decreases as k increases. The search was also shown to be more robust to 'defects' within the CFV for larger values of k , as shown in Scenario 3. The method was successfully applied to a number of other scenarios and was shown to be easy and quick to use. In general, the Class I tasks reported in this chapter were planned within 4-5 minutes for beam sizes, k of 5-10. This is of course also dependent on the PC processing speed and design of the search algorithm.

It is also important to note that the proposed RPAR-I methodology is generic; alternative methods for generating and representing the CFV, algorithms for collision-free verification and search strategies may be explored in the future.

CHAPTERS

RPAR METHODOLOGY FOR CLASS II TASKS

5.1 Chapter Introduction

The primary contribution of Chapter 5 is the proposal of a new robot programming methodology for Class II tasks as defined in Section 1.4, using the RPAR approach. The chapter begins by describing the overall RPAR-II methodology and discussing the path/curve learning problem. In the proposed methodology, the human is involved in a number of activities: (1) obtaining data points for the desired curve to be followed through a number of demonstrations, (2) defining the free space, *i.e.*, CFV, and (3) defining the orientation of the end-effector with respect to the path. These are carried out using visual feedback from AR for evaluation. A proposed Piecewise Linear Parameterization (PLP) algorithm is used to parameterize the collected data points. This is followed by a proposed method based on a combination of Bayesian neural networks and reparameterization, for generating an output curve in the parametric form. The method used for calculating the end-effector transformations along the output curve based on input from the user, is also described. Finally, the results for a number of case studies are presented and discussed.

5.2 Proposed RPAR-II Methodology for Class II Tasks

The objective of the proposed methodology is to generate a collision-free path for a robot, where the end-effector is required to follow a visible but positionally unknown

path/curve, at a certain orientation consistently along the curve. The relationships between the 3D curve to be followed, the base of the robot and/or other objects in the environment may be unknown (non-calibrated environment). In the first step, the user is required to collect data points through a number of repeated and separate demonstrations of the desired curve to be followed. As the data points are collected, visual feedback is provided to aid the evaluation by the user. A similar use of an AR interface is to model virtual curves and surfaces on top of real objects as reported in the Spacedesign project [76] (Figure 5.1). In the second step known as *learning* the curve or *learning* the human intent, the data is processed by a Bayesian neural network to generate an output curve that is close to the desired curve. An illustration of the curve learning problem and proposed approach is shown in Figure 5.2.

Figure 5.1: Constructing a virtual car roof for aesthetic evaluation [76]

Figure 5.2: Curve learning problem and proposed approach

After the output curve is obtained, the user may familiarize himself/herself with the robot and the environment through moving the virtual robot in the environment. In the

next step, a CFV is manually generated at the areas relevant to the curve, using the swept volume of a sphere attached to a probe, as described in Chapters 4-5. The CFV is needed to verify that the end-effector does not collide with any obstacles along the curve. The orientation of the end-effector with respect to the curve is then planned interactively by the user using the AR interface. To verify that the path (in terms of the robot's generalized coordinates) is indeed collision-free, the path is simulated using the calculated transformations along the curve. If the end-effector cannot be guaranteed to be within the CFV at all times, adjustments are made to the end-effector's previous selected orientation until a successful geometric path is found. The overall methodology is shown in Figure 5.3.

Figure 5.3: Proposed RPAR-II methodology

5.3 Output Curve Representation and Parameterization

5.3.1 Parametric Curve Representation

In this research, the output curve which is the learnt desired curve that is unknown, is represented using the parametric form [77],

$$\begin{pmatrix} X(\mathbf{a}, t) \\ Y(\mathbf{a}, t) \\ Z(\mathbf{a}, t) \end{pmatrix} \text{ for a 3D space curve.} \quad (5.1)$$

where X , Y , and Z are 3D coordinates, $\mathbf{a}$ is the vector of model parameters and t is the location parameter (*e.g.*, time).

An illustration of a parametric curve is shown in Figure 5.4.

Figure 5.4: Parametric curve

5.3.2 Data Collection and Parameterization

The data points of the desired curve need to be collected from demonstrations as shown in Figure 5.2. There are two data collection modes that may be used. The first is the Point-by-Point (PbP) mode shown in Figure 5.5(a) where points are collected separately, one-by-one in an ordered way from the start to the end of the curve. The user is able to reject a latest data point if it is deemed bad, *i.e.*, an obvious outlier. The

major disadvantages are that the PbP mode is tedious especially if hundreds of points need to be taken, and the resolution is low. The second mode is tracing as shown in Figure 5.5(b) where the user intuitively drags the tracked probe tip along the curve. Aided with visual feedback provided by AR, the user is able to select and keep only the good demonstrations and reject ones that are bad. If a demonstration is bad (an obvious mismatch of the desired curve or contains many outliers), it can be discarded immediately before proceeding to the next demonstration. Although the tracing mode contains larger variation, it is preferred over the PbP mode for the common user as it is much faster and more convenient; this also makes it more practical. The data points are with reference to the base of the virtual robot on the stationary markers in Figure 5.5. A number of issues regarding data collection using ARToolkit and guidelines on how to handle them will be explained in Sections 5.7.1.1 and 5.7.1.2.

Figure 5.5: Data collection modes: (a) Point-by-Point (PbP) mode, and (b) tracing mode

The collected data points from the tracing mode need to be parameterized as shown by Equation (5.1). This is carried out using either one of two methods. The first method is called *simple parameterization* where each demonstration is parameterized using time, t . The first data point is assigned $t = 0$, and subsequent data points are assigned the times at their respective samplings. The parameter t for each individual demonstration is then normalized to be in the range $[-1, 1]$ (the reason will be given in Section 5.4.1).

If the data points from more than one demonstration are used for processing, the data is grouped and sorted according to the normalized t . The second parameterization method uses a Piecewise Linear Parameterization (PLP) algorithm to parameterize the cloud of data points, according to the closest locations of the points to a piecewise linear approximation of the desired curve. This approximation is interactively provided by the human using the PbP mode. The definitions used for the PLP algorithm are shown in Figure 5.6 and the algorithm is described in Algorithm 5.1. The number of data points is M and the number of piecewise linear lines is N (with $N + 1$ points).

Figure 5.6: Definitions for PLP algorithm

Algorithm 5.1: Proposed PLP algorithm

-
1. **For** $j=1, \dots, N$
 - a) Calculate the unit vectors $\hat{u}_j$ for piecewise linear line j .
 - b) Calculate the length L_j of piecewise linear line j .
 2. **For** $i = 1, \dots, M$
 - a) **For** $j = 1, \dots, N$
 - i) Calculate the vector $v_{i,j}$ between the start point p_j of piecewise linear line j and the data point d_i ,
 - ii) Find the projection of $v_{i,j}$ in the direction of $\hat{u}_j$:
$$a_j = v_{i,j} \cdot \hat{u}_j \quad (5.2)$$
 - iii) Find the point $b_{i,j}$ in the direction of $\hat{u}_j$ where d_i is closest to, In an orthogonal distance sense,
$$b_{i,j} = p_j + a_j \hat{u}_j \quad (5.3)$$
 - iv) Calculate the distances between P_i and $b_{i,j}$, $L_j^{p_j \rightarrow b_{i,j}}$ and, $b_{i,j}$ and p_{j+1} , $L_j^{b_{i,j} \rightarrow p_{j+1}}$.

- v) If $(L_j^{p_j \rightarrow b_{i,j}} \leq L_j)$ and $(L_j^{b_{i,j} \rightarrow p_{j+1}} \leq L_j)$,

d_i potentially belongs to line j .

Calculate distance between $b_{i,j}$ and d_i , $L_j^{b_{i,j} \rightarrow d_i}$.

else

d_i does not belong to line i (e.g., d_i does not belong to line 2 in

Figure 5.6)

- b) If d_i has at least one line that it potentially belongs to:

- i) Find which line has the minimum orthogonal distance to d_i :

Line $W = j$ such that $L_j^{b_{i,j} \rightarrow d_i} \rightarrow \min$.

- ii) For $k = 1, \dots, N + 1$

Calculate distance between d_i and p_k on the piecewise linear curve, $L_k^{d_i \rightarrow p_k}$ and find:

Point $X = k$ such that $L_k^{d_i \rightarrow p_k} \rightarrow \min$.

(This is because even if d_i belongs to line W from 2.b(i), it might actually be closer to a particular point p_X on a line that it does not belong to. Refer to Figure 5.6 where point p_{N+1} is closer to d_i compared to $b_{i,1}$ even though d_i does not belong to line N)

- iii) If $L_W^{b_{i,W} \rightarrow d_i} < L_X^{d_i \rightarrow p_X}$, parameter for d_i :

$$t_i = \frac{L_W^{p_W \rightarrow b_{i,W}} + \sum_{j=1}^{W-1} L_j}{\sum_{j=1}^N L_j} \quad (5.4)$$

else

$$t_i = \frac{\sum_{j=1}^{X-1} L_j}{\sum_{j=1}^N L_j} \quad (5.5)$$

- c) If d_i does not belong to any line, i.e., does not fulfil the criterion for 2.a(v):

- i) For $k = 1, \dots, N + 1$

Calculate distance between d_i and p_k on piecewise linear curve, $L_k^{d_i \rightarrow p_k}$ and find:

Point $Y = k$ such that $L_k^{d_i \rightarrow p_k} \rightarrow \min$.

(If a point does not belong to any line, it is assigned to the closest point on the piecewise linear curve)

- ii) Parameter for d_i :

$$t_i = \frac{\sum_{j=1}^{Y-1} L_j}{\sum_{j=1}^N L_j} \quad (5.6)$$

For the 2D example in Figure 5.7, data points d_1 and d_3 do not belong to any of the three lines according to the definition in 2(a) in Algorithm 5.1, and are assigned by 2(c) to the nearest points on the piecewise linear curve, p_1 and p_2 respectively. Although data point d_4 belongs to L_2 , it is closer to p_4 . Hence, 2(b)(ii) assigns d_4 to p_4 .

Figure 5.7: 2D example of parameterization using PLP algorithm

Lemma 5.1: The complexity of Algorithm 5.1 is $O(MN)$.

Proof For each of the M data points d_i :

1. Lines 2(a)(i-v) calculate the orthogonal distances between the data point and every piecewise linear line vector, assigning the data points to their respective line(s). This takes $O(N)$.
2. Line 2(b)(i) assigns the data point to the correct line by looking for the shortest orthogonal distance. This takes $O(N)$.
3. Line 2(b)(ii) compares the orthogonal distance of the correct line from 2(b)(i) against every point p_j on the piecewise linear curve. This takes $O(N)$.
4. Lines 2(c)(i-ii) calculate the distances between the data point and every point p_j on the piecewise linear curve, and assigns the data point to the closest one. This takes $O(N)$.

Steps (1-4) above are carried out for M data points d_i and this takes $O(MN)$. $\square$

5.3.3 Discussions on Curve Representation and Parameterization

5.3.3.1 Curve Representations

The simplest of 'curves' is a straight line which can be represented as the difference between two vectors in $\mathfrak{R}^3$. A much more difficult problem is the generic space curve in Cartesian coordinates. In literature, there are two other ways of representing a curve other than the parametric form, *i.e.*, the explicit and implicit forms [77-79]. The explicit form is given by Equation (5.7) and the implicit form is the solution to Equation (5.8),

$$\begin{pmatrix} Y = F(\mathbf{a}, X) \\ Z = G(\mathbf{b}, X) \end{pmatrix} \quad \text{for a space curve.} \quad (5.7)$$

$$\begin{pmatrix} F(\mathbf{a}, X, Y, Z) \\ G(\mathbf{b}, X, Y, Z) \end{pmatrix} = \mathbf{0} \quad \text{for a space curve.} \quad (5.8)$$

where X , Y , and Z are 3D coordinates, and $\mathbf{a}$ and $\mathbf{b}$ are vectors of the parameters for functions F and G respectively.

A major disadvantage of the explicit and implicit forms is their coordinate system dependence [77-78]. Coordinate system dependence is unsuitable for fitting geometric curves because the least-squares error changes for different choices of coordinate systems. In other words, shape invariance is not maintained because the model parameters keep changing. The space curve in the implicit form is obtained by intersecting two implicit surfaces. However, this is an inefficient approach because of the use of more parameters than required and it is also unclear how these surfaces are to be selected.

The parametric form does not suffer from the disadvantages of the explicit and implicit forms, and is the most suitable representation for the proposed methodology in Figure 5.1. Not only is it independent of the choice of the coordinate system, it is also able to contain more than just geometric data. For example, position as a function of time is useful for applications such as arc welding. In Figure 5.8, a planar circle is used to illustrate the three different curve forms [77].

Explicit:

$$y = \pm \sqrt{r^2 - x^2}; \quad \text{upper half } (\geq 0).$$

Implicit:

$$y^2 + x^2 = r^2 \quad \text{or} \quad y^2 + x^2 - r^2 = 0$$

Parametric:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} r \cos t \\ r \sin t \end{pmatrix} \quad 0 \leq t \leq 2\pi.$$

Figure 5.8: Different representations of a planar circle [77]

5.3.3.2 Parameterization

The proposed PLP algorithm parameterizes data points based on their orthogonal distances to the piecewise linear approximation provided by the human. This approximation is required because the curve model is unknown. The PLP algorithm is similar to algorithms proposed in [80-81] for fitting problems, where parameterization is also performed using an approximate or 'good' curve. Hoschek proposed an approximating curve based on a particular model that is fit [80]. Gothasby used the spine of the noisy data set as an approximating curve but the method is limited to 2D planar curves [81].

5.4 Curve Learning and Generation

5.4.1 Proposed Bayesian Neural Network and Reparameterization Algorithm

The proposed method for curve learning using the parameterized data is a combination of Bayesian neural networks and reparameterization. Reparameterization is used for fitting problems where the parameters of the data points with respect to the model that is fit, are updated after every least-squares iteration. The primary objective is to perform fitting according to the *orthogonal* distances between the data points and the model that is fit [77, 79-80]. The Bayesian framework can be incorporated into a neural network by assuming that each network weight is a random variable and by modifying the standard back-propagation error equation to obtain a new objective function [82-85],

$$E(\mathbf{W}) = \beta E_D + \alpha E_W \quad (5.9)$$

where $\mathbf{W}$ is a vector of weights (including biases) w_i , E_D is the sum of squared errors, SSE for the network, $E_W = \frac{1}{2} \sum_{\epsilon_i} w_i^2$ is the sum of squared weights, SSW, and β and α are regularization parameters.

The first term in Equation (5.9) is a measure of how much the network reduces the error or fits the data. Overfitting occurs when a curve model that is overly complex, is fit to the data. Although the error is small, the noise in the data is 'captured' by the model and this causes the curve to be more wiggly than necessary. This is known as poor generalization. In order to avoid overfitting the data, the second term in Equation

(5.9) is a weight decay element which makes smaller values of W more desirable. It is the portion that guides the *adjusting* or *learning* of the model that best represents the data. The network balances the fitting and learning activities through the parameters β and a ; this is also known as Bayesian regularization. Before each update of the weights is performed, these parameters can be determined by evaluating the Hessian, H at the most probable weights. Foresee and Hagan proposed a Gauss-Newton approximation to H that is free if a Levenberg-Marquardt (LM) algorithm is used in the neural network [86-87]. The input and output of the training data needs to be normalized between $[-1, 1]$ for the Gauss-Newton approximation in a LM network and the noise or variation in the training data needs to be Gaussian for good performance.

A method, that combines the reparameterization concept from fitting and a Bayesian neural network with a single hidden unit layer with LM implementation, is proposed. The network structure is shown in Figure 5.8 where the input is the parameter t , the model/neural network is a , and the outputs are the 3D coordinates of the curve. The commonly used tan-sigmoid function which 'squashes' its output to be between $[-1, 1]$ is used for the hidden layer whereas the linear function which allows its outputs to take on any value, is used for the output layer. These transfer functions have been successfully implemented for a variety of function approximation problems [88]. The initial curve learning by the network is not based on the orthogonal distances, even though a close approximation may be obtained using the PLP algorithm for *initial parameterization*. Therefore, this research proposes the inclusion of reparameterization using the output curve from the Bayesian neural network as a good initial guess. Curve learning is still performed because the complexity of the neural network is further adjusted using Bayesian regularization, after each reparameterization step.

Convergence is achieved if the objective function of Equation (5.9) does not change significantly, continuously over a particular number of iterations. Alternatively, an early stopping strategy can be adopted if the generalization of the solution deteriorates with each new iteration. In Figure 5.9, the final values taken on by the effective weights of the neural network are the model a of the learnt curve. The proposed algorithm is described in Algorithm 5.2. The variable res is the increment value ($0 < res < 2$) for the input which is in the range of $[-1, 1]$ and determines the resolution of the output curve.

Figure 5.9: Proposed Bayesian neural network structure with one hidden layer

Algorithm 5.2: Proposed Bayesian neural network and reparameterization algorithm

-
1. Run Bayesian neural network with initial parameterization using PLP algorithm, until convergence.
 2. Simulate network for inputs $[-1: res: 1]$ to obtain initial output curve.
 3. Repeat
 4. Perform reparameterization.
 5. Run current Bayesian neural network from previous iteration (do not reinitialize network parameters), for one iteration with new input-output set.
 6. Simulate network for inputs $[-1: res: 1]$ to obtain output curve.
 7. Until convergence.
-

5.4.2 Reparameterization and Error Definitions

The reparameterization step or line 4 in Algorithm 5.2 is performed using *point-to-point* matching [89]. If a particular curve is given as a set of parameterized points, the orthogonal distances between the data points and curve points can be found. The shortest Euclidean distance between a particular data point $\mathbf{X}_i = [x, y, z]_i$ and a curve F that consists of p points, $\mathbf{X}_j^F$ is defined as,

$$l_i = \min_{j \in \{1, \dots, p\}} \|\mathbf{X}_i - \mathbf{X}_j^F\| \quad (5.10)$$

After each neural network run, the parameters are updated based on Equation (5.10) which determines the nearest location of the data points on the current output curve.

Lemma 5.2: For m data points and p curve points, reparameterization takes $O(mp)$.

Proof For every data point, the orthogonal distances between the point and all p curve points are calculated to find the minimum and this takes $O(mp)$. The update takes $O(1)$ as it is merely an assignment of the parameter of the identified curve point where the distance is shortest, to the data point. Hence, reparameterization takes $O(mp)$. $\square$

In this research, the performance of a number of curve learning methods based on parameterization, Bayesian neural networks and reparameterization, were compared. The final output curve was compared to a *known* desired curve to gauge how well it matches the desired curve. This is based on positional information and not on comparing data points at a particular parameter t for the two different models under

consideration. If an output curve G that consists of n parameterized points is similar in shape and resolution to a desired curve F that consists of p parameterized points, a simple measure of goodness between the two can be defined based on Equation (5.11).

For point $\mathbf{X}_i^G$ on the output curve, the error between G and F is defined as,

$$e_i = \min_{j \in (1, \dots, p)} \|\mathbf{X}_i^G - \mathbf{X}_j^F\|; \quad \in i = 1, \dots, n \quad (5.11)$$

The average error, maximum error and variance of error are defined as,

$$\bar{e} = \frac{1}{n} \sum_{i=1}^n e_i \quad (5.12)$$

$$e_{\max} = \max_{i \in (1, \dots, n)} e_i \quad (5.13)$$

$$\sigma^2 = \frac{1}{n-1} \sum_{i=1}^n (e_i - \bar{e})^2 \quad (5.14)$$

where the average error represents the overall 'closeness' of the output curve to the desired curve, the maximum error denotes how well the output curve matches the desired curve at specific portions of the curve, and the error variance indicates the level of unwanted 'wiggling' that exists in the output curve compared to the desired one.

5.4.3 Discussions on Curve Learning and Generation

5.4.3.1 Learning, Fitting and Modelling of Curves

The modelling and fitting of curves are different from the learning problem addressed in this research. Modelling is the representation of a curve using mathematical models and is used extensively in CAD. The curve fitting problem, which is similar to learning

as compared to modelling, is concerned with finding the optimal parameter values of a *pre-determined* model where *every* parameter is used to minimize the sum of squared errors of a set of data. However, curve learning refers to the case where the model type, and/or the complexity of a particular model, *i.e.*, appropriate number/type of parameters used, is unknown. A mechanism to *learn* what the user perceives as the desired curve needs to be included instead of requiring the user to select a suitable model. This is important to make the methodology as intuitive and user-friendly as possible. Even if an appropriate model type can be identified for fitting, say a spline, the number of control points, locations of knots and orders of basis functions need to be pre-determined. If composite curves exist such as those with linear sections and curvy sections, the basis functions need to be of multiple order types at specific sections of the curve. Standard least-squares fitting cannot automatically address these curve learning issues. Hence, the proposed curve learning strategy has to be able to balance between minimizing the sum of squared errors and *overlearning* the curve model (or overfitting). A comparison between modelling, fitting and learning can be found in Table 5.1.

Table 5.1: Comparison between modelling, fitting and learning

	Modelling	Fitting	Learning
Model	Known	Known	Unknown
Task	To create mathematical models for representing curves and lines.	To find the appropriate parameter values for a pre-determined model based on the given data by minimizing error (<i>e.g.</i> , number of control points for splines).	To learn an appropriate model based on the given data. This involves balancing between minimizing error and obtaining good generalization.
Examples of applications	Representing CAD models for design, analysis and visualization.	Processing tool for CMM, pattern recognition/template matching	Learning paths/curves based on data from human demonstrations (PbD)

For the standard curve fitting problem, the least-squares method is used to minimize errors based on two types of distances, algebraic and geometric. The algebraic distance is used mainly for implicit representations. However, in physical terms, the algebraic distance is both ambiguous and dependent upon the choice of the coordinate system. It is generally accepted that the use of the algebraic distance is inaccurate for metrology fitting applications [77, 90]. Metrology fitting involves the fitting of curves and surfaces for applications such as pattern recognition, template matching and reverse engineering, using the data obtained from measurement equipment such as Coordinate Measurement Machines (CMM) and laser scanners. Unlike the algebraic distance, the *geometric distance* is based on the fact that a measurement point is the most probable observation of a point on the curve closest to the measurement point [91]. This measure is used for Orthogonal Distance Fitting (ODF) which is the standard measure of the quality of a fit as described in ISO 10360-6 [77]. For this reason, ODF is applied to the proposed learning algorithm, Algorithm 5.2, through the inclusion of reparameterization. The sum of squared errors for n data points is defined as,

$$SSE = \sum_{i=1}^n \|\mathbf{X}_i - \mathbf{X}_i^G\|^2 \quad (5.15)$$

where the norm is the orthogonal distance (shortest Euclidean distance) between a measurement point $\mathbf{X}_i = [x, y, z]_i$ and its corresponding point on the model of the output curve G , $\{\mathbf{X}_i^G\}_{i=1}^n \in G(\mathbf{a}, \mathbf{X})$.

5.4.3.2 Neural Networks

The design of the proposed curve learning method is based on the popular use of feedforward neural networks for solving curve fitting and learning problems [92-93].

Multilayered feedforward neural networks are parameterized non-linear models that are known as *universal approximators* because of their ability to learn or approximate any function [94]. The basic unit in a neural network is a single perceptron which produces an output y that is a continuous function of a weight vector w , a bias b and an input vector x (Figure 5.10(a)). The transfer function f can take on many forms depending on the application. An example is the commonly used logistics function which produces an output between 0 and 1,

$$y_o = f(w \cdot x + b)$$

$$= \frac{1}{1 + e^{-(w \cdot x + b)}} \quad (5.16)$$

Figure 5.10: (a) Single perceptron, and (b) multilayered feedforward neural network

Figure 5.10(b) shows how combinations of single perceptrons are able to construct multilayered feedforward neural networks. The layers in between the inputs and outputs are called the hidden layers. In the learning process, the parameters (weights and biases) are first initialized and then iteratively updated by minimizing the sum of squared errors, SSE,

$$E_D = \frac{1}{2} \sum_{i=1}^{n_t} (D_i - y_i^t)^2 \quad (5.17)$$

where D_i is the desired output and y_i^L is the output i for layer L (output layer).

The classic back-propagation algorithm uses gradient methods to fine-tune the weights and biases to minimize the overall error (management by objective) [95]. The following back-propagation derivations are adapted from [95]. In a multilayered network, there are L layers of weights and biases, and $L + 1$ neuron layers. The activity of neuron i at layer l , u_i^l in a forward pass is given by,

$$u_i^l = \sum_{j=1}^{n_{(l-1)}} w_{ij}^l h_j^{(l-1)} + b_i^l; \quad h_i^l = f(u_i^l); \quad \in l = 1, \dots, L \quad (5.18)$$

where n_l is the number of neurons for layer l , w_{ij}^l is weight i for layer l connected to neuron j from layer $l-1$, $h_j^{(l-1)}$ is the output of neuron j from layer $l-1$, b_i^l is bias i for layer l , and f is the transfer function. For the output layer, $h = y$ and $l = L$.

The updating of the weights and biases is performed using a backward pass. The sensitivities of each neuron output to the objective function of Equation (5.17) are found by starting at layer L and moving 'backwards' or left till all the sensitivities are obtained using the following equations,

$$s_i^L = f'(u_i^L) \frac{\partial E_D}{\partial y_i^L} = f'(u_i^L)(y_i^L - d_i) \quad \text{for the output layer } L. \quad (5.19)$$

$$s_j^{(l-1)} = f'(u_j^{(l-1)}) \sum_{i=1}^{n_l} s_i^l w_{ij}^l; \quad l = L, \dots, 2 \quad (5.20)$$

The updates on the weights and biases are then performed using the following equations,

$$\Delta w_{ij}^l = \lambda s_i^l h_j^{(l-1)} \quad (5.21)$$

$$\Delta b_i^l = \lambda s_i^l \quad (5.22)$$

where $\lambda > 0$ is a fixed learning rate.

The training of the network stops when the objective function is at a desired level (depending on whether the function is to be maximized or minimized) continuously for a certain number of iterations.

One frequently encountered problem is the design of the hidden units in the network which represent the dof of the network. The standard back-propagation method suffers from a lack-of-fit if too few hidden units are used and an overfit if too many units are used. To deal with this, MacKay introduced a Bayesian framework for neural networks to regulate the number of parameters (hence complexity) used by the network and thus prevent overfitting [82-84]. This approach works very well for empirical regression problems which are similar to the curve learning problem.

5.4.3.3 Bayesian Framework

MacKay proved that a Bayesian framework embodies Occam's razor:

"Occam's razor is the principle that states a preference for simple theories. If several explanations are compatible with a set of observations, Occam's razor advises us to buy the least complex explanation. " [85]

Occam's razor is advocated for mathematical elegance and its empirical success [85]. The principle is embodied in the Bayesian framework using two levels of inferences of data modelling. The rest of this section is a summary of the derivations reported in [82-85] and are presented for continuity. The first level of inference involves the assumption that a particular model that is fit to the data is correct. This is exactly the same problem as fitting a pre-determined function to a set of noisy data. In the context of neural networks, each model, M has weights/parameters $\mathbf{W}$ (refers to both weights w and biases b) where each weight is described by a *prior* probability distribution $P(\mathbf{W} | M)$ that states the likelihood of the model taking on a particular set of weights. Given a set of data D , the *posterior probability* of $\mathbf{W}$ is defined as [82-85],

$$\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}}$$

$$P(\mathbf{W} | D, M) = \frac{P(D | \mathbf{W}, M)P(\mathbf{W} | M)}{P(D | M)} \quad (5.23)$$

The evidence or normalizing constant is ignored at the first level of inference but is useful for the second level. Using gradient methods to find the maximum of Equation (5.23), the most likely weights $\mathbf{W}_{MP}$ can be found. The posterior can be approximated as a Gaussian with mean $\mathbf{W}_{MP}$ and $\Delta\mathbf{W} = \mathbf{W} - \mathbf{W}_{MP}$,

$$P(\mathbf{W} | D, M) \simeq P(\mathbf{W}_{MP} | D, M) \exp\left(-\frac{1}{2} \Delta\mathbf{W}^T \mathbf{H} \Delta\mathbf{W}\right) \quad (5.24)$$

where $\mathbf{H}$ is the Hessian for evaluating the curvature of the posterior distribution at $\mathbf{W}_{MP}$.

Figure 5.11 illustrates how for a single weight $\mathbf{W}$, the posterior probability distribution is an updated version of the prior distribution *after* the data is presented. The range of possible values that can be taken by $\mathbf{W}$ becomes narrower, thus exhibiting a preference for simpler models, *i.e.*, evidence of Occam's razor. The Occam factor is the ratio between the standard deviations of the two distributions, $\frac{\sigma_{\mathbf{W}|D}}{\sigma_{\mathbf{W}}}$.

Figure 5.11: Evidence of Occam's razor [82-85]

The second level of inference involves two parts: (1) finding the appropriate complexity level for a given model type, *i.e.*, the best sub-model where each sub-model consists of the effective parameters used by the network at a particular iteration, and (2) finding the preferred model type after the best sub-models for each model type are determined. Here, model type refers to different network architectures, noise models and so forth. For part (1), the sub-models' complexities are regulated through the parameters β and a from Equation (5.9). For part (2), the constant $P(D|M)$ may be used to compare between different model types after the final sub-models for each are obtained. However, part (2) is not considered in this research as a network structure based on [88] had been proposed in Section 5.4.1. For each model type,

$$P(M | D) \propto P(D | M)P(M) \quad (5.25)$$

Assuming that $P(M)$ is a uniform distribution (to be 'fair' to all i models before the data is presented) the most plausible model is the one that maximizes $P(D|M)$.

In the proposed Bayesian neural network approach for curve learning, the network begins by fitting the data to the most complex sub-model which uses every single free parameter or dof of the network, *i.e.*, every weight and bias. This sub-model is then continually 'trimmed' or 'adjusted' to obtain a better sub-model after each network run, by regulating the complexity of the network as explained in Section 5.4.1. A detailed explanation of the Bayesian neural networks implementation is found in Appendix C.

5.4.3.4 Reparameterization

Reparameterization is a concept that is commonly used for fitting problems. In the proposed Algorithm 5.2, a simple method is used for reparameterization based on Equation (5.10). As mentioned earlier, this is to ensure that the Bayesian neural network learns the curve based on the orthogonal distances of the data points to the current output curve. In related work on fitting, Sung developed reparameterization algorithms for fitting implicit and parametric curves and surfaces using the *total* and *variable separation* methods [77]. The total method estimates the model parameters and orthogonal distances simultaneously whereas the variable separation method updates the orthogonal distances *after* the model parameters are updated for every least-squares iteration, until convergence. The orthogonal distances are found using the Newton, Lagrangian Multipliers and LM [87] methods and the updates on the parameters are carried out using gradient methods.

5.5 Collision-Free Volume for RPAR-II

The RPAR-II methodology also involves the generation of a CFV as described in Chapter 4. However, for Class II tasks, the CFV is generated *specifically* at areas along the output curve where obstacles might be present. When the simulation is performed, the *CFV Check* will verify if the end-effector is collision-free along the entire curve.

5.6 End-Effector Orientation Planning

There are two cases that will be presented, the first is the simple case using linear interpolation and the second is the complex case using generic space curves. The RPAR-II methodology utilizes the ability of the human to evaluate the surface on which the curve lies, and its surroundings. Firstly, the user intuitively defines a unique curve coordinate system (and the corresponding end-effector transformation) at the start of the curve using the AR interface. The remaining transformations that maintain the relationship between the end-effector and the curve, along the entire curve can then be calculated. Finally, these transformations are used by the IK module described in Section 3.5 to generate the robot path.

5.6.1 Simple Case using Linear Interpolation

For the simple case, the desired 'curve' is a straight line and the orientation of the end-effector is constant along the entire line (Figure 5.12). The user directly demonstrates the desired orientation at the start of the line and at the same time, selects start point P1. This is followed by the selection of end point P2. The transformations from the

base to the wrist of the robot, for the line that consists of $i=1,...,n$ ordered points, are then defined by the 4×4 homogenous transformation matrix,

$${}^B_{W_i}M = \left[\begin{array}{c|c} {}^B_{W_i}R & {}^B_{W_i}T \\ \hline \mathbf{0} & 1 \end{array} \right]; \quad i = 1, \dots, n \quad (5.26)$$

where ${}^B_{W_i}R$ is the 3×3 selected end-effector orientation matrix, $\mathbf{0}$ is a 1×3 vector of zeros, and ${}^B_{W_i}T$ is the translation from the robot base B to the wrist position corresponding to line point i , W_i defined by,

$${}^B_{W_i}T = \begin{bmatrix} {}^B_{W_i}x \\ {}^B_{W_i}y \\ {}^B_{W_i}z \\ 1 \end{bmatrix} \quad (5.27)$$

The orientations at the wrist are constant along the entire line,

$${}^B_{W_1}R = {}^B_{W_2}R = \dots = {}^B_{W_{n-1}}R = {}^B_{W_n}R \quad (5.28)$$

$${}^B_{W_1}T \neq {}^B_{W_2}T \neq \dots \neq {}^B_{W_{n-1}}T \neq {}^B_{W_n}T \quad (5.29)$$

Equation (5.26) is derived by first calculating the vector between the start and end points of the line, $\vec{L}$,

$$\vec{L} = \begin{bmatrix} {}^B_nx - {}^B_1x \\ {}^B_ny - {}^B_1y \\ {}^B_nz - {}^B_1z \end{bmatrix} \quad (5.30)$$

where $|\vec{L}| = \sqrt{({}^B_nx - {}^B_1x)^2 + ({}^B_ny - {}^B_1y)^2 + ({}^B_nz - {}^B_1z)^2}$.

The points of the line will be on the vector $\vec{L}$ and the resolution can be controlled using a factor q . The total number of points, n is determined by,

$$n = \begin{cases} \lfloor |\vec{L}|/q \rfloor + 2 & \text{if } |\vec{L}| \text{ is a non-integer} \\ |\vec{L}|/q + 1 & \text{if } |\vec{L}| \text{ is an integer} \end{cases} \quad (5.31)$$

where q is a positive real number in *mm/number-of-points*.

The coordinates of all the points on the line are given by,

$${}^B_iT = \begin{bmatrix} {}^B_1x + (i-1)({}^B_nx - {}^B_1x)/(n-1) \\ {}^B_1y + (i-1)({}^B_ny - {}^B_1y)/(n-1) \\ {}^B_1z + (i-1)({}^B_nz - {}^B_1z)/(n-1) \\ 1 \end{bmatrix}; \quad i = 1, \dots, n \quad (5.32)$$

After the orientation is defined, the coordinate systems on the line are translated to the wrist of the robot,

$$\begin{aligned} {}^B_{w_i}M &= {}^B_iM {}^i_{w_i}M \\ &= \begin{bmatrix} {}^B_iR & {}^B_iT \\ \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{I} & {}^i_{w_i}T \\ \mathbf{0} & \mathbf{1} \end{bmatrix}; \quad i = 1, \dots, n \end{aligned} \quad (5.33)$$

where the selected end-effector orientation is ${}^B_iR = {}^B_{w_i}R$, and $\mathbf{I}$ is a 3 x 3 identity matrix.

Figure 5.12: Simple case using linear interpolation and unconstrained end-effector orientation selection

5.6.2 Complex Case using Space Curve

For the complex case using the output curve from the Bayesian neural network, the objective is to move the end-effector along the output curve while maintaining a certain orientation with respect to the *local part surface*. The surfaces on both sides of the curve can be defined by tracing rough curves that are slightly offset and generally parallel to the output curve as shown in Figure 5.13. For each point on the output curve, the two closest points on both sides (perpendicular to the output curve) would define the local surface planes. Different options may be available with this information; for example, a single plane at the intersection of planes 1 and 2 in Figure 5.13, that is tangent to both planes, may be used to define a local curve coordinate system. This process is then repeated for the entire output curve. Finally, a pre-defined relationship between the curve coordinate systems and the end-effector (this relationship is dependent on the application, *e.g.*, required torch angle for arc welding) can be used to calculate the necessary transformations for inverse kinematics.

Figure 5.13: Local surface planes defined by tracing rough curves on both sides of the output curve

In this thesis, a simpler method is used where the user is only required to define the orientation of the end-effector *at the start* of the output curve in a controlled and constrained manner, without having to trace rough curves on both sides. This method is based on the assumption that the surface with respect to the curve at its starting point, does not change drastically along the entire curve. The definition of the curve coordinate system is performed by first setting one of the axes of the Euclidean 3D system in the direction of the curve, at the start of the curve. This is known as the curve axis in Figure 5.14 which is the curve tangent at the start point. There are two planes that exist; the first is the plane normal to the curve axis, *i.e.*, *normal plane* and the second is the plane formed by the curve axis and surface axis, *i.e.*, *surface plane*. Therefore, the two axes corresponding to the curve axis, that are coplanar and on the normal plane, are the surface axis and a third axis. The surface axis is determined by a user-defined surface point. As it is impractical to require the user to select this point *exactly* on the normal plane, a tracked probe tip for general interaction is projected onto the normal plane, defining the surface axis in Figure 5.14. The method used for point projection can be found in Appendix D. Using the curve and surface axes, the third axis can then be calculated uniquely. As the user moves the user-defined surface point, the surface plane changes while the normal plane is fixed.

The orientation of the end-effector can be calculated using a pre-defined transformation from the curve coordinate system determined by the user, to the coordinate system of the wrist of the robot. As mentioned earlier, this transformation is dependent on the application. For example, the end-effector is required to be parallel to the curve and at a fixed angle $0 < \gamma < 90^\circ$ to the surface plane. Hence, the transformation required is a translation to the wrist of the end-effector, followed by a

positive rotation of $90^\circ - \gamma$ about the new axis of the translated coordinate system that is parallel to the surface axis in Figure 5.14. This transformation aligns the third axis to the direction of the principal axis of the end-effector, *EE direction*, which is displayed to the user. By visually evaluating the obstacles and surface on which the curve lies, the user intuitively selects a suitable end-effector direction. The user-defined curve coordinate system is then traversed along the entire curve using Algorithm 5.3.

Figure 5.14: Defining the curve coordinate system at its start (and the corresponding end-effector direction) using the user-defined surface point

Figure 5.15: Coordinate systems involved for calculating end-effector transformations

The transformations needed by the IK module to generate the final path for the n-dof robot, are the transformations from the base to the wrist coordinate systems, ${}^B M_{w_i}$. Without loss of generality, the conventions in Figure 5.15 are proposed for Algorithm 5.3:

1. The output curve from the neural network consists of $i=1,...,n$ parameterized points and the distances between points depend on the resolution of the output curve, *i.e.*, the variable *res* in Algorithm 5.2.
2. The Y_i axis is set to be the curve segment between points i and $i+1$ for $i < n$.
3. The corresponding X_i and Z_i axes are based on the *right-hand convention*.

Algorithm 5.3: Algorithm for calculating end-effector transformations along the output curve

-
1. For $i=1, ..., n-1$
 Calculate the 3×1 unit vectors $\hat{Y}_i$ for each point i to its next point $i+1$.
 2. Set the surface axis to X_1 and calculate its 3×1 unit vector $\hat{X}_1$.
 3. Calculate the corresponding 3×1 unit vector $\hat{Z}_1$, using the cross-product rule for the *right-hand convention*. The entire curve coordinate system changes when the user changes the user-defined surface point and hence, the surface axis.
 4. After the user selects/defines the desired curve coordinate system at the start of the curve (curve coordinate system 1), the 4×4 homogenous transformation from the base to curve coordinate system 1 is defined as,

$${}^B M_1 = \left[\begin{array}{ccc|c} \hat{X}_1 & \hat{Y}_1 & \hat{Z}_1 & {}^B T_1 \\ \hline \mathbf{0} & & & 1 \end{array} \right] \quad (5.34)$$

where ${}^B T_1 = \begin{bmatrix} {}^B x \\ {}^B y \\ {}^B z \\ 1 \end{bmatrix}$ and $\mathbf{0}$ is a 1×3 vector of zeros.

5. The transformation from the base to the wrist coordinate system corresponding to curve coordinate system 1 is defined as,

$${}^B M_{w_1} = {}^B M_1 {}^1 M_{w_1} \quad (5.35)$$

where ${}^1 M_{w_1}$ is a pre-defined transformation from curve coordinate system 1 to wrist coordinate system 1. This transformation aligns $\hat{Z}_1$ to the direction of the

principal axis of the end-effector, $\hat{Z}_{w1}$. To maintain consistency along the curve, ${}_{w1}^1M = {}_{w2}^2M = \dots = {}_{wn}^nM$.

6. Traversing the curve; the transformation from curve coordinate system i to $i+1$ can be determined using two Euler angles. Each transformation ${}_{i+1}^iM$ will align $\hat{Y}_i$ to $\hat{Y}_{i+1}$. To obtain all the transformations along the curve:

a) **For** $i=1, \dots, n-1$

- i) Obtain the coordinates of point $i+1$ with respect to curve coordinate system i . As the coordinates of point $i+1$ with respect to the base, ${}_{i+1}^BT$ are available from the output curve,

$${}_{i+1}^iT = {}_i^BM^{-1} {}_{i+1}^BT \quad (5.36)$$

$$\text{where } {}_{i+1}^iT = \begin{bmatrix} {}_{i+1}^ix \\ {}_{i+1}^iy \\ {}_{i+1}^iz \\ 1 \end{bmatrix}$$

- ii) The transformation from curve coordinate system i to $i+1$ is a translation from point i to $i+1$, followed by a rotation about the $\hat{Z}_{i+1}$ axis by ϕ_i , and a rotation about the $\hat{X}_{i+1}$ axis by α_i

$${}_{i+1}^iM = \begin{bmatrix} \cos \phi_i & -\sin \phi_i \cos \alpha_i & \sin \phi_i \sin \alpha_i & \vdots \\ \sin \phi_i & \cos \phi_i \cos \alpha_i & -\cos \phi_i \sin \alpha_i & \vdots \\ 0 & \sin \alpha_i & \cos \alpha_i & \vdots \\ & 0 & & \vdots \end{bmatrix} {}_{i+1}^iT \quad (5.37)$$

$$\text{where } \phi_i = -\text{atan2}({}_{i+1}^ix, {}_{i+1}^iy), \quad (5.38)$$

$$\alpha_i = \text{atan2}({}_{i+1}^iz, \sqrt{{}_{i+1}^ix^2 + {}_{i+1}^iy^2}) \quad (5.39)$$

- iii) The last transformation, ${}_{n-1}^nM$ is merely a translation along $\hat{Y}_{n-1}$ from point $n-1$ to point n because there is no $\hat{Y}_n$ to be calculated, i.e., point $n+1$ does not exist. Therefore, $\hat{Y}_n = \hat{Y}_{n-1}$ and $\phi_{n-1} = \alpha_{n-1} = 0$.

- iv) The transformation from the base to coordinate system $i+1$ is,

$${}_{i+1}^BM = {}_i^BM {}_{i+1}^iM \quad (5.40)$$

- v) The transformation from the base to wrist coordinate system $i+1$ is,

$${}_{wi+1}^BM = {}_{i+1}^BM {}_{wi+1}^{i+1}M \quad (5.41)$$

Lemma 5.3: The complexity of Algorithm 5.3 is $O(n)$.

Proof Lines 1 and 6 perform $O(1)$ calculations for $O(n)$ curve points. Lines (2-5) are all $O(1)$ calculations. Hence, the complexity of Algorithm 5.3 is $O(n)$.

5.7 Results and Discussions for Class II Tasks

There are two major parts in this section, curve learning and end-effector orientation planning. For curve learning, experiments are carried out to study the performance of four different methods used to generate the output curve. The limitations of the ARToolkit tracking system are first addressed and the experimental procedure described. This is followed by detailed discussions on the observations for three curve learning case studies. The end-effector orientation planning activities are shown through the demonstration of the overall RPAR-II methodology. Four case studies are presented, the simple case study using linear interpolation and the complex case studies using the three earlier curve learning examples.

5.7.1 Curve Learning Results and Discussions

The objectives of the comparative studies are, to compare between the performance of the four possible methods described in Table 5.2 in producing an approximate output curve of a *known* desired curve (in practice, the desired curve is unknown), and to observe the effects of the number of demonstrations used on the quality of the output curve. Methods A and B do not use reparameterization, where A is the base case using simple parameterization and B uses the PLP algorithm. Method C uses simple parameterization and reparameterization, and 0 is the proposed method described in Algorithm 5.2 using the PLP algorithm and reparameterization. The Bayesian neural network used is a 1-10-3 network with 10 hidden units or a total of 52 available parameters (weights and biases). The number of parameters is sufficiently large for the curve learning case studies. The implementations were carried out using MATLAB.

Table 5.2: Four methods used for curve learning

Method	PLP-Algorithm Parameterization	Reparameterization	Remarks
A	-	-	Initial simple parameterization before running the network.
B	✓	-	Initial parameterization using PLP algorithm before running the network.
C	-	✓	Initial simple parameterization before running the network. The output is used as a good guess for subsequent reparameterization based on orthogonal distance fitting.
D	✓	✓	Initial parameterization using PLP algorithm before running the network. The output is used as a good guess for subsequent reparameterization based on orthogonal distance fitting.

5.7.1.1 Limitations of ARToolkit

There are two major limitations of ARToolkit that affect the process of acquiring 3D data points. The first is the inaccuracy, in the estimated Z direction which is very much dependent on the orientation and/or position of the moving marker ⁵(the accuracy in the X and Y directions are significantly higher). Therefore, if the probe tip is pointing at the same point for two instances where the probe marker is at two different orientations and/or positions, the estimated Z values may be different for both. To handle this, the desired curve against which the output curves will be compared, is obtained using the PbP mode and Method B or D, and the orientations of the marker

⁵ Although the absolute accuracy in the Z direction is not high, ARToolkit renders the data points and curves more accurately *relative* to the tracking system and view of the user.

along different sections of the curve are noted. When the actual demonstrations are performed later, similar orientations (as when producing the desired curve) are used to ensure that the data is indeed trying to replicate the Z values of the desired curve. Although these values do not have high absolute accuracy, it is important that they are generally consistent for the desired curve and every demonstration. The second limitation is the sensitivity of ARToolkit to the orientation of the moving marker, which causes large noise in the Z direction. Although much larger than in the X and Y directions, this noise can be reduced by avoiding unnecessary jittering of the moving marker. However, these two limitations do not severely affect the goal of the research which is to compare between the performances of different methods and not on quantifying the *absolute* accuracy of the tracking system. Sophisticated tracking systems do not suffer from the limitations above, and can be employed in the future.

5.7.1.2 Experimental Procedure

1. The output curves are compared against a *known* desired curve obtained as follows:
 - i) Draw *real* desired curve on the real object concerned using an erasable medium.
 - ii) Using the AR interface, carefully record curve points one-by-one (PbP Mode) until a satisfactory set is found. Note the orientations of the moving marker along different sections of the curve.
 - iii) Obtain a piecewise linear approximation of the drawn curve using the PbP mode and parameterize the set of points using the PLP algorithm.
 - iv) Execute Method 8 or D and overlay output curve on drawn curve for evaluation.
 - v) Manually adjust (redraw affected segments) drawn curve until it is *exactly* aligned with the output curve. This is the *known* desired curve.
 - vi) Repeat steps (i-v) until a final satisfactory curve is obtained.

2. Using the AR interface, trace the desired curve without any virtual curve overlay. Select and keep ten good demonstrations as explained in Section 5.3.2.
3. Obtain a new piecewise linear approximation of the desired curve using the PbP mode. This is an approximation of the *known* desired curve from step 1.
4. Execute the following for Methods A-D in Table 5.2:
 - i) Obtain various data sets using data from ten independent demonstrations. For ten demonstrations, there is only one data set using all the demonstrations. For one to nine demonstrations, randomly pick three different data sets for each number of demonstrations, *e.g.*, for two demonstrations: Set 1: Demo 1, Demo 8, Set 2: Demo 5, Demo 10, and Set 3: Demo 4, Demo 8. Parameterize data using simple parameterization or the PLP algorithm.
 - ii) Normalize the inputs and outputs to be between $[-1, 1]$ which is required for the Bayesian neural network with LM implementation.
 - iii) Execute the methods and convert the resulting data into unnormalized units.
 - iv) Calculate errors and average the results of the three sets (average error, maximum error and variance of error from Equations (5.12-5.14)).

5.7.1.3 Curve Learning Case Studies

Three case studies are used; the first is the miniature planar 'AR' curve (Figure 5.16), the second is a miniature non-planar curve on a mouse (Figure 5.17), and the third is a full-scale non-planar curve on a piece of bent sheet metal (Figure 5.18). The planar case study is designed to minimize noise in the Z direction by maintaining the orientation of the moving marker as constant as possible for each demonstration. This is relatively easy because the curve is on a flat plane. For the non-planar cases, the

orientations of the moving marker when performing the demonstrations are kept generally consistent" as when generating the desired curve. The virtual cubes on the stationary markers in Figures 5.16-5.18 represent the base of the robot. In general, each demonstration would only take a few seconds depending on how fast the user traces the desired curve.

Figure 5.16: Case Study 1- miniature planar 'AR' curve: (a) desired curve and robot base, (b) real drawn curve, (c) piecewise linear approximation of desired curve obtained from Step 3, and (d) desired curve using piecewise linear approximation from Step 1(iii)

⁶ Note that consistent here is different from constant for the planar curve. For the non-planar cases, a number of orientations might be used, each for different sections of the curve.

Figure 5.17: Case Study 2- miniature non-planar curve on mouse: (a) desired curve and robot base, (b) real drawn curve, (c) piecewise linear approximation of desired curve obtained from Step 3, and (d) desired curve using piecewise linear approximation from Step 1(iii)

Figure 5.18: Case Study 3- full-scale non-planar curve on bent sheet metal: (a) desired curve and robot base, (b) real drawn curve, (c) piecewise linear approximation of desired curve obtained from Step 3, and (d) desired curve using piecewise linear approximation from Step 1(iii)

5.7.1.4 Effects of the PLP Algorithm, Reparameterization and ARToolkit Limitations

Parameterization significantly affects the performance of the neural network. Figures 5.19-5.21 show the output curves obtained using Methods A-D for ten demonstrations. When simple parameterization is used, the parameters of the data points along the curve are dependent on the speed with which the user traces the desired curve. This causes large variation when the demonstration data are grouped together as shown in Figure 5.19(a) for Case Study 1, Figure 5.20(a) for Case Study 2 and Figure 5.21(a) for Case Study 3. The PLP algorithm parameterizes data points according to a piecewise linear approximation of the curve. Figure 5.22 shows an example of 3D data points parameterized using this method. The effects of using the PLP algorithm can be observed in Figures 5.19(b), 5.20(b) and 5.21(b) where the data points are more compact as compared to Figures 5.19(a), 5.20(a) and 5.21(a) respectively. This is because significant speed differences from demonstration to demonstration causes large variation for simple parameterization. In contrast, the PLP algorithm parameterizes the data points based purely on position and not speed. Another observation is that reparameterization helps the network to 'recover' from the initial inaccurate simple parameterization. This can be observed in Figures 5.19(c), 5.20(c) and 5.21(c) for Method C (simple parameterization) where the variation is comparable to Methods B and D (parameterization using the PLP algorithm).

The limitations of ARToolkit also affect the output curves. The inaccuracy of ARToolkit in the estimated Z direction can be observed in Figures 5.19-5.21 (and also in Figures 5.16-5.17). Although the actual curve in Case Study 1 is planar horizontally,

the small changes in orientation and/or position of the moving marker causes the output curve values to be non-planar, resembling an 'M' pattern, even though the orientation was kept as constant as possible during the demonstrations. Another example is the non-planar Case Study 2. The initial part of the actual curve on the mouse should be moving upwards instead of downwards as shown in the output curve, for the same reasons as Case Study 1. In these two examples, 'actual' refers to the *physical curves*, and not the known desired curves which would also possess the characteristics of the output curves (refer to Figures 5.16-5.18). In addition, it can be observed that the noise and variation in the Z direction is largest among the three directions; the planar case with the least noise in the Z direction. This large noise is the result of the sensitivity of ARToolkit to changes in the moving marker orientation. Although the marker orientations are generally maintained as consistent as possible from demonstration to demonstration, the inherent variation in the orientations between manual demonstrations is a contributor to unwanted variation. Ideally, the noise from tracking should not be dominant so that curve learning is performed based only on human-induced variation. This is more the case for the relatively accurate X and Y directions.

Figure 5.19: Output curve and final parameterized data (ten demonstrations) for Case Study 1: (a) Method A, and (b) Method B

(c)

(d)

Figure 5.19 (cont.): Output curve and final parameterized data (ten demonstrations) for Case Study 1: (c) Method C, and (d) Method D

(a)

(b)

(c)

(d)

Figure 5.20: Output curve and final parameterized data (ten demonstrations) for Case Study 2: (a) Method A, (b) Method B, (c) Method C, and (d) Method D

Figure 5.21: Output curve and final parameterized data (ten demonstrations) for Case Study 3: (a) Method A, (b) Method B, (c) Method C, and (d) Method D

Figure 5.22: Example of 3D data points parameterized using the PLP algorithm according to a piecewise linear approximation

5.7.1.5 Effects of the Number of Demonstrations and Different Curve Learning Methods

The results of using Methods A-D for the three case studies are shown in Figures 5.23-5.25. For all cases, the output curve improves with the number of demonstrations. This is because with more demonstrations, the probability of obtaining good data points which are able to compensate for poor ones increases. The distribution of the errors also becomes increasingly normal with respect to the desired curve. This is important because the Bayesian neural network is based on the assumption that the error is Gaussian/normally distributed. Another observation is that the improvement decreases as the number of demonstrations increase. This can be attributed to the increase in density of the point cloud. When the density increases, it is less likely that the network would learn any other relevant trends in the data, *i.e.*, it has less 'room' to adjust the curve compared to a sparse and irregular cloud where unwanted trends might be picked up.

Methods that use the PLP algorithm (Methods B and D), perform better than if simple parameterization is used. This is because the algorithm provides relevant input to the network that better represents the desired curve compared to simple parameterization. The only minor drawback is that the quality of the output curve is dependent on the quality of the piecewise linear approximation provided by the user.

For the planar case with low noise, methods which have the additional reparameterization (Methods C and D) steps tend to perform better. Although the error variances are similar for Methods C and D compared to Methods A and B, the average

and maximum errors are lower for Methods C and D. This indicates that the networks for Methods A and B are not learning the curve strictly based on orthogonal distances between the data points and the curve. The performances of Methods C and 0 also do not differ much which indicates that reparameterization makes up for the initial inaccurate simple parameterization used in Method C. However, Method 0 performs slightly better than Method C because it provides a better initial curve for the reparameterization steps compared to Method C.

For the non-planar cases which are much noisier in the Z direction, methods which have the additional reparameterization steps do not always perform best. However, they always perform *close to best* if they are not the best already. A useful observation is that the performances of methods that use reparameterization would deteriorate if the data set is very noisy (relative to the planar case study). Under these circumstances, there is a possibility that reparameterization changes the parameters of the data points largely enough, so as to create a performance surface that is significantly different from the one obtained from a previous reparameterization step. Hence, the sparseness of the noisy data causes the network with reparameterization to learn unwanted trends in the data, and the overall generalization to be poor. This can be avoided in practice by using good tracking systems, as the source of unwanted variation from human demonstrations is relatively low. This is indicated by the dense point clouds with low variation in the X and Y directions for Methods B-D in Figures 5.19-5.21.

Figure 5.23: Case Study 1: (a) average error, (b) maximum error, and (c) variance of error

Figure 5.24: Case Study 2: (a) average error, (b) maximum error, and (c) variance of error

Figure 5.25: Case Study 3: (a) average error, (b) maximum error, and (c) variance of error

Figures 5.26-5.28 are a graphic evaluation of how well the output curves (red) match the desired curves (black). The improvements in the output curves from one demonstration (Figures 5.26(a), 5.27(a) and 5.28(a)), to five (Figures 5.26(b), 5.27(b) and 5.28(b)) and ten (Figures 5.26(c), 5.27(c) and 5.28(c)) demonstrations can clearly be seen. However, the differences between the output curves for five and ten demonstrations become less apparent. For each figure, the effects of different curve learning methods A-D can also be roughly observed.

Figure 5.26: Output curves overlaid on drawn desired curve for Case Study 1: (a) one demonstration

(b)

Figure 5.26 (cont.): Output curves overlaid on drawn desired curve for Case Study 1: (b) five demonstrations

(c)

Figure 5.26 (eont.): Output curves overlaid on drawn desired curve for Case Study 1: (c) ten demonstrations

(a)

Figure 5.27: Output curves overlaid on drawn desired curve for Case Study 2: (a) one demonstration

(b)

Figure 5.27 (cont.): Output curves overlaid on drawn desired curve for Case Study 2: (b) five demonstrations

(c)

Figure 5.27 (cont.): Output curves overlaid on drawn desired curve for Case Study 2: (c) ten demonstrations

(a)

Figure 5.28: Output curves overlaid on drawn desired curve for Case Study 3: (a) one demonstration

Figure 5.28 (cont.): Output curves overlaid on drawn desired curve for Case Study 3: (b) five demonstrations

(c)

Figure 5.28 (cont.): Output curves overlaid on drawn desired curve for Case Study 3: (c) ten demonstrations

5.7.2 End-Effector Orientation Planning Case Studies

After the output curve is obtained, the orientation of the end-effector can be planned interactively with the aid of AR. For the four case studies presented in this section, the overall RPAR-II methodology is also demonstrated.

5.7.2.1 Simple Case Study

Linear interpolation is useful for planning sequential paths that consist of straight lines and having sharp edges between them. After the user is familiar with the robot and the environment, a CFV is generated along the straight line to be followed as shown in Figure 5.29(a). The user then directly demonstrates the desired orientation of the end-effector and at the same time selects the start point of the line (Figure 5.29(b)). Besides making the selection process more intuitive, the virtual robot is rendered to verify if it is able to reach a particular desired configuration. This is followed by the selection of the end point in Figure 5.29(c). The path is then simulated and the *CFV Check* is used to verify if the path is collision-free (Figures 5.29(d-e)). In this case study, the tip of the end-effector is 2 mm (for example) away from the line, along the principal axis of the end-effector as indicated by the white dotted lines in the enlarged figure on the top right corner of Figure 5.29(d). This gap is known as the *arc length* for arc welding applications. The angle profiles for the final geometric collision-free path are shown in Figure 5.30.

Figure 5.29: Planning the end-effector orientation and a successful collision-free path for the simple case using linear interpolation

Figure 5.30: Angle profiles for the collision-free path for the simple case study

5.7.2.2 Complex Case Studies

After moving the robot and familiarizing with the environment and task (Figures 5.31(a), 5.33(a) and 5.35(a)), the user generates a CFV at the relevant areas along the curve as shown in Figures 5.31(b), 5.33(b) and 5.35(b). The first trials of planning the orientation of the end-effector with respect to the curve are shown in Figures 5.31(c), 5.33(c) and 5.35(c) where the arrows (not part of the original picture) indicate the projection of the probe tip onto the normal plane. The direction of the end-effector is indicated by the annotation "EE Direction" which can be seen clearer in the enlarged figures on the top right corners of Figures 5.31(c), 5.33(c) and 5.35(c). After the user defines the desired orientation of the end-effector at the start of the curve, the remaining transformations of the end-effector along the entire curve are calculated. The pre-defined transformations from the curve coordinate systems to the wrist of the robot, ${}^1_{w_1}M = {}^2_{w_2}M = \dots = {}^n_{w_n}M$ are a translation to the origin of the wrist of the end-

effector followed by a $+20^\circ$ degree rotation about the X-axis for Case Studies 1 and 2, and $+15^\circ$ for Case Study 3. As in the simple case study, a distance of 2 mm between the tip of the end-effector and the output curve is also taken into account in this transformation. As the paths are simulated in Figures 5.31(d), 5.33(d) and 5.35(d), Figures 5.31(e), 5.33(e) and 5.35(e) show the end-effector moving outside the CFV and colliding with an obstacle along the curve. The user then reselects orientations that are not as 'leaning' in towards the obstacles as shown in Figures 5.31(t), 5.33(t) and 5.35(t). After these corrections, successful collision-free paths are simulated in Figures 5.31(g-h), 5.33(g-h) and 5.35(g-h). The angle profiles for the collision-free geometric paths are shown in Figures 5.32, 5.34 and 5.36 for Case Studies 1, 2 and 3 respectively.

Figure 5.31: Planning the end-effector orientation and a successful collision-free geometric path for Case Study 1

Figure 5.31 (cont.): Planning the end-effector orientation and a successful collision-free geometric path for Case Study 1

Figure 5.32: Angle profiles for the collision-free path for Case Study 1

Figure 5.33: Planning the end-effector orientation and a successful collision-free geometric path for Case Study 2

Figure 5.33(cont.): Planning the end-effector orientation and a successful collision-free geometric path for Case Study 2

Figure 5.34: Angle profiles for the collision-free path for Case Study 2

Figure 5.35: Planning the end-effector orientation and a successful collision-free geometric path for Case Study 3

Figure 5.35 (cont.): Planning the end-effector orientation and a successful collision-free geometric path for Case Study 3

Figure 5.36: Angle profiles for the collision-free path for Case Study 3

5.8 Chapter Summary

In this chapter, a RPAR-II methodology for Class II tasks was proposed. The methodology involves the processing of human generated data points to learn an unknown path/curve that is to be followed by the end-effector. A description of the curve learning approach followed by detailed discussions on the motivations behind the choices of methods was also presented. Two algorithms were proposed, the PLP algorithm for parameterization, and the Bayesian neural network and reparameterization algorithm for curve learning. After the output curve is obtained, the orientation of the end-effector with respect to the curve can be planned interactively using visual feedback from AR. The simple case involves a direct demonstration of the desired orientation for a straight line, and the complex case involves a more constrained and controlled orientation selection process for generic space curves.

For the curve learning experiments, the following conclusions can be made:

1. The increase in the number of demonstrations improves the quality of the output curve but the improvement becomes less significant as more demonstrations are used. For the case studies in this chapter, only a small number of good demonstrations (3-5 demonstrations) are required for a reasonably good output using the proposed curve learning approach. This number is also dependent on the sampling rate of the tracking system; with higher sampling rates, more data points can be collected for each demonstration. However, at least *two* demonstrations are needed so that curve learning can be carried out using the human-induced variation *between* demonstrations.

2. The PLP algorithm generally improves the performances of methods with and without reparameterization, compared to simple parameterization.
3. When the data set is dense and less noisy (*e.g.*, planar case), which is expected to be the norm in practice, Method D performs the best among the four methods. Method D is the proposed Bayesian neural network and reparameterization algorithm, Algorithm 5.2.
4. When the data set is very noisy because of tracking errors (*e.g.*, non-planar cases), the advantage of reparameterization becomes less apparent. In general, the performances of all the methods also deteriorate.
5. If sophisticated, robust and accurate tracking systems with low levels of noise are employed, the output curve is expected to be much more accurate as only human-induced variation is used for the curve learning process.

The planning of the end-effector orientation can be carried out in an unconstrained (simple case) or constrained manner (complex case). The capabilities of the human in evaluating both the surface on which the curve lies, and the obstacles along the curve, are fully utilized and guided using AR visualization. In the Class II case studies presented, collision-free paths were successfully planned in a quick and intuitive way.

For the Class II tasks reported in this chapter, the time required for demonstrations depended on the rate with which the user traces the paths and the number of demonstrations used. The neural network typically converges within minutes even for up to 10 demonstrations. Again, this is dependent on the PC processing speed. In general, the Class II tasks for complex curves using four good demonstrations required about 5 minutes from data collection to the final path simulation.

CHAPTER 6

CONCLUSIONS AND RECOMMENDATIONS

The research conducted in this thesis has presented the case for Robot Programming using Augmented Reality (RPAR), a paradigm in robot programming where the user directly interacts with a virtual robot in a real environment. The key contributions of the research are:

1. The proposal of an AR environment for *immersive* robot programming in static environments that may be unknown, where the user directly controls a virtual robot by moving its end-effector. Existing AR robot programming and PbD approaches are slightly different because the human does not directly interact with a real functional robot or a virtual robot. The benefits and possibilities of RPAR over conventional robot programming methods (such as increased intuition and flexibility) were highlighted and discussed.
2. The proposal of a new RPAR-I methodology for tasks where there are a number of possible collision-free path solutions for a given start-goal configuration pair. These are known as Class I tasks; examples are general pick-and-place, spot welding and inspection tasks. The methodology utilizes the capabilities of the human in manoeuvring through and making sense of an unknown environment that is static. A CFV is interactively generated to define the free space relevant to the task; if any part of the robot remains within this volume, it is guaranteed to be collision-free. Collision-free paths can then be planned in an intuitive way by directly demonstrating the desired start and goal

configurations of the virtual robot (as opposed to lines of codes or a teaching pendant) and using search algorithms to automatically generate the paths. In this work, a beam search strategy was adopted. A number of scenarios and case studies were discussed.

3. The proposal of a new RPAR-II methodology for tasks where the end-effector is constrained to follow a user-defined 3D path, at a certain orientation, consistently with respect to that path. These are known as Class II tasks; examples are arc welding and laser cutting. The first step in the methodology requires the user to perform a number of demonstrations to obtain data points of a path/curve to be followed. These data points are then processed by a learning system to produce an output curve that best represents the human demonstrations/intentions:

- i) A Piecewise Linear Parameterization (PLP) algorithm was proposed for parameterizing data points obtained from human demonstrations.
- ii) A Bayesian neural network and reparameterization algorithm was proposed for curve learning. The algorithm combines concepts from Bayesian regularization and Orthogonal Distance Fitting (ODF).
- iii) Experiments, to compare the performances of different curve learning methods (based on parameterization, Bayesian neural networks and reparametrization) and the effects of the number of demonstrations used on the quality of the output curve, were carried out and discussed. A total of three curve learning case studies were presented and discussed.
- iv) After the output curve is found, the user is required to interactively define the CFV along the curve. The orientation of the end-effector can then be

planned intuitively using visual feedback from AR and a final collision-free path for the robot can be obtained. A total of four case studies were presented.

The research conducted has revealed a number of opportunities for further research and development. These are:

1. The development of a more comprehensive RPAR system that includes:
 - i) Libraries of different robot models for improving the process of evaluating different robotic options.
 - ii) More sophisticated and suitable input devices such as data gloves and virtual widgets, and input modes such as speech and gesture control.
 - iii) The incorporation of sophisticated tracking systems that are accurate and robust to further test the methodologies developed in this thesis (especially for curve learning).
 - iv) Sensors for extracting useful information of the environment that is relevant to the task.
2. The development of alternative and improved algorithms for:
 - i) Generating and defining the CFV. This is especially difficult because the CFV may be irregularly shaped.
 - ii) Verifying if objects are within the CFV. This is slightly different from most collision detection algorithms which are more focused on verifying if objects collide with each other.
 - iii) Improved path search strategies in configuration space. The incorporation of real-time sensor data would also be able to guide the search.

- iv) Alternative curve learning approaches that are fast and intuitive.
- 3. The incorporation of motion planning in the path generation where the dynamics of the robot are included.
- 4. The extension of RPAR to more complex tasks that involve contact such as assembly. This will require real-time force analyses and haptic feedback.
- 5. The possibility of using RPAR for the programming of multiple robots with more than one user involved. The roles of each user and robot with relation to the task(s) to be planned have to be clearly established. AR has the potential of facilitating this complex process.

REFERENCES

- [1] Burke, J. L., Murphy, R. R., Rogers, E., Lumelsky, V. J., and Scholtz, J., "Final Report for the DARPA/NSF Interdisciplinary Study on Human-Robot Interaction," *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, Vol. 34, No.2, May 2004, pp. 103-112.
- [2] *JETRO Industrial Report: Trends in the Japanese Robotics Industry, Japan Economic Monthly, Mar. 2006*. Available: <http://www.jetro.go.jp/en/market/trend/industrial/index.html/> , last accessed on 10 Sep. 2006.
- [3] *JETRO Business Report: New Possibilities for Japan's Robot Industry, Japan Economic Monthly, Feb. 2006*. Available: <http://www.jetro.go.jp/en/market/trend/topic/> , last accessed on 10 Sep. 2006.
- [4] Thrun, S., "Toward a Framework for Human-Robot Interaction," *Human-Computer Interaction*, Vol. 19, No. 1/2, Mar. 2004, pp. 9-24.
- [5] Schraft, R. D., Degenhart, E., Hagele, M., and Kahmeyer, M., "New Robot Applications in Production and Service," *Proceedings of the IEEE International Workshop on Advanced Robotics*, Tsukuba, Japan, 8-9 Nov. 1993, pp. 15-23.
- [6] Russell, S., and Norvig, P., *Artificial Intelligence: A Modern Approach*, Englewood Cliffs, NJ: Prentice Hall, 1995.
- [7] *OSHA Technical Manual, Industrial Robots and Robot System Safety, Section IV: Chapter 4*. Available: http://www.osha.gov/dts/ostalotm/otm_iv/otm_iv_4.html, last accessed on 10 Sep. 2006.
- [8] Ulrika, S., "Industrial IT for Robotic Applications - Offline Programming is just like having the Robot on your PC!," *ABB Review- The Corporate Technical Journal of the ABB Group*, No.3, 2001, pp. 28-30.
- [9] Natonek, E., Zimmerman, T., and Fluckiger, L., "Model Based Vision as Feedback for Virtual Reality Robotics Environment," *Proceedings of the IEEE Virtual Reality Annual International Symposium (VRAIS)*, Los Alamitos, CA, 11-15 Mar. 1995, pp. 110-117.

- [10] Aleotti, J., Caselli, S., and Reggiani, M., "Leveraging on a Virtual Environment for Robot Programming by Demonstration," *Robotics and Autonomous Systems*, Vol. 47, No. 2-3, 30 Jun. 2004, pp. 153-161.
- [11] *Robot Work cell Simulation, University of Missouri, Rolla, MO*. Available: <http://web.umar.edu/vrpl/proj-robot.htm>. last accessed on 10 Sep. 2006.
- [12] Burdea, G. C., "Invited Review: The Synergy between Virtual Reality and Robotics," *IEEE Transactions on Robotics and Automation*, Vol. 15, No.3, Jun. 1999, pp. 400-410.
- [13] Latombe, J.-C., *Robot Motion Planning*, Dordrecht: Kluwer Academic, 1991.
- [14] Pires, J., Godinho, T., and Ferreira, P., "CAD Interface for Automatic Robot Welding Programming," *Industrial Robot: An International Journal*, Vol. 31, No.1, Jan. 2004, pp. 71-76.
- [15] Pires, J., Loureiro, A., Godinho, T., Ferreira, P., Fernando, B., and Morgado, F., "Object Oriented and Distributed Software Applied to Industrial Robotic Welding," *Industrial Robot: An International Journal*, Vol. 29, No.2, May 2002, pp. 149-161.
- [16] Asada, H., and Izumi, H., "Automatic Program Generation from Teaching Data for the Hybrid Control of Robots," *IEEE Transactions on Robotics and Automation*, Vol. 5, No.2, Apr. 1989, pp. 166-173.
- [17] Friedrich, H., Kaiser, M., and Dillmann, R., "What can Robots Learn from Humans?" *Annual Reviews in Control*, Vol. 20, 1996, pp. 167-172.
- [18] Paynter, G. W., "Generalising Programming by Demonstration," *Proceedings of the 6th Australian Conference on Computer-Human Interaction*, Hamilton, New Zealand, 24-27 Nov. 1996, pp. 344-345.
- [19] Friedrich, H., Munch, S., Dillmann, R., Bocionek, S., and Sassin, M., "Robot Programming by Demonstration (RPD): Supporting the Induction by Human Interaction," *Machine Learning*, Vol. 23, No. 2-3, May-Jun. 1996, pp. 163-189.

- [20] Delson, N., and West, H., "Robot Programming by Human Demonstration: The Use of Human Variation in Identifying Obstacle Free Trajectories," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, San Diego, CA, Vol. 1, 8-13 May 1994, pp. 564-571.
- [21] Delson, N., and West, H., "Robot Programming by Human Demonstration: The Use of Human Inconsistency in Improving 3D Robot Trajectories," *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems (IROS)*, Munich, Germany, Vol. 2, 12-16 Sept. 1994, pp. 1248-1255.
- [22] Delson, N., and West, H., "Robot Programming by Human Demonstration: Adaptation and Inconsistency in Constrained Motion," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Minneapolis, MN, Vol. 1, 22-28 Apr. 1996, pp. 30-36.
- [23] Atkeson, C. G., and Schaal, S., "Learning Tasks from a Single Demonstration," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Albuquerque, NM, Vol. 2, 20-25 Apr. 1997, pp. 1706-1712.
- [24] Kaiser, M., Retey, A., and Dillmann, R., "Robot Skill Acquisition via Human Demonstration," *Proceedings of the 7th International Conference on Advanced Robotics (ICAR)*, Sant Feliu de Guixols, Spain, Sep. 1995, pp. 763-768.
- [25] Kaiser, M., and Dillmann, R., "Building Elementary Robot Skills from Human Demonstration," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Minneapolis, MN, Vol. 3, 22-28 Apr. 1996, pp. 2700-2705.
- [26] Ehrenmann, M., Zollner, R., Knoop, S., and Dillmann, R., "A Sensor Fusion Approaches for Observation of User Actions in Programming by Demonstration," *Proceedings of the International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, Baden-Baden, Germany, 20-22 Aug. 2001, pp. 227-232.
- [27] Zollner, R., Rogalla, O., Dillmann, R., and Zollner, M., "Understanding Users Intention: Programming Fine Manipulation Tasks by Demonstration," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and System (IROS)*, Lausanne, Switzerland, Vol. 2, 30 Sept.-5 Oct. 2002, pp. 1114-1119.

- [28] Chen, J., and Zelinsky, A., "Programming by Demonstration: Coping with Suboptimal Teaching Actions," *The International Journal of Robotics Research*, Vol. 22, No.5, May 2003, pp. 299-319.
- [29] Lozano-Perez, T., "Spatial Planning: A Configuration Space Approach," *IEEE Transactions on Computers*, Vol.c-32, No.2, Feb. 1983, pp. 108-120.
- [30] Dillmann, R., "Teaching and Learning of Robot Tasks via Observation of Human Performance," *Robotics and Autonomous Systems*, Vol. 47, No. 2-3, 30 Jun. 2004, pp. 109-116.
- [31] Abdel-Malek, L. L., and Li, Z., "The Application of Inverse Kinematics in the Optimum Sequencing of Robot Tasks," *International Journal of Production Research*, Vol. 28, No.1, Jan. 1990, pp. 75-90.
- [32] Milgram, P., and Kishino, F., "A Taxonomy of Mixed Reality Virtual Displays," *Institute of Electronics, Information and Communication Engineers (IEICE) Transactions on Information and Systems*, Vol. E77-D, No.9, Sept. 1994, pp. 1321-1329.
- [33] Azuma, R., "A Survey of Augmented Reality," *Presence: Teleoperators and Virtual Environments*, Vol. 6, No.4, Aug. 1997, pp. 355-385.
- [34] Rekimoto, J., and Nagao, K., "The World through the Computer: Computer Augmented Interaction with Real World Environments," *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology (UIST)*, Pittsburgh, Pennsylvania, 14-17 Nov. 1995, pp. 29-36.
- [35] Rolland, J., and Fuchs, H., "Optical versus Video See-Through Head-Mounted Displays," *Fundamentals of Wearable Computers and Augmented Reality*, Mahwah, NJ: Lawrence Erlbaum Associates Inc., 2001, pp. 113-156.
- [36] Azuma, R., "Tracking Requirements for Augmented Reality," *Communications of the ACM*, Vol. 36, No.7, Jul.1993, pp. 50-51.
- [37] Marx, Z., "Interaction Elements in Studierstube," *Master's Thesis*, The Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2002.

- [38] Rolland, I., Davis, L., and Baillet, Y., "A Survey of Tracking Technology for Virtual Environments," *Fundamentals of Wearable Computers and Augmented Reality*, Mahwah, NJ: Lawrence Erlbaum Associates Inc., 2001, pp. 67-112.
- [39] Kato, H., and Billinghurst, M., "Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System," *Proceedings of the 2nd International Workshop on Augmented Reality*, San Francisco, CA, Oct. 1999, pp.85-94.
- [40] Holloway, R., "Registration Errors in Augmented Reality Systems," *Ph.D Dissertation*, University of North Carolina, Chapel Hill, 1995.
- [41] Holloway, R., "Registration Error Analysis for Augmented Reality Systems," *Fundamentals of Wearable Computers and Augmented Reality*, Mahwah, NJ: Lawrence Erlbaum Associates Inc., 2001, pp. 183-217.
- [42] Smailagic, A., and Siewiorek, D., "Application Design for Wearable and Context-aware Computers," *IEEE Pervasive Computing*, Vol. 1, No.4, Oct.-Dec. 2002, pp.20-29.
- [43] 'Augmented Reality' Speeding Assembly and Service Tasks, Available: <http://www.eetonline.com/story/OEG20030707S0066> , last accessed on 10 Sep. 2006.
- [44] Neumann, U., and Majoros, A., "Cognitive, Performance, and Systems Issues for Augmented Reality Applications in Manufacturing and Maintenance," *Proceedings of the IEEE Virtual Reality Annual International Symposium (VRAIS)*, Atlanta, GA, 14-18 March, 1998, pp. 4-11.
- [45] Feiner, S., MacIntyre, B., and Seligmann, D., "Knowledge-Based Augmented Reality," *Communications of the ACM*, Vol. 36, No.7, Jul. 1993, pp. 53-62.
- [46] Barfield, W., Baird, K., Shewchuk, J., and Ioannou, G., "Applications of Wearable Computers and Augmented Reality to Manufacturing," *Fundamentals of Wearable Computers and Augmented Reality*, Mahwah, NJ: Lawrence Erlbaum Associates Inc., 2001, pp. 695-714.
- [47] Schwald, B., and Laval, B., "An Augmented Reality System for Training and Assistance to Maintenance in the Industrial Context," *Journal of the International Conference in Central Europe on Computer Graphics and Visualization (WSCG)*, Vol. 11, No.1, Plzen, Czech Republic, 3-7 Feb. 2003, pp. 425-432.

- [48] Friedrich, W., "ARVIKA-Augmented Reality for Development, Production and Service," *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, Darmstadt, Germany, 30 Sept.-1 Oct. 2002, pp. 3-4.
- [49] Tang, A., Owen, C., Biocca, F., and Mou, W., "Experimental Evaluation for Augmented Reality in Object Assembly Task," *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, Darmstadt, Germany, 30 Sept.-1 Oct. 2002, pp. 265-266.
- [50] Sharma, R., Molineros, J., and Raghavan, V., "Interactive Evaluation of Assembly Sequences with Mixed (Real and Virtual) Prototyping," *Proceedings of the International Symposium on Assembly and Task Planning (ISATP)*, Marina del Rey, CA, USA, 7-9 Aug. 1997, pp. 287-292.
- [51] Mizell, D., "Boeing's Wire Bundle Assembly Project," *Fundamentals of Wearable Computers and Augmented Reality*, Mahwah, NJ: Lawrence Erlbaum Associates Inc., 2001, pp. 447-467.
- [52] Doil, F., Schreiber, W., Alt, T., and Patron, C., "Augmented Reality for Manufacturing Planning," *Proceedings of the Workshop on Virtual Environments*, Vol. 39, Zurich, Switzerland, 22-23 May 2003, pp. 71-76.
- [53] Rastogi, A., and Milgram, P., "Augmented Telerobotic Control: A Visual Interface for Unstructured Environments", *Proceedings of the Knowledge-Based Systems (KBS)/Robotics Conference*, Montreal, Canada, 16-18 Oct. 1995, pp. 16-18.
- [54] Pettersen, T., Pretlove, J., Skourup, C., Engedal, T., and Lokstad, T., "Augmented Reality for Programming Industrial Robots," *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, Tokyo, Japan, 7-10 Oct. 2003, pp. 319-320.
- [55] Bobrow, J. E., Dubowsky, S., and Gibson, J. S., "Time-Optimal Control of Robotic Manipulators along Specified Paths," *International Journal of Robotics Research*, Vol. 4, No.3, Fall 1985, pp. 3-17.
- [56] *Assistive Robotic Manipulator (ARM)*, Available: <http://www.exactdynamics.nl/>. last accessed on 10 Sep. 2006.

- [57] *Fanuc Pick-and-Place and Spot Welding Robots*, Available:
http://www.fanuc.co.jp/en/news/2003/0311/0311_robotshow.html, last accessed on 10 Sep. 2006.
- [58] *Bridge Inspection Robot*, Available:
http://voronoLsbp.rLcmu.edu/projects/prj_bridgeinspection.html, last accessed on 10 Sep. 2006.
- [59] Wanner, M. C., and Herkomrner, T. F., "Offline Programming for the aircraft cleaning robot "SKYWASH"," *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems (IROS) "Advanced Robotic Systems and the Real World"*, Munich, Germany, Vol. 3, 12-16 Sept. 1994, pp. 1972-1979.
- [60] *Arc Welding Robot*: Available:
<http://www.roboticsonline.com/public/articles/articlesdetails.cfm?id=1000>, last accessed on 10 Sep. 2006.
- [61] *Laser Cutting Robot*: <http://www.lasertechnologies-jenoptik.com/plasticlasercuttingtrimming.htm>, last accessed on 10 Sep. 2006.
- [62] *ARTtrack Optical Tracking Systems*: <http://www.ar-tracking.de/>, last accessed on 10 Sep. 2006.
- [63] *Northern Digital Inc (NDI) Optotrak Certus Tracking System*:
<http://www.ndigital.com>, last accessed on 10 Sep. 2006.
- [64] Craig, 1.C., *Introduction to Robotics, Mechanics and Control, 2nd Ed.*, Reading, MA: Addison-Wesley, 1989.
- [65] *PointGrey FireFly Camera*: <http://www.ptgrey.com/products/firefly/firefly.pdf>, last accessed on 10 Sep. 2006.
- [66] *i-Glasses Video Display Goggles*: <http://www.i-glassesstore.com/iglasses-pc-hr.html>, last accessed on 10 Sep. 2006.
- [67] Khatib, O., "Real-time Obstacle Avoidance for Manipulators and Mobile Robots," *The International Journal of Robotics Research*, Vol. 5, No.1, Jan. 1986, pp. 90-98.

- [68] Kavraki, L. E., Svestka, P., Latombe, J. C., and Overmaers, M. H., "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," *IEEE Transactions on Robotics and Automation*, Vol. 12, No.4, Jun. 1996, pp. 566-580.
- [69] Hsu, D., "Randomized Single-Query Motion Planning in Expansive Spaces," *Ph.D Dissertation*, Department of Computer Science, Stanford University, 2000.
- [70] Hsu, D., Latombe, J. C., and Motwani, R., "Path Planning in Expansive Configuration Spaces," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Albuquerque, NM, Vol. 3, 20-25 Apr. 1997, pp.2719-2726.
- [71] Phillips, J. M., Bedrossian, N., and Kavraki, L. E., "Guided Expansive Space Trees: A Search Strategy for Motion- and Cost- Constrained State Spaces," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, New Orleans, LA, Vol. 4, 26 Apr.-1 May 2004, pp. 3968-3973.
- [72] Sabuncuoglu, I., and Bayiz, M., "Analysis of Reactive Scheduling Problems in a Job Shop Environment," *European Journal of Operations Research*, Vol. 126, No.3, 1 Nov. 2000, pp. 567-586.
- [73] Erel, E., Sabuncuoglu, I., and Sekerci, H., "Stochastic Assembly Line Balancing using Beam Search," *International Journal of Production Research*, Vol. 43, No. 7, Apr. 2005, pp. 1411-1426.
- [74] Furey, D., "Speeding up the Convergence of Online Heuristic Search and Scaling up Offline Heuristic Search," *Ph.D Dissertation*, College of Computing, Georgia Institute of Technology, 2004.
- [75] Cormen, T., Leiserson, C., Rivest, R., and Stein, C., *Introduction to Algorithms, 2nd Ed.*, Cambridge, MA: The MIT Press, 2001.
- [76] Fiorentino, M., de Amicis, R., Monno, G., and Stork, A., "Spacedesign: A Mixed Reality Workspace for Aesthetic Industrial Design," *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, Darmstadt, Germany, 30 Sept.-1 Oct. 2002, pp. 86-94.
- [77] Sung, J. A., *Least Squares Orthogonal Distance Fitting of Curves and Surfaces in Space*, Berlin and Heidelberg: Springer-Verlag, 2004.

- [78] Cohen, E., Riesenfeld, R. F., and Elber, G., *Geometric Modeling with Splines: An Introduction*, Natick, MA: A K Peters, 2001.
- [79] Farin, G., *Curves and Surfaces for CAGD: A Practical Guide, 5th Ed.*, San Francisco, CA: Morgan-Kaufman, 2002.
- [80] Hoschek, J., "Intrinsic Parametrization for Approximation," *Computer-Aided Geometric Design*, Vol. 5, No.1, Jun. 1988, pp. 27-31.
- [81] Goshtasby, A. A., "Grouping and Parameterizing Spaced Points for Curve Fitting," *ACM Transactions on Graphics*, Vol. 19, No.3, Jul. 2000, pp. 185-203.
- [82] MacKay, D. J. C., "Bayesian Methods for Adaptive Models," *Ph.D Dissertation*, Computation and Neural Systems, California Institute of Technology, 1992.
- [83] MacKay, D. J. C., "Bayesian Interpolation," *Neural Computation*, Vol. 4, No.3, May 1992, pp. 415-447.
- [84] MacKay, D. J. C., "Bayesian Methods for Neural Networks: Theory and Applications," *Lecture Notes: University of Cambridge Programme for Industry*, 24-25 Jul. 1995.
- [85] MacKay, D. J. C., *Information Theory, Inferences and Learning Algorithms*, Cambridge: Cambridge University Press, 2003.
- [86] Foresee, F. D., and Hagan, M. T., "Gauss-Newton Approximation to Bayesian Regularization," *Proceedings of the International Conference on Neural Networks (ICNN)*, Houston, TX, Vol. 3, 9-12 Jun. 1997, pp. 1930-1935.
- [87] Marquardt, D. W., "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," *SIAM Journal on Applied Mathematics*, Vol. 11, No.2, 1963, pp. 431-441.
- [88] Demuth, H., and Beale, M., *Neural Network Toolbox: User's Guide Version 3.0: The MathWorks*, 1998.
- [89] Besl, P. J., and McKay, N. D., "A Method for Registration of 3D Shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 14, No.2, Feb. 1992, pp. 239-256.

- [90] Faber, P., and Fisher, R. B., "Euclidean Fitting Revisited," *Proceedings of the International Workshop on Visual Form*, Capri, Italy, 28-30 May, 2001, pp. 165-175.
- [91] Adcock, R. J., "Note on the Method of Least Squares," *Analyst*, Vol. 4, No.6, Nov. 1877, pp. 183-184.
- [92] Setiono, R., and Thong, J. Y. L., "An Approach to Generate Rules from Neural Networks for Regression Problems," *European Journal of Operational Research*, Vol. 155, No.1, May 2004, pp. 239-250.
- [93] Oja, E., and Wang, L., "Robust Fitting by Non-Linear Neural Units," *Neural Networks*, Vol. 9, No.3, Apr. 1996, pp. 435-444.
- [94] Hornik, K., Stinchcombe, M., and White, H., "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, Vol. 2, No.5, 1989, pp. 359-366.
- [95] Rumelhart, D., Hinton, G., and Williams, R., "Learning representations by back-propagating errors," *Nature*, Vol. 323, Oct. 1986, pp. 533-536.

APPENDIX A

COMPARISON BETWEEN TRACKING SYSTEMS

Table A.1 shows the reported levels of accuracy achievable by a number of tracking systems that have been used in AR applications. Optical tracking systems [62-63] are able to provide accurate measurements using spherical retro-reflective markers that can be easily mounted onto the tracked object (*e.g.*, probe). Marker redundancy is also easily achievable with the use of multiple markers or cameras. Ultimately, a trade-off needs to be made between the level of accuracy required by the application and the cost of the equipment.

Table A.I: Comparison between ARToolkit and optical tracking systems

	ARToolkit [39]	ARTtrack [62]	Optotrak Certus [63]
Principle	Image processing of flat square markers	Optical tracking of retro-reflective markers	Optical tracking of retro-reflective markers
Camera Type	Normal	Infra-Red	Infra-Red
Positional Error	$\pm 14\text{mm}$ (1 m distance) $\pm 18\text{ mm}$ (1.5 m distance) $\pm 22\text{mm}$ (2 m distance) $\pm 27\text{mm}$ (2.5 m distance)	$0.2\text{ mm} - 1.4\text{ mm}$ RMS ⁷ (3 m x 3 m x 3 m volume)	$x - 0.10\text{mm}$ $y - 0.10\text{mm}$ $z - 0.15\text{ mm}$ (all RMS and 2.25 m distance)
Angular Error	-	$0.12^\circ - 0.4^\circ$ (3m x 3m x 3m volume)	-

⁷ Root Mean Square

APPENDIX B

BEAM SEARCH PSEUDOCODE

A more detailed version of the pseudocode in Algorithm 4.2 is presented in Algorithm

B.1. The following notations are used:

1. $i = 1, \dots, \mathcal{F}$ where i is a possible configuration after a node is perturbed and n is the degrees of freedom (dot) of the robot.
2. $p = 1, \dots, \max = k$ where p is a path and k is the maximum number of paths maintained (beam size).
3. $H = 1, \dots, \max = k$ where H is the head (current) node for each path p being maintained.
4. $N_{i,p}$ or $N_{H,p}$ are nodes that contain information on: (1) the configuration, q , where $t = 0, \dots, O(h)$ and h is the length of the final path, (2) *current-path-cost*, and (3) *dist-to-goal*.

Algorithm B.1: Detailed beam search pseudocode

-
1. Set the first and only path (length = 1) which consists of only its head node to S , $N_{H,p} \leftarrow S$.
 2. Set *goal-path* = NULL.
 3. **Repeat**
 4. Get $N_{H,p}$ from each path p maintained.
 5. Add all possible branch nodes $N_{i,p}$ from all $N_{H,p}$ to neighbourhood list.
 6. **Repeat**
 7. **If** $N_{i,p}$ not within CFV
 8. Discard U .
 9. **If** $N_{i,p}$ is a revisited node
 10. Discard $N_{i,p}$.
 11. **If** two or more different paths reach the same $N_{i,p}$ at the same search step
 12. Select the $N_{i,p}$ with the lowest *current-path-cost*.
 13. Discard the remaining nodes.
 14. **Until** all nodes in the neighbourhood list are processed.

15. Sort neighbourhood list in ascending order according to *dist-to-goal* (Equation (4.11).
 16. Choose k best nodes from sorted neighbourhood list (or all nodes if total number of nodes in the list is less than k).
 17. Discard remaining nodes (if any).
 18. Repeat
 19. If path p does not have any of its $N_{i,p}$ among the chosen nodes
 20. Delete path p .
 21. else
 22. Copy path ($S \rightarrow \dots \rightarrow N_{H,p}$) from where $N_{i,p}$ originates.
 23. Set the node $N_{i,p}$ as the new head of the path from where it originated,
 $NH,p \leftarrow N_{i,p}$.
 24. Until all nodes in list processed
 25. If any head node reaches the goal according to Equation (4.12), *i.e.*, $NH,p = NH,G$
 26. If *goal-path* = NULL
 27. Save *goal-path* where $p = G$ (candidate goal path G now exists).
 28. Sort head node list according to *current-path-cost*.
 29. If NH,G is the first node in the list, *i.e.*, lowest *current-path-cost*.
 30. Return *goal-path*
 31. else
 32. Delete all paths with *current-path-costs* that are larger than the *current-path-cost* of the *goal-path*.
 33. else
 34. Add current saved NH,G into list of head nodes.
 35. Sort head node list (including NH,G) according to *current-path-cost*.
 36. If goal node with lowest *current-path-cost* found is not NH,G
 37. Replace current *goal-path* with new lowest cost *goal-path*.
 38. If NH,G is the first node in the list, *i.e.*, lowest *current-path-cost*.
 39. Return *goal-path*
 40. else
 41. Delete all paths with *current-path-costs* that are larger than the *current-path-cost* of the *goal-path*.
 42. Until *goal-path* is returned.
-

APPENDIX C

BAYESIAN NEURAL NETWORKS

The following derivations were reported by MacKay [82-85], and Forsee and Hagan [86]. Including β and α as parameters in the network model, the posterior distribution of Equation (5.23) becomes,

$$P(\mathbf{W} | D, \beta, \alpha, M) = \frac{P(D | \mathbf{W}, \beta, M)P(\mathbf{W} | \alpha, M)}{P(D | \beta, \alpha, M)} \quad (\text{C.1})$$

Assuming that both the noise in the data and the prior distributions of the weights (here, weights refers to both the weights and biases of the network) are Gaussian,

$$P(D | \mathbf{W}, \beta, M) = \frac{1}{Z_D(\beta)} \exp(-\beta E_D) \quad (\text{C.2})$$

$$P(\mathbf{W} | \alpha, M) = \frac{1}{Z_W(\alpha)} \exp(-\alpha E_W) \quad (\text{C.3})$$

where $Z_D(\beta) = \left(\frac{\pi}{\beta}\right)^{N/2}$ and $Z_W(\alpha) = \left(\frac{\pi}{\alpha}\right)^{N/2}$, and N is the total number of weights.

Substituting Equations (C.2-C.3) into Equation (C.1),

$$P(\mathbf{W} | D, \beta, \alpha, M) = \frac{\frac{1}{Z_D(\beta)Z_W(\alpha)} \exp(-(\beta E_D + \alpha E_W))}{P(D | \beta, \alpha, M)}$$

$$= \frac{1}{Z_F(\beta, \alpha)} \exp(-E(\mathbf{W})) \quad (C.4)$$

The optimal or most probable weights $\mathbf{W}_{MP}$ are when Equation (C.4) is at its maximum (or at its peak). This occurs when the positive objective function $E(\mathbf{W})$ is minimised.

The derivations above complete the first level of inference. The second level of inference involves finding the optimal values for the parameters β and α that adjust the complexity of the network for the next round of learning,

$$P(\beta, \alpha | D, M) = \frac{P(D | \beta, \alpha, M) P(\beta, \alpha | M)}{P(D | M)} \quad (C.5)$$

Hence, the optimal values for β and α occur when Equation (C.5) is maximised, *i.e.*, $P(D | \beta, \alpha, M)$ is maximised. This result affects Equation (C.1) because $P(D | \beta, \alpha, M)$ is the normalization factor, *i.e.*, good generalization may not necessarily occur when the objective function is minimised. Substituting Equations (C.2-C.4) into Equation (C.5) and assuming that the posterior distribution is Gaussian,

$$P(D | \beta, \alpha, M) = \frac{Z_F(\beta, \alpha)}{Z_D(\beta) Z_W(\alpha)} \quad (C.6)$$

Using a Taylor series expansion around the maximum of the posterior from Equation (C.4) (and minimum of $E(\mathbf{W})$), the following results can be obtained,

$$\beta_{MP} = \frac{N - \gamma'}{2E_D(\mathbf{W}_{MP})} \quad (C.7)$$

$$\alpha_{MP} = \frac{\gamma'}{2E_{\mathbf{w}}(\mathbf{W}_{MP})} \quad (\text{C.8})$$

$$\gamma' = N - 2\alpha_{MP} \text{trace}(\mathbf{H}_{MP}^{-1}) \quad (\text{C.9})$$

where γ' is a measure of the effective parameters used by the network out of a total N weights, and $\mathbf{H}$ is the Hessian matrix,

$$\mathbf{H}_{MP} = \beta_{MP} \nabla^2 E_D + \alpha_{MP} \nabla^2 E_{\mathbf{w}} \quad (\text{C.10})$$

Foresee and Hagan proposed a Gauss-Newton approximation to $\mathbf{H}$ which is virtually free if a Levenberg-Marquardt (LM) algorithm is used in the neural network [86]. This is because the LM algorithm uses $\mathbf{H}$ to update the weights of the network,

$$\Delta \mathbf{W} = -(\mathbf{H} + \lambda \mathbf{I})^{-1} \nabla E(\mathbf{W}) \quad (\text{C.11})$$

$$\nabla E(\mathbf{W}) = 2\mathbf{J}^T(\mathbf{W})e(\mathbf{W}) \quad (\text{C.12})$$

where $\mathbf{I}$ is an identity matrix, and e is the residual.

λ is a learning parameter that is increased if the gradient $\nabla E(\mathbf{W})$ is small (plateau) and decreased if the gradient is very high (steep slope). The purpose is to speed up the algorithm when travelling along small gradients and to avoid overstepping potential minimum points when arriving at steep gradients. The update is performed only after the gradient is known; this is done by retracting a step and resetting the weights to values prior to finding the gradient, and then performing the weight updates using the updated λ . The Jacobian, $\mathbf{J}$ contains the sensitivities of each residual e to every weight in the network and can be obtained by slightly modifying Equations (5.19-5.20) [86].

When the residuals are small, the Gauss-Newton approximation ignores the multiplication of the residuals with their second derivatives. Therefore,

$$\mathbf{H} = \mathbf{J}^T \mathbf{J} \quad (\text{C.13})$$

which means that $\mathbf{H}$ can be obtained easily using the Jacobian.

For the case of Bayesian networks, $\mathbf{H}$ is given by taking the second derivative of the objective function or Equation (5.9), $\mathbf{H} = \nabla^2 E$ and using Equation (C.13),

$$\mathbf{H} \approx 2\beta \mathbf{J}^T \mathbf{J} + 2\alpha \mathbf{I} \quad (\text{C.14})$$

With $\mathbf{H}$, the effective number of parameters is found using Equation (C.9) and the updates for β and a can be made using Equations (C.7-C.8). Convergence is achieved when the objective function stops changing significantly.

A number of important issues for the Bayesian neural network are [82-86]:

1. If γ' is found to be close to N , the complexity of the network is most likely inadequate and the number of hidden layer neurons should be increased. In practice, N should be kept relatively large to find the optimal value of γ' which should be significantly smaller than N and then proceed to trim the network. This was used to decide the appropriate network size for the curve learning problem in this thesis.
2. The input and output of the training data needs to be between $[-1, 1]$ for the Gauss-Newton approximation in a LM network.
3. The noise/variation in the data needs to be Gaussian for good performance.

APPENDIX D

CALCULATING THE PROJECTION OF THE TRACKED PROBE TIP ONTO THE NORMAL PLANE

Let $\hat{Y}_1 = [a, b, c]^T$ be the unit vector of the curve tangent at the start of the curve.

The normal plane can be defined uniquely by,

$$a(x - x_0) + b(y - y_0) + c(z - z_0) = 0 \quad (0.1)$$

where $[x_0, y_0, z_0]^T$ are the coordinates of the start point of the curve.

The projected point $[x_p, y_p, z_p]^T$ with respect to the world coordinate system can then

be calculated with the following equations,

$$\begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + u \begin{bmatrix} a \\ b \\ c \end{bmatrix} \quad (0.2)$$

where

$$u = \frac{a(x_0 - x_{tracked}) + b(y_0 - y_{tracked}) + c(z_0 - z_{tracked})}{a^2 + b^2 + c^2}, \quad (D.3)$$

and $[x_{tracked}, y_{tracked}, z_{tracked}]^T$ are the tracked probe tip coordinates.

APPENDIX E

CONTENTS OF ATTACHED CD-ROM

The attached CD-ROM contains the following:

- I. Jon thesis final.pdf: PDF file of the thesis (Acrobat Viewer 7.0 is needed to view this file).
2. Demo videos (Windows Media Player 10 is needed to open these files):
 - a) Class I tasks:
 - i) Miniature poles.wmv: Three examples of navigating the end-effector through a set of poles (pins) in a miniature environment.
 - ii) Miniature two path pick-and-place.wmv: Planning two sequential paths in a miniature factory environment.
 - iii) Full-scale pick and place.wmv: Full scale pick-and-place example.
 - b) Class II tasks:
 - i) Miniature simple case using linear interpolation.wmv: Simple case where the desired end-effector's orientation is directly demonstrated (unconstrained) and the path to be followed is a straight line. The end-effector's orientation with respect to the line is constant along the entire line.
 - ii) Case 1- Miniature 'AR' planar curve.wmv: Miniature case study for a complex planar curve.
 - iii) Case 2- Miniature mouse.wmv: Miniature case study for a curve on a mouse.
 - iv) Case 3- Full-scale sheet metal.wmv: Full-scale case study for a curve on a piece of bent sheet metal.