Static and Bootstrapped Neuro-Simulation for Complex Robots in Evolutionary Robotics

Grant Warren Woodford

Submitted in fulfilment of the requirements for the degree of Philosophiae Doctor (Computer Science) in the Faculty of Science at the Nelson Mandela University

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

December 2019

Supervisor: Mathys C. du Plessis

Acknowledgements

I would like to thank my examiners for their time and energy in reviewing this research. I hope you find this work interesting and thoughtful.

A sincere thanks is due to the many people who helped with the completion of this thesis. I would like to thank Dr MC du Plessis and Dr CJ Pretorius for their support and patience. I could not have completed this thesis without their expertise. I would like to thank Ms A De Franca for her help with the important and lengthy data collection phase of this research.

I would like to thank the Department of Computing Sciences at the Nelson Mandela University for the resources required to complete this work. I have learnt and grown a lot during this period.

Thank you to my parents, friends and family for their support and patience.

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the authors and are not necessarily to be attributed to the NRF.

Abstract

Evolutionary Robotics (ER) is a field of study focused on the automatic development of controllers and robot morphologies. Evolving controllers on real-world hardware is time-consuming and can damage hardware through wear. Robotic simulators can be used as an alternative to a real-world robot in order to speed up the ER process. Most simulation techniques in practice use physics-based models that rely on an understanding of the robotic system in question. Developing effective physics-based simulators is time-consuming and requires a significant level of specialised knowledge. A lengthy simulator development and tuning process is typically required before the ER process can begin.

Artificial Neural Networks simulators (SNNs) can be used as an alternative to a physics-based simulation approach. SNNs are simple to construct, do not require significant levels of prior knowledge of the robotic system, are computationally efficient and can be highly accurate. Two types of ER approaches utilising SNNs exist. The Static Neuro-Simulation (SNS) approach involves developing SNNs before the ER process where these SNNs are used instead of a physics-based simulator. Alternatively, SNNs can be developed during the ER process, called the Bootstrapped Neuro-Simulation (BNS) approach.

Prior work investigating SNNs has largely been limited to simple robots. A complex robot has many degrees of freedom and if a low-level controller design is used, the solution search space is high-dimensional and difficult to traverse. Prior work investigating the SNS and BNS approaches have mostly relied on simplified controller designs which rely on built-in prior knowledge of intended robot behaviours. This research uses low-level controller designs which in turn rely on low level simulators.

Most ER studies are conducted on a single type of robot morphology. This research investigates the SNS and BNS approaches on two significantly different classes of robots. A Hexapod and Snake robot are used to study the SNS and BNS approaches. The Hexapod robot exhibits limbed, walking behaviours. The Snake robot is limbless and generates crawling behaviours. Demonstrating the viability of the SNS and BNS approaches for two different classes of robots provides strong evidence that the tested approaches are likely viable on other classes of robots.

Various proposed improvements to the SNS and BNS approaches are investigated. The

performance and transferability properties of these tested improvements are studied and presented in this research. For each robot investigated, comparisons are made between the SNS and BNS approaches in terms of the observed performance and transferability properties achieved.

Results demonstrate that the SNS and BNS approaches are viable when applied to Hexapod and Snake robots without restricting controller designs to those with significant levels of built-in prior knowledge of robot behaviours. SNNs configured in ensembles improve the likely performance outcomes of solutions. The expected benefit of adding simulator noise during the evolutionary process were not as pronounced for problems investigated in this work.

NELSON MANDELA
UNIVERSITY

DECLARATION BY CANDIDATE

NAME: Grant Warren Woodford

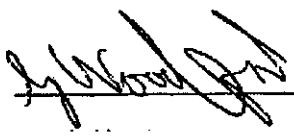
STUDENT NUMBER: s205014224

QUALIFICATION: PhD (Computer Science)

TITLE OF PROJECT: Static and Bootstrapped Neuro-Simulation for Complex Robots in Evolutionary Robotics

DECLARATION:

In accordance with Rule G5.6.3, I hereby declare that the above-mentioned treatise/ dissertation/ thesis is my own work and that it has not previously been submitted for assessment to another University or for another qualification.

SIGNATURE: 

DATE: 2019/07/29

Contents

1 RESEARCH CONTEXT **1**

1.1 Introduction 1

1.2 Research Objectives 4

1.3 Methodology 4

1.4 Dissertation Layout 6

2 RELATED WORK **9**

2.1 Introduction 9

2.2 Artificial Neural Networks 10

2.2.1 The Artificial Neuron 10

2.2.2 Neural Networks 11

2.2.3 ANN Training 15

2.2.4 Uncertainty Estimation 17

2.2.5 Bias-Variance Tradeoff 18

2.2.6 Ensembles 20

2.3 Evolutionary Computation 20

2.3.1 Evolutionary Algorithms 21

2.3.2 Evolutionary Robotics 22

2.3.2.1 Evolutionary Robotics Process 22

2.3.2.2 General State of the Evolutionary Robotics Field 23

2.4 Snake Robot Morphologies 25

2.5 Simulators 27

2.5.1 Evolving Controllers in Reality 28

2.5.2 Simulators in Evolutionary Robotics 28
2.5.2.1 Simulation Strategies 29
2.5.2.2 Evolving Controllers in Simulation 29
2.5.2.3 Simulator Development using Machine Learning Techniques 31
2.5.3 Bidirectional Simulation Development 32
2.5.3.1 Bidirectional Mechanism 32
2.5.3.2 Anytime Learning 33
2.5.3.3 Estimation-Exploration Algorithm 33
2.5.3.4 Back to Reality Algorithm 35
2.5.3.5 Transferability Approach 37
2.5.3.6 Intelligent Trial-and-Error Learning 38
2.5.3.7 Model-fitting based on Empirical Data 39
2.5.4 Simulator Neural Networks 41
2.5.4.1 Behavioural Components 42
2.5.4.2 Level of Prior Knowledge 43
2.5.4.3 Evaluation Platforms 44
2.5.4.4 The SNS Approach 45
2.5.4.5 The BNS Approach 48
2.5.4.6 Simulating Snake Robots using SNNs 51
2.5.4.7 Simulating Hexapod Robots using SNNs 52
2.6 High Level Comparison of ER Approaches 54
2.7 Conclusions 59

3 EXPERIMENTAL METHOD 61

3.1 Introduction 61
3.2 Robot Morphologies 62
3.2.1 Hexapod Robot 62
3.2.2 Snake Robot 63
3.3 Controller Design 65
3.4 Data Collection 66
3.5 Challenges 67

3.6 Methodology . . . 70
3.7 Adaptations . . . 73
3.7.1 Simulator Configurations . . . 75
3.7.1.1 Basic Configuration . . . 76
3.7.1.2 Dropout Configuration . . . 77
3.7.1.3 Ensemble Configuration . . . 78
3.7.1.4 Basic Multi-output Configuration . . . 78
3.7.1.5 Ensemble Multi-output Configuration . . . 79
3.7.2 Simulator Noise . . . 81
3.7.3 Controller Resetting . . . 82
3.7.4 Simulator Resetting . . . 83
3.7.5 Sampling Strategy . . . 83
3.8 Conclusions . . . 84

4 HEXAPOD STATIC NEURO-SIMULATION 85

4.1 Introduction . . . 85
4.2 Experimental Procedure . . . 85
4.2.1 Controllers . . . 86
4.2.2 The Simulator . . . 87
4.3 SNS Experiments . . . 88
4.4 The SNS Experiment Results . . . 90
4.4.1 Demonstration . . . 91
4.4.2 Static SNN Results . . . 93
4.4.2.1 Behavioural Data . . . 94
4.4.2.2 Training Errors . . . 94
4.4.3 Performance . . . 98
4.4.4 Transferability . . . 101
4.4.5 Simulated and Real-world Trajectories . . . 104
4.4.6 Best Controllers . . . 111
4.5 Conclusion . . . 116

5 HEXAPOD BOOTSTRAPPED NEURO-SIMULATION 118

5.1 Introduction . . . 118

5.2 Experimental Procedure . . . 119

5.2.1 Hardware and Data Capture . . . 120

5.2.2 Controllers . . . 120

5.2.3 Simulator . . . 121

5.3 BNS Experiments . . . 123

5.3.1 The Simulated BNS Experiments . . . 123

5.3.2 The BNS Validation Experiments . . . 127

5.4 Successful BNS Hexapod Controller . . . 128

5.5 The Simulated BNS Experiment Results . . . 130

5.5.1 Overall Comparisons . . . 131

5.5.1.1 Simulator Configurations . . . 131

5.5.1.2 Resetting Procedures . . . 132

5.5.1.3 Simulator Noise . . . 135

5.5.1.4 Sampling Strategies . . . 136

5.5.1.5 Summary . . . 137

5.5.2 Transferability . . . 138

5.5.2.1 Simulator Configurations . . . 138

5.5.2.2 Resetting Procedures . . . 140

5.5.2.3 Simulator Noise . . . 142

5.5.2.4 Sampling Strategies . . . 143

5.5.2.5 Summary . . . 144

5.5.3 Convergence Properties . . . 145

5.5.4 Top Performers . . . 147

5.5.5 Summary . . . 151

5.6 The BNS Validation Experiment Results . . . 154

5.6.1 Performance . . . 156

5.6.2 Transferability . . . 159

5.6.3 Validation Solutions . . . 161

5.6.4 Summary . . . 170

5.7 SNS and BNS Comparisons . . . 171
5.8 Conclusion . . . 172

6 SNAKE STATIC NEURO-SIMULATION 174
6.1 Introduction . . . 174
6.2 Experimental Procedure . . . 175
6.2.1 Controllers . . . 175
6.2.2 Simulator . . . 177
6.3 Simulator Benchmark Experiments . . . 177
6.4 Static SNN Results . . . 178
6.4.1 Behavioural Data . . . 178
6.4.2 Ideal SNN Architectures . . . 180
6.4.3 Training Errors . . . 182
6.5 SNS Experiments . . . 186
6.6 The SNS Experiment Results . . . 190
6.6.1 Demonstration . . . 191
6.6.2 Performance . . . 192
6.6.3 Transferability . . . 198
6.6.4 Comparisons . . . 199
6.6.5 Simulated and Real-world Trajectories . . . 202
6.6.6 Best Controllers . . . 207
6.7 Comparing the SNS approach for the Hexapod and Snake robots . . . 212
6.8 Conclusion . . . 213

7 SNAKE BOOTSTRAPPED NEURO-SIMULATION 215
7.1 Introduction . . . 215
7.2 Experimental Procedure . . . 216
7.2.1 Hardware and Data Capture . . . 217
7.2.2 Controllers . . . 217
7.2.3 Simulator . . . 217
7.3 BNS Experiments . . . 218
7.3.1 The Simulated BNS Experiments . . . 218

7.3.2 The BNS Validation Experiments . . . 222
7.4 Successful BNS Snake Controller . . . 223
7.5 The Simulated BNS Experiment Results . . . 225
7.5.1 Overall Comparisons . . . 225
7.5.1.1 Simulator Configurations . . . 225
7.5.1.2 Resetting Procedures . . . 226
7.5.1.3 Simulator Noise . . . 228
7.5.1.4 Sampling Strategies . . . 229
7.5.1.5 Summary . . . 230
7.5.2 Transferability . . . 230
7.5.2.1 Simulator Configurations . . . 231
7.5.2.2 Resetting Procedures . . . 232
7.5.2.3 Simulator Noise . . . 234
7.5.2.4 Sampling Strategies . . . 235
7.5.2.5 Summary . . . 235
7.5.3 Convergence Properties . . . 236
7.5.4 Top Performers . . . 237
7.5.5 Summary . . . 243
7.6 The BNS Validation Experiment Results . . . 248
7.6.1 Performance . . . 249
7.6.2 Transferability . . . 250
7.6.3 Validation Solutions . . . 252
7.6.4 Summary . . . 262
7.7 SNS and BNS Comparisons . . . 263
7.8 Conclusions . . . 264

8 CONCLUSIONS AND FUTURE WORK 265

8.1 Introduction . . . 265
8.2 Overview of Experimental Results . . . 266
8.3 Outcomes of Research Objectives . . . 266
8.4 Contributions and Recommendations . . . 274

8.5 Limitations 276
8.6 Future Research 277
8.7 Summary 277

Appendices **290**

A Research Output by the Author **291**

B Experimental Results **293**

List of Figures

1.1 Structure of dissertation 7

2.1 Artificial Neuron 11
2.2 Rectified Linear Unit (ReLu) activation function 12
2.3 Artificial Neural Network 13
2.4 ANN predictions with dropout enabled for a simple regression problem (Gal and Ghahramani, 2016) 18
2.5 Graphical illustration of bias and variance (Fortmann-Roe, 2012) 19
2.6 Bias and variance contribution to total error (Fortmann-Roe, 2012) 20
2.7 The Evolutionary Robotics Process applied to the evolution of an ANN based controller (Floreano, Husbands, and Nolfi, 2008; Pretorius, 2010) . . 23
2.8 Snake locomotion modes 26
2.9 Snake robot (left) and rattlesnake (right) executing turning behaviours . . . 27
2.10 Anytime Learning System 34
2.11 Estimation-Exploration Algorithm (Bongard, Zykov, and Lipson, 2006b) . . 36
2.12 Intelligent Trial-and-Error Learning (Cully, Clune, Tarapore, and Mouret, 2015) 40
2.13 Behavioural components of a robot 43
2.14 Controller evaluation platforms 46
2.15 Static Neuro-Simulation (SNS) approach 47
2.16 Bootstrapped Neuro-Simulation (BNS) 49
2.17 Simulated and real-world fitness over time for BNS approach (Woodford, Pretorius, and du Plessis, 2016) 51
2.18 Snake controller and simulator design 53

2.19 Hexapod controller and simulator design 54

3.1 Hexapod robot 63
3.2 Snake robot 65
3.3 Controller example 65
3.4 Behavioural Components 66
3.5 Data Collection 68
3.6 Methodology A: Adaptations to the SNS approach 71
3.7 Methodology B: Adaptations to the BNS approach 71
3.8 Methodology C: Real-world validation of adaptations 72
3.9 Venn Diagram of SNS and BNS adaptations 74
3.10 Basic Configuration 76
3.11 SNN Architecture 77
3.12 Dropout Configuration 77
3.13 Ensemble SNN Configuration 79
3.14 Basic Multi-output SNN Configuration 80
3.15 Multi-output SNN architecture 80
3.16 Ensemble Multi-output SNN Configuration 81

4.1 Methodology A: Adaptations to the SNS approach 86
4.2 Solution controller demonstration 92
4.3 Simulated and real-world trajectory paths for solution controller 93
4.4 Scatter plot for the training data Δx and Δy components 95
4.5 Density plot for the training data Δx and Δy components 95
4.6 Density plot of the training data Δa component 96
4.7 Performance distributions for SNS adaptations 101
4.8 Transferability distributions for the SNS adaptations 103
4.9 Solution paths for the Basic (HBE) and Basic Multi-output (HSE) simulator configurations without noise 107
4.10 Solution paths for the Basic (HBN) and Basic Multi-output (HSN) simulator configurations with noise 108

4.11 Solution paths for Dropout (HDE), Ensemble (HEE) and Ensemble Multi-output (HME) configurations without noise 109
4.12 Solution paths for Dropout (HDN), Ensemble (HEN) and Ensemble Multi-output (HMN) configurations with noise 110
4.13 Solution diversity and magnitude trends between SNS adaptations 111
4.14 Best performing solutions for the Basic and Basic Multi-output simulator configurations 114
4.15 Best performing solutions for the Dropout, Ensemble and Ensemble Multi-output configurations 115

5.1 Methodology B: Adaptations to the BNS approach 119
5.2 Methodology C: Real-world validation of adaptations 120
5.3 Solution controller demonstration 129
5.4 Real-world and simulated trajectories 130
5.5 Performance comparisons between simulator configurations 132
5.6 Performance comparison between resetting procedures 134
5.7 Performance comparison for simulator noise 136
5.8 Performance comparisons of sampling strategies 137
5.9 Transferability distributions for each simulator configurations 139
5.10 Transferability distributions for each resetting procedure 141
5.11 Transferability distributions between simulator noise approaches 143
5.12 Transferability distributions between sampling strategies 144
5.13 Performance over time for the best performer of each resetting procedure . 146
5.14 Performance distributions of the top 10 adaptations for the BNS approach . 148
5.15 Transferability distributions for the top 10 BNS adaptations 151
5.16 Validation experiment performance distributions 156
5.17 BNS validation performance distributions grouped by simulator noise 159
5.18 Validation experiment transferability distributions 160
5.19 BNS validation transferability statistics grouped by simulator noise 162
5.20 HESEU Real-world experiments 163
5.21 HESNU Real-world experiments 164

5.22 HMSEU Real-world experiments . . . 165
5.23 HMSNU Real-world experiments . . . 166
5.24 HBSET Real-world experiments . . . 167
5.25 HBSNT Real-world experiments . . . 168

6.1 Methodology A: Adaptations to the SNS approach . . . 175
6.2 Scatter plot for the training data Δx and Δy behavioural components . . . 179
6.3 Density plot for the Δx and Δy behavioural components . . . 180
6.4 Density plot for the Δa and Δo behavioural components . . . 181
6.5 Controller solution that produces turning behaviours . . . 187
6.6 Solution controller demonstration (SME adaptation) . . . 192
6.7 Simulated and Real-world trajectories of solution controller . . . 193
6.8 Performance distributions for SNS adaptations . . . 194
6.9 Transferability distributions for SNS adaptations . . . 198
6.10 Scatter plot comparing the tested adaptations according to the number of failed controller solutions, median performance and median transferability . 201
6.11 Scatter plot (2D) comparing the tested adaptations according to the number of failed controller solutions, median performance and median transferability 202
6.12 Solution paths for the Basic and Basic Multi-output simulator configurations without noise . . . 203
6.13 Solution paths for the Basic and Basic Multi-output simulator configurations with noise . . . 204
6.14 Solution paths for Dropout, Ensemble and Ensemble Multi-output configurations without noise . . . 205
6.15 Solution paths for Dropout, Ensemble and Ensemble Multi-output configurations with noise . . . 206
6.16 Best performing solutions for the Basic and Basic Multi-output simulator configurations . . . 208
6.17 Best performing solutions for the Dropout, Ensemble and Ensemble Multi-output configurations . . . 209

7.1 Methodology B: Simulated experimental trial of BNS adaptation . . . 216

7.2 Methodology C: Real-world demonstration of BNS adaptation 216
7.3 Solution controller demonstration (SMSNU adaptation) 224
7.4 Performance comparison between simulator configurations 226
7.5 Performance comparison between resetting procedures 228
7.6 Performance distributions for simulator noise and sampling strategies 230
7.7 Transferability distributions for simulator configurations 231
7.8 Transferability distributions for resetting procedures 233
7.9 Transferability distributions for simulator noise and sampling strategies . . 236
7.10 Performance over time for the best performer of each resetting procedure . 237
7.11 Performance distributions of the top 10 adaptations for the BNS approach . 239
7.12 Density of performances for best performing adaptations 243
7.13 Best solution for the SSNET adaptations 244
7.14 Transferability distributions for the top 10 BNS adaptations 245
7.15 Validation experiment performance distributions 250
7.16 BNS validation performance distributions grouped by simulator noise 252
7.17 Validation experiment transferability distributions 253
7.18 BNS validation transferability statistics grouped by noise 255
7.19 SESEU Real-world experiments 256
7.20 SESNU Real-world experiments 257
7.21 SMSEU Real-world experiments 258
7.22 SMSNU Real-world experiments 259
7.23 SBSET Real-world experiments 260
7.24 SBSNT Real-world experiments 261

List of Tables

2.1 Robot morphologies investigated using the SNS and BNS approaches 45
2.2 Comparison of ER approaches 55
2.3 Validated robot morphologies per ER approaches 57

4.1 Parameters for controller evolution 89
4.2 SNS Experimental Adaptations 90
4.3 Training dataset MSE and IQR for each simulator configuration 97
4.4 Test dataset MSE and IQR for each simulator configuration 98
4.5 Performance statistics for the SNS adaptations 99
4.6 Comparisons between the performances of adaptations for the SNS approach 100
4.7 SNS trial run durations 102
4.8 Transferability statistics for the SNS adaptations 104
4.9 Comparison between transferability distributions of adaptations for the SNS approach 105
4.10 Behavioural metrics of the best controller in each adaptation 113

5.1 BNS adaptations using no resetting procedure 124
5.2 BNS adaptations using the controller resetting procedure 125
5.3 BNS adaptations using the simulator resetting procedure 125
5.4 BNS adaptations using the controller and simulator resetting procedures . . 126
5.5 Number of controller evolution generations iterated per sampling controller evaluation for the BNS approach on the Hexapod robot 127
5.6 Summary statistics for the simulator configuration performance distributions 133

5.7 The p-values of post hoc analysis comparing performance distributions between simulator configurations . . . 133
5.8 Summary statistics for the resetting procedure performance distributions . . 135
5.9 The p-values from post hoc analysis comparing performance distributions between resetting procedures . . . 135
5.10 Summary statistics for the simulator noise performance distributions 136
5.11 Summary statistics of sampling strategy performance distributions 137
5.12 Transferability statistics for the simulator configurations 138
5.13 Transferability distribution p-values for comparisons between simulator configurations . . . 139
5.14 Transferability statistics for the resetting procedures 142
5.15 Transferability distribution p-values for comparisons between resetting procedures . . . 142
5.16 Transferability statistics for the simulator noise 143
5.17 Transferability statistics for sampling strategies 144
5.18 Performance summary of the top 10 adaptations for the BNS approach . . . 149
5.19 Performance comparisons between the top 10 adaptations for the BNS approach . . . 150
5.20 The transferability statistics for the top 10 adaptations for the BNS approach 152
5.21 Transferability comparisons between the top 10 adaptations for the BNS approach . . . 153
5.22 The p-values for transferability variance comparisons between the HESEU adaptation and the other top adaptations for the BNS approach . . . 154
5.23 BNS performance statistics on validation results 157
5.24 BNS validation performance statistics grouped by simulator noise 158
5.25 BNS transferability statistics on validation results 161
5.26 BNS validation transferability statistics grouped by simulator noise 161

6.1 SNN architectures 181
6.2 Training dataset MSE and IQR for each simulator configuration 183
6.3 Test dataset MSE and IQR for each simulator configuration 184

6.4 Parameters for controller evolution . . . 189
6.5 SNS Experimental Adaptations . . . 190
6.6 Performance statistics for the SNS adaptations . . . 195
6.7 Comparisons between the performance distributions of adaptations for the SNS approach . . . 196
6.8 Failure rates for the tested SNS adaptations . . . 197
6.9 SNS trial run durations . . . 197
6.10 Transferability statistics for the SNS adaptations . . . 199
6.11 Comparisons between the transferability distributions of adaptations for the SNS approach . . . 200
6.12 Behavioural metrics of the best controller in each adaptation . . . 211

7.1 BNS adaptations using no resetting procedure . . . 219
7.2 BNS adaptations using the controller resetting procedure . . . 220
7.3 BNS adaptations using the simulator resetting procedure . . . 220
7.4 BNS adaptations using the controller and simulator resetting procedures . . . 221
7.5 Number of controller evolution generations iterated per sampling controller evaluation for the BNS approach on the Snake robot . . . 222
7.6 Summary Statistics for the simulator configuration performance distributions 226
7.7 The p-values of post hoc analysis comparing performance distributions between simulator configurations . . . 227
7.8 Summary statistics for the resetting procedure performance distributions . . . 227
7.9 The p-values of post hoc analysis comparing performance distributions between resetting procedures . . . 227
7.10 Summary statistics for the simulator noise performance distributions . . . 229
7.11 Summary statistics for the sampling strategy performance distributions . . . 229
7.12 Transferability statistics for the simulator configurations . . . 232
7.13 Transferability distribution p-values for comparisons between simulator configurations . . . 232
7.14 Transferability statistics for the resetting procedures . . . 233

7.15 Transferability distribution p-values for comparisons between resetting procedures . . . 234
7.16 Transferability statistics for simulator noise . . . 235
7.17 Transferability statistics for the sampling strategies . . . 235
7.18 Performance distributions of the top 10 adaptations for the BNS approach . 238
7.19 Performance comparisons between the top 10 adaptations for the BNS approach . . . 240
7.20 The transferability statistics for the top 10 adaptations for the BNS approach 246
7.21 Transferability comparisons between the top 10 adaptations for the BNS approach . . . 247
7.22 BNS performance statistics on validation results . . . 251
7.23 BNS validation performance statistics grouped by simulator noise . . . 251
7.24 BNS transferability statistics on validation results . . . 254
7.25 BNS validation transferability statistics grouped by simulator noise . . . 254

8.1 Summary of high level findings . . . 267
8.2 Summary of adaptation result for the SNS approach . . . 268
8.3 Summary of adaptations results for the BNS approach . . . 269

B.1 Performance results for the SNS approach applied to the Hexapod robot . . 294
B.2 Transferability results for the SNS approach applied to the Hexapod robot . 295
B.3 Performance results for the SNS approach applied to the Snake robot . . . 296
B.4 Transferability results for the SNS approach applied to the Snake robot . . 297
B.5 Validation performance results for the Hexapod robot . . . 298
B.6 Validation transferability results for the Hexapod robot . . . 298
B.7 Validation performance results for the Snake robot . . . 298
B.8 Validation transferability results for the Snake robot . . . 298
B.9 Performance summary for the simulated BNS experiments of the Hexapod robot . . . 300
B.10 Transferability summary for the simulated BNS experiments of the Hexapod robot . . . 302
B.11 Performance summary for the simulated BNS experiments of the Snake robot 304

B.12 Transferability summary for the simulated BNS experiments of the Snake robot 306

List of Abbreviations

Abbreviation	Term
Adam	Adaptive Moment Estimation
AN	Artificial Neuron
ANN	Artificial Neural Network
BNS	Bootstrapped Neuro-Simulation
BTR	Back to Reality Algorithm
EA	Evolutionary Algorithm
EC	Evolutionary Computing
EEA	Estimation-Exploration Algorithm
ER	Evolutionary Robotics
GA	Genetic Algorithm
IQR	Interquartile Range
SNN	Simulator Neural Network
SNS	Static Neuro-Simulation

Chapter 1

RESEARCH CONTEXT

1.1 Introduction

Evolutionary Robotics (ER) is a methodology for the automatic creation of autonomous robotic systems [Floreano *et al.*, 2008]. ER is related to the Evolutionary Computation (EC) field where algorithms are inspired by biological evolution. A controller, often referred to as a policy [Chatzilygeroudis, Rama, Kaushik, Goepp, Vassiliades, and Mouret, 2017], is a program that manages a robot's interactions with its environment [Brooks, 1992; Miglino, Lund, and Nolfi, 1995]. In ER, a population of controllers and/or robot morphologies is evolved using Darwinian principles. Evaluating the large number of controllers produced through the ER process on real-world hardware is impractical and time-consuming. A simulator, often referred to as a model [Nguyen-Tuong and Peters, 2011], is a method for approximating the behaviour of a real-world robotic system. Controllers can be evaluated in a robotic simulation that acts as a surrogate to real-world hardware [Zagal and Ruiz-del Solar, 2007]. A simulator can greatly speed up the ER process by evaluating controllers faster than possible in reality. The ER process, carried out in simulation, produces a final controller solution that can be validated on a real-world robot.

Traditional, physics-based simulators can become complex and time-consuming to develop. Simulators are constructed before the ER process can begin. Specialised domain knowledge is normally required to construct and effectively utilise simulators in ER. Specialised knowledge specific to a given robotic system may need to be acquired in order to

be accurately simulated.

The behaviours observed when evaluating controllers in simulation are often significantly different when those same controllers transferred into reality, known as the "Reality Gap" problem [Jakobi, Husbands, and Harvey, 1995]. The degree to which a controller's simulated behaviours transfer well into reality is commonly referred to as the transferability. Controllers evolved in simulation often do not transfer well into reality and have poor transferability caused by exploiting inaccuracies in modelled phenomena [Koos, Mouret, and Doncieux, 2013b]. Developing ER approaches that are resistant to the *reality-gap* problem is an important topic in ER [Jakobi *et al.*, 1995; Miglino *et al.*, 1995].

ER approaches that produce highly transferable controllers generally have some sort of bi-directional mechanism. Bi-direction ER approaches should be considered a broad category of ER approaches. In a bi-directional ER process, controllers are continually evaluated in reality and information about controller transferability is used to improve the simulator during the ER process [Bongard, 2013]. The optimisation process simultaneously improves both the controllers and simulator. A physics-based simulator is typically used in most bi-directional ER approaches. For bi-directional ER approaches, a least a partially developed simulator needs to be built before the ER process can begin.

Due to the complexity and prior knowledge required to build physics-based simulators, automating simulator development is difficult. However, an alternative simulation approach can be used. Artificial Neural Networks (ANNs) can be used to simulate robot behaviours. ANN-based simulators are referred to as Simulator Neural Networks (SNNs). SNNs are simple to construct, accurate and require little specialised knowledge [Pretorius, du Plessis, and Gonsalves, 2017, 2019]. Traditional SNNs are created and trained before the ER process begins and remain unchanged during the ER process. Static, precomputed SNNs are developed using the Static Neuro-Simulation (SNS) approach. The SNS approach requires a lengthy data collection process. Static SNNs can become inaccurate if the robot or environment changes significantly. Inaccurate SNNs would need to be retrained with data obtained from the changed robotic system.

SNNs combined with a bi-directional approach alleviates some of the disadvantages inherent in both the static SNNs and physics-based simulation approaches. The current author proposed the Bootstrapped Neuro-Simulation (BNS) approach [Woodford *et al.*,

2016] in order to take advantage of SNNs in a bi-directional ER process. The BNS approach can be considered a model-based policy search algorithm where the learning process alternates between optimising robot simulators (models) and controllers (policies) in order to maximise an objective function [Kaushik, Chatzilygeroudis, and Mouret, 2018]. During the BNS approach, SNNs and controllers are concurrently optimised during the ER process. For the BNS approach, the ER process can begin before SNNs are fully trained.

The complexity of the robot morphology and controller design are important factors to consider when studying the SNS and BNS approaches. Most of the work investigating SNNs has been performed on simple wheeled robots [Pretorius, du Plessis, and Cilliers, 2009; Pretorius, du Plessis, and Gonsalves, 2014; Pretorius *et al.*, 2017; Woodford *et al.*, 2016]. Robot morphologies with many degrees of freedom are significantly more difficult to simulate. The complexity of the controller design can be measured in terms of the dimensionality of the solution space. Controller designs for complex robots often take into account expert domain knowledge, such as specially crafted mathematical functions, curve fitting procedures or pattern generating functions. For example, a Snake robot controller design could be based on a mathematical function that is able to generate joint angles mimicking biological snake locomotion [Melo, Hernandez, and Gonzalez, 2012]. Alternatively, the controller design could consist of low level commands that do not take advantage of any prior knowledge. Controller designs based on prior knowledge have a lower solution space dimensionality compared to those without.

If the controller design is simplified and relies on prior expert knowledge, the simulator design can also be simplified based on the controller design in order to reduce the complexity of SNNs [Woodford, du Plessis, and Pretorius, 2017]. However, the simulator would then only be compatible with the particular controller design and not generalisable for use with other controller designs. However, a simplified controller design can be used with a non-simplified simulator design.

Little work exists that investigates the scaling of SNNs for use on complex robot morphologies with high dimensional solution spaces. The SNS approach has been successfully demonstrated on a complex Hexapod robot using a controller design without prior knowledge [Pretorius *et al.*, 2019]. When considering complex robot morphologies, the SNS approach has only been investigated using a Hexapod robot platform. The BNS approach

has been successfully applied to a complex Snake robot but with a controller design based on prior knowledge [Woodford *et al.*, 2017]. No studies have investigated the BNS approach for complex robots using controller designs without prior knowledge. No work has investigated the BNS approach on a Hexapod robot platform.

1.2 Research Objectives

The main objectives of this Thesis are:

1. Identify relevant trends in the existing ER literature.
2. Identify shortcomings of existing ER approaches.
3. Demonstrate the scalability, generality, effectiveness and feasibility of the SNS and BNS approaches on complex robots using controllers that do not utilise prior knowledge.
4. Propose and investigate potential improvements to the SNS and BNS approaches.
5. Determine the relative advantages and disadvantages of the SNS and BNS approaches.

1.3 Methodology

Certain measurements are used in this work in order to quantify properties mentioned in the previous section, such as *Scalability*, *Generality*, *Effectiveness* and *Feasibility* of the SNS and BNS approaches studied in this research. The SNS and BNS approaches have mainly been studied on simple robot morphologies and have not been rigorously shown to scale well to more complex robot morphologies. *Scalability* is defined in terms of how well an approach is able to develop viable controllers on complex robots and high dimensional controller designs. *Generality* is the viability of an approach to develop viable solutions on different classes of complex robots. *Effectiveness* is measured in terms of the how well solution controllers perform a given task. *Feasibility* is measured in terms of the amount of behavioural data required and time taken to evolve viable solution controllers.

The purpose of this research is to investigate the use of SNNs on complex robot morphologies and controller designs without built-in prior knowledge. This work develops proof-of-concept prototypes that are evaluated through a series of experiments. An analysis of the experimental results helps demonstrate the scalability, generality, effectiveness and feasibility of the SNS and BNS approaches on the chosen robotic platforms.

The positivist approach gains knowledge through the use of empirical evidence instead of introspection or intuition [Easterbrook, Singer, Storey, and Damian, 2008]. This ensures that any researcher bias is kept to a minimum by objective observations of reality. A positivist approach will be taken in this study because it is believed the research objectives can be answered through an empirical study.

The deductive approach is focused on testing a particular hypothesis or theory as opposed to the inductive approach which involves discovering a theory based on available data [Gray, 2009]. This study uses a deductive approach to test hypotheses in order to confirm the theory that complex robots with high dimensional controllers can be successfully evolved using the SNS and BNS approaches. Hypotheses regarding the accuracy, generality, effectiveness, feasibility and scalability are judged through experimentation.

A thorough literature review of related work is conducted. Existing ER approaches are identified and their particular contributions to the field are noted. The literature presents known problems inherent in using existing ER approaches and current solution strategies are discussed. The literature study helps elucidate how this research fits into the existing body of knowledge (Objectives 1 and 2).

Few existing ER approaches in the literature demonstrate viability on more than one complex robot morphology. In this work, complex robots are defined to have more than 11 degrees of joint freedom. All controller designs in this work are purposely not simplified and do not rely on significant levels of human expertise. This research purposely investigates the SNS and BNS approaches on two complex robot morphologies. These robots produce completely different classes of behaviour compared to each other. The first robot used in this research is a limbed, walking Hexapod robot while the second is a limbless, crawling Snake robot. For any particular ER approach, demonstrating viability on either robot indicates that said approach can scale for use on certain complex robot morphologies (Objective 3). Demonstrating viability on more than one complex

robot where robots exhibit significantly different classes of behaviour indicates that an ER approach can generalise well (Objective 3).

The effectiveness and feasibility of each approach is judged based on experimental observations (Objective 3). Various proposals for improvements to the SNS and BNS approaches are investigated. Proposed improvements include changes to the simulator, the inclusion or exclusion of simulator noise, controller resetting, simulator resetting and behavioural data sampling strategies (Objective 4). Experimental results are studied quantitatively and qualitatively for the SNS and BNS approaches (Objectives 3, 4 and 5). The accuracy and effectiveness of solutions produced by the SNS and BNS approaches are compared to each other (Objective 5).

1.4 Dissertation Layout

The structure of this dissertation is illustrated in Figure 1.1. The background, problem statement and related research is presented in Chapter 2. The research objectives and methodology used are discussed in Chapter 3. Two different classes of robots are used in the experimental work. For the Hexapod robot, the SNS and BNS approaches are studied in Chapters 4 and 5, respectively. The Snake robot is utilised in Chapters 6 and 7 for experimental work related to the SNS and BNS approaches, respectively. Lastly, conclusions are drawn in Chapter 8.

Each chapter is detailed below:

Chapter 2 An introduction to the related work is presented. Existing ER approaches that are particularly relevant to this research are discussed and compared. In particular, prior work investigating the SNS and BNS approaches is presented.

Chapter 3 Preliminaries to the experimental work and the methods used to investigate the SNS and BNS approaches are discussed. Detailed explanations of the experimental procedures, data acquisition methods and experimental hardware is covered in this chapter. Proposed improvements to the SNS and BNS approaches are also outlined.

Chapter 4 This chapter covers experimental work related to the SNS approach, performed on the Hexapod robot. The best SNN architectures for the Hexapod robot are

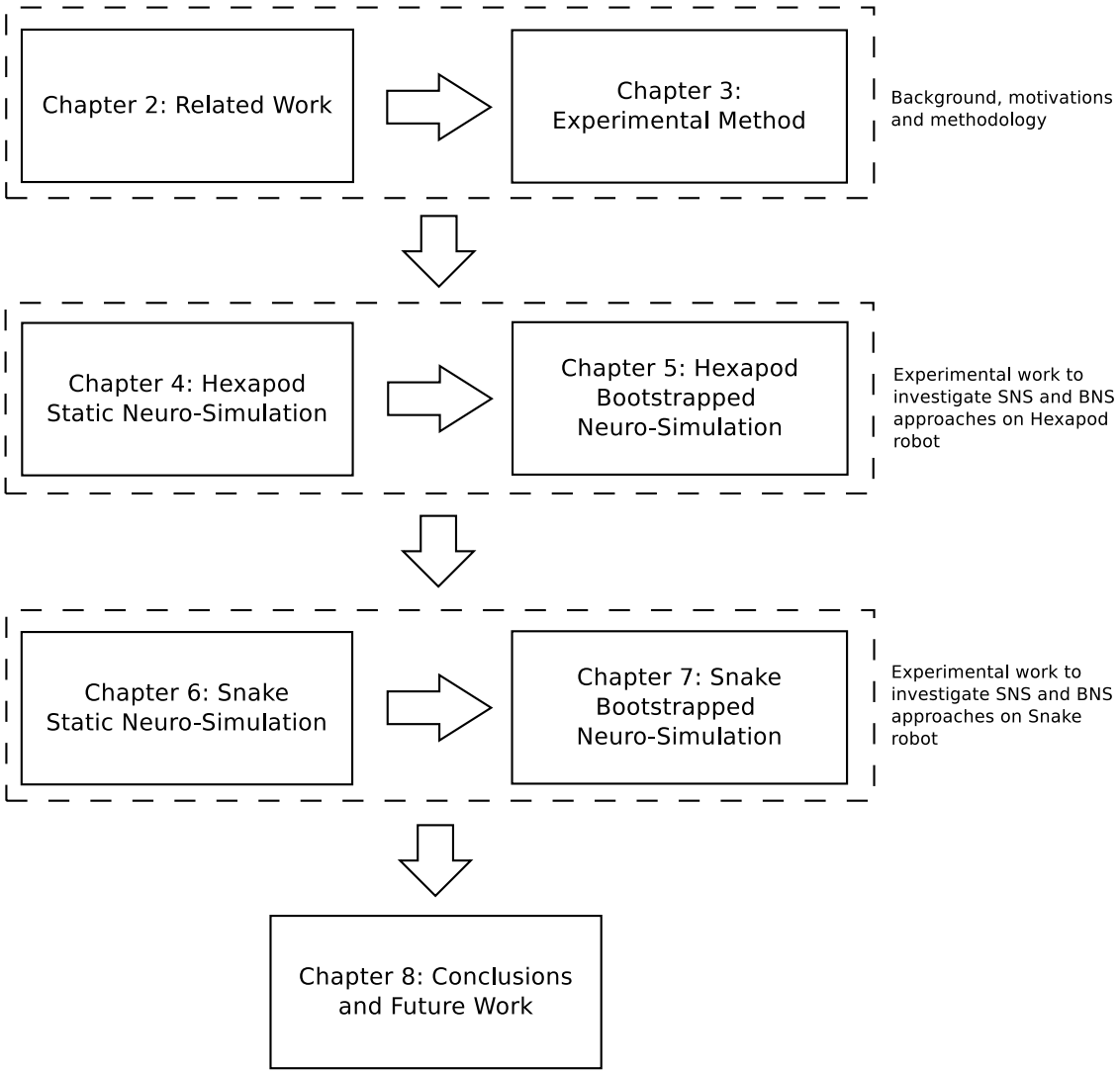


Figure 1.1: Structure of dissertation

chosen based on prior work. Improvements to the SNS approach are proposed and investigated through a series of experiments. Quantitative and qualitative results produced by the experiments performed in this chapter are presented and discussed.

Chapter 5 The experimental work related to the Hexapod robot and the BNS approach are covered in this chapter. Proposed improvements to the BNS approach are investigated through a series of experiments. Results related to the experimental work performed in this chapter are also presented and discussed.

Chapter 6 The experimental work specific to the Snake robot and the SNS approach are covered in this chapter. Benchmarks are conducted in order to identify the best SNN architectures to use when simulating the Snake robot. Improvements to the SNS approach are proposed and tested through a series of experiments. Results related to this chapter are presented and discussed.

Chapter 7 This chapter involves investigations specific to the Snake robot and the BNS approach. Proposed improvements to the BNS approach are studied through a series of experiments. Results related to the experimental work in this chapter is presented and discussed.

Chapter 8 The overall results to this study are summarized and conclusions are drawn. The research contributions in this work is described, limitations are discussed and possible future work is suggested.

Chapter 2

RELATED WORK

2.1 Introduction

This chapter introduces the background knowledge specific to the study. Prior studies related to this research are also outlined. The Evolutionary Robotics (ER) approaches investigated in this thesis necessitate a basic understanding of ANN theory (Section 2.2). Background knowledge and procedures followed during the ER process are covered in Section 2.3.

ER approaches are typically studied using wheeled or limbed robot morphologies. Swimming, flying and limbless robots are the least well studied morphologies in the ER field. The study of ER approaches using a Snake robot morphologies is therefore of particular interest (Section 2.4).

Prior work related to integrating simulators into the ER process are of particular interest. Many ER approaches have been designed to automatically improve or augment an existing simulator during the ER process. A literature review covering ER approaches that integrate simulators into the ER process are covered in Section 2.5. A high level comparison of ER approaches particularly relevant to this work are presented in Section 2.6. Finally, conclusions to the chapter are covered in Section 2.7.

2.2 Artificial Neural Networks

Biological brains can process complex information and generate fine motor control which has led researchers to develop computational models of biological neural systems [Engelbrecht, 2007]. These computational models, called Artificial Neural Networks (ANNs), have been demonstrated to possess excellent processing and control capabilities [Maren, Harston, and Pap, 2014; Pretorius *et al.*, 2017]. ANNs consist of many computational units called Artificial Neurons (ANs). Theory related to ANs is covered in Section 2.2.1. The combination of many ANs to form ANNs is discussed in Section 2.2.2. The ANN training process optimises weight parameter settings of an ANN in order to model a given system (Section 2.2.3). Specific trade-offs, features and improvements that are relevant to ANNs are discussed in Sections 2.2.4, 2.2.5 and 2.2.6.

2.2.1 The Artificial Neuron

A representation of an AN is given in Figure 2.1. ANs are mathematical functions based on biological neurons [Engelbrecht, 2007]. An input vector of numeric features $(x_1, x_2, ..., x_n)$ is taken as input to the AN and produces a single numeric output y, where n represents the number of input features. Each input feature x_i has a corresponding weight w_i (for $i = 1, 2, 3, ..., n$). All weights, inputs and outputs are real numbers. Input features are multiplied by their corresponding weights which weaken or strengthen input signals. A constant bias input node (x_{n+1}) is usually assigned the value -1 and has an associated weight (w_{n+1}). A bias is needed to effectively model certain phenomena [Engelbrecht, 2007]. The weighted sum of the input features is expressed as the *net* (equation (2.1)). An activation function takes the *net* summation as input and outputs a single result (equation (2.2)).

$$net = \sum_{i=1}^{n+1} x_i w_i \tag{2.1}$$

$$y = f(net) \tag{2.2}$$

Activation functions are important for modelling non-linear systems [Russell and Norvig, 2016]. The Sigmoid, Hyperbolic tangent, Linear and Rectified Linear Unit (ReLU) func-

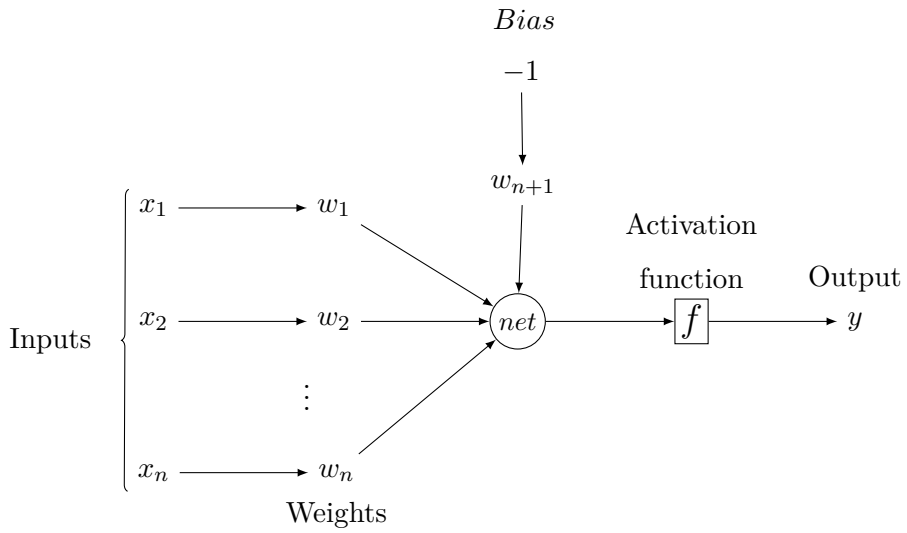


Figure 2.1: Artificial Neuron

tions are examples of commonly used activation functions. The ReLU activation function was popularised after 2010 when it was shown to outperform many traditional activation functions in terms of computational efficiency, convergence and sparsity [Glorot, Bordes, and Bengio, 2011; Nair and Hinton, 2010] (Section 2.2.3). A graphical representation of the ReLU function is given in Figure 2.2. The ReLu activation function (equation (2.3)) receives as input any real number x, returns zero for any negative values of x or outputs x for positive values of x.

$$f(x) = \max(0, x) \tag{2.3}$$

2.2.2 Neural Networks

ANNs are commonly used to solve classification and regression problems. Regression measures the relationship between output and input variables where outputs are numerical. Classification is used to categorise input variables.

An ANN consists of many connected ANs. Many different types of ANN topologies exist. A common topology is called the Feed-Forward Neural Network (FFNN) and is illustrated in Figure 2.3. Circular nodes represent ANs and arrows are the connections between nodes. The illustrated topology consists of three layers. Namely, the input,

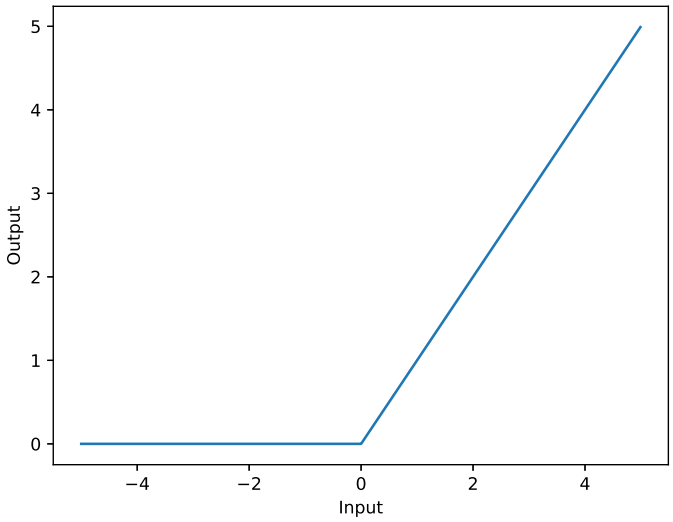


Figure 2.2: Rectified Linear Unit (ReLu) activation function

hidden and output layers. However, FFNNs can contain zero or many hidden layers. Hidden layers are required to accurately model many complex phenomena [Engelbrecht, 2007].

The input layer receives inputs and the computed signals propagate through the hidden layer to the output layer. No signals propagate backwards towards previous layers in FFNN topologies. The input layer receives I input values, $x_1, ..., x_i, ..., x_I$, plus a bias input x_{I+1}. Inputs are multiplied by their associated weights ($v_{j,i}$) and the results serve as inputs to the hidden layer neurons $y_1, ..., y_j, ..., y_J$. For each hidden layer neuron, the *net* (equation (2.1)) is calculated and activation functions are applied (equation (2.2)). Similarly, the signals from the hidden layer neurons are multiplied by their associated weights ($w_{k,j}$) and the results serve as input to the output layer neurons $o_1, ..., o_k, ..., o_K$, *net* summations are calculated and activation functions are applied.

The output for neuron o_k, given input values $x_1, ..., x_i, ..., x_I$ is calculated using equation (2.4). The activation functions for neurons o_k and y_j are represented as f_{o_k} and f_{y_j}, respectively.

$$o_k = f_{o_k}(\sum_{j=1}^{J+1} w_{k,j} f_{y_j}(\sum_{i=1}^{I+1} v_{j,i} x_i)); \quad 1 \leq k \leq K \tag{2.4}$$

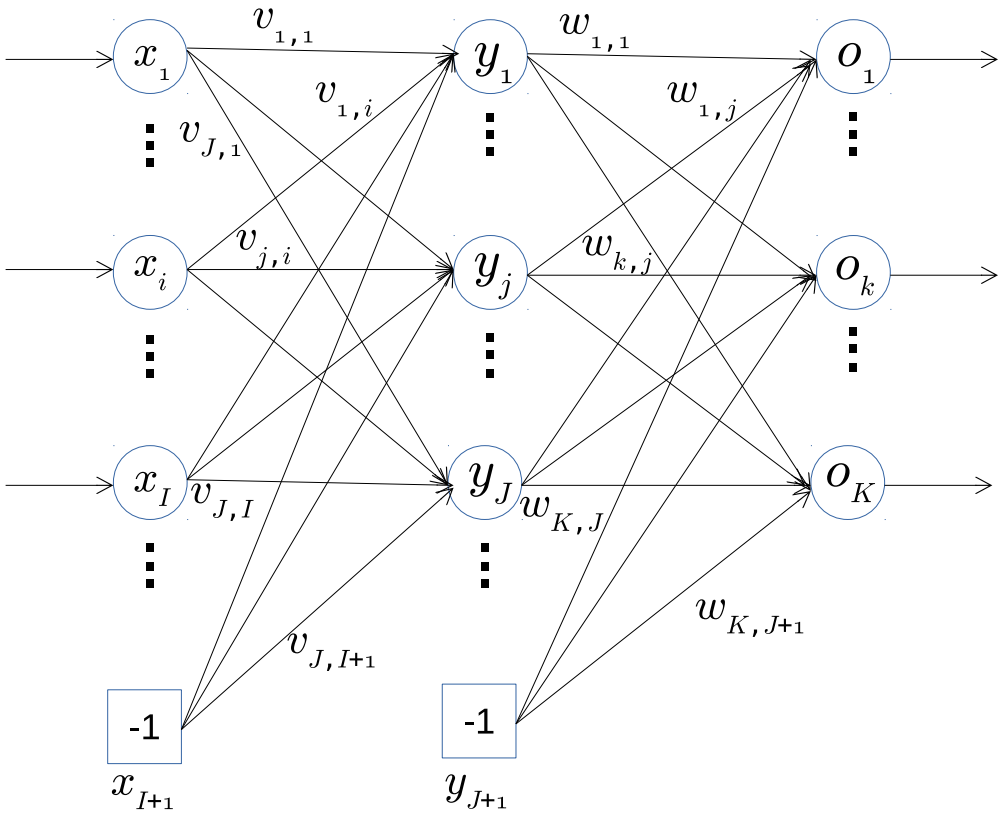


Figure 2.3: Artificial Neural Network

Explicit mathematical models based on human understanding are only viable when a system is well understood. ANNs can model systems where the complexity has increased to the point where human understanding is difficult or time-consuming to achieve. Machine learning approaches, such as ANNs are advantageous when the relationship between the input and output variables is difficult to establish through expert knowledge. A strong statistical or mathematical background is usually not required in order to use ANNs effectively.

Researchers typically describe ANNs as black-boxes due to the difficulty in interpreting their behaviours. Gaining insights into the inner workings of underlying systems being modelled using ANNs can be challenging. An ANN is essentially a mathematical function that maps an input space to an output space.

Systems with a higher number of input features are more difficult to model compared to systems with fewer inputs. As the dimensionality of an input space increases, the volume of the input space increases exponentially. An exponential increase in the volume of the input space eventually leads to a point where available data becomes sparse. This phenomena is referred to as the *curse of dimensionality* [Bellman, 1961; Domingos, 2012]. High dimensional search spaces could require significant amounts of training data and may suffer from over-fitting. Over-fitting occurs when a model memorises the training data such that the model performs poorly on unseen data [Engelbrecht, 2007].

Deep Learning involves models that consist of multiple stages or layers of nonlinear information processing and learning happens using supervised or unsupervised methods [Deng and Yu, 2014]. The field of Deep Learning is a relatively new research area [Bengio, 2009]. Advances in Deep Learning have greatly impacted the Machine Learning and Artificial Intelligence fields. Recent advancements includes better activation functions [Glorot *et al.*, 2011], stochastic backpropagation [Ruder, 2016], multi-hidden-layer training [Huang, Sun, Liu, Sedra, and Weinberger, 2016], regularisation techniques [Srivastava, Hinton, Krizhevsky, Sutskever, and Salakhutdinov, 2014] and Open Source Deep Learning libraries [Abadi, Agarwal, Barham, Brevdo, Chen, Citro, Corrado, Davis, Dean, and Devin, 2015; Chollet, 2015]. Approaches developed in this thesis have greatly benefited from advances in the Deep Learning field. It is doubtful that the approaches developed in this thesis would be viable without recent advances in the Deep Learning field.

2.2.3 ANN Training

ANN training is an iterative process that optimises the weights of an ANN in order to approximate some function. The optimisation process relies on many input-output pair examples, called the training dataset. A single input-output pair is commonly referred to as a pattern.

Pre-processing steps are often applied to all patterns before training. Training pattern inputs and target outputs can be standardised or normalised. Standardisation is where a feature/output variable is transformed so that they have a zero mean and unit standard deviation. Normalisation is where a feature/output variable is transformed to be between the domain of -1 and 1 or between 0 and 1.

ANN weights are optimised by reducing differences between the training dataset's ideal outputs and the ANN's predicted outputs. The difference between the training dataset outputs and the ANN predicted outputs is measured as the Mean Squared Error (MSE), also known as the cost or loss function. The MSE is the average of the squares of errors (equation (2.5)). The squaring of errors results in larger errors being significantly more influential in the calculation.

$$MSE = \frac{1}{P}\sum_{p=1}^{P}\sum_{k=1}^{K}(t_{k,p} - o_{k,p})^2 \tag{2.5}$$

where P is the total number of training patterns. The total number of output neurons is represented as K. The predicted output for the k^{th} output neuron and p^{th} training pattern is shown as $o_{k,p}$. The ideal output for the k^{th} output neuron and p^{th} training pattern is represented by $t_{k,p}$.

Gradient Descent (GD) is a common algorithm for minimising the cost function. Initially, ANN weights are initialised with randomly generated values. Weights are iteratively updated based on the derivative with respect to the weights of the cost function (gradient). Gradients are multiplied by a learning rate and added to the weights in the direction of minimising the cost function. New gradients are then calculated and weights updated based on the new gradients. This process repeats until some stopping condition is met.

GD calculates gradients using the entire training dataset. A variation on the GD algorithm is called Stochastic Gradient Descent (SGD). SGD calculates gradients using a

subset sample of the full training dataset. Gradient calculations based on a small number of training patterns makes SGD less computationally expensive than GD. SGD often converges faster than GD depending of the function being modelled.

The learning rate applied to gradients in GD and SGD can be static or adjusted during the training process. A popular extension of the SGD algorithm implements adaptive learning rates and is called Adaptive Moment Estimation (Adam). A dynamic learning rate is maintained for each weight parameter setting. Learning rates are computed from the first and second moments of the gradients. Adam was first introduced in 2014 and is simple, computationally efficient, scalable to many types of problems and requires minimal parameter tuning [Kingma and Ba, 2014]. The Adam optimisation algorithm has also been demonstrated to outperform many other ANN training algorithms [Ruder, 2016].

The training process can result in a problem called model over-fitting. This occurs when the errors between the predicted and ideal outputs are low for the training dataset but new patterns not presented during training have high errors. Over-training can result in ANNs memorising the training data but lose the ability to generalise to unseen data. Over-fitting can be avoided by continually monitoring an ANNs generalisation ability during training. Available patterns can be divided into two separate groups, namely, the training and validation datasets. The validation set is used to measure the generalisation ability of the ANN during training. Once the errors between predictions and the validation dataset start to increase during training, the training process should be stopped early in order to prevent over-fitting.

In 2014, Dropout was proposed as a technique to help ANNs avoid over-fitting [Srivastava *et al.*, 2014]. During training, hidden layer neurons are temporarily and randomly turned on or off along with their connections. The dropout rate is the probability of any given hidden layer node being ignored during feed forward calculations. This prevents the modelling process from relying too heavily on a small subset of neurons and improves an ANN's generalisation ability.

The number of hidden layers and neurons for a particular ANN morphology is directly proportional to the complexity of the phenomena being modelled. For many problems, it is difficult to determine the optimum ANN morphology [Russell and Norvig, 2016]. A trial and error approach is often used in practice for selecting a good ANN architecture. ANNs

can increase in complexity by either adding more neurons or increasing the number of hidden layers. More hidden layers increases the capacity of an ANN by creating complex mapping functions and forming abstractions of the underlying relationships. Morphologies with an insufficient level of complexity will fail to adequately model a given phenomena while too much complexity increases the likelihood of over-fitting.

The vanishing gradient problem [Bengio, Simard, and Frasconi, 1994] occurs when training ANNs using gradient-based optimisation techniques and gradient information is lost when many hidden layers are used. Computing gradients over multiple hidden layers can produce gradients that become vanishing small and can prevent or slow down ANN training. The vanishing gradient problem can be solved using alternative activation functions, better weight initialisations and increasing the training time. ANNs that use the ReLU activation function are less prone to suffering from the vanishing gradient problem compared to other activation functions.

2.2.4 Uncertainty Estimation

ANNs can make point predictions on unseen observation, however, ANNs do not inherently indicate the reliability of predictions. Bayesian Neural Networks and other Bayesian modelling techniques create models that produce uncertainty information but these modelling techniques can be prohibitively slow for high dimensional problems. Techniques have been developed for providing uncertainty for standard ANNs [Gal and Ghahramani, 2016; Lakshminarayanan, Pritzel, and Blundell, 2017].

Figure 2.4 illustrates predictions from an ANN with dropout enabled for a simple regression problem. ANNs with dropout enabled produce a confidence interval. These confidence intervals can then be used to estimate the reliability of any given ANN prediction. The grey line represents the ideal values. The blue line represents the ANN predictions. The only known observations are the black dots. The shades of blue represent the distribution of predictions from an ANN with dropout enabled. The standard deviations of ANN predictions close to known observations is low compared to the standard deviation of predictions far from known observations. Alternatively, confidence intervals can be estimated using ensembles of ANNs trained on the the same dataset, producing a distribution of predictions for any given input [Lakshminarayanan *et al.*, 2017].

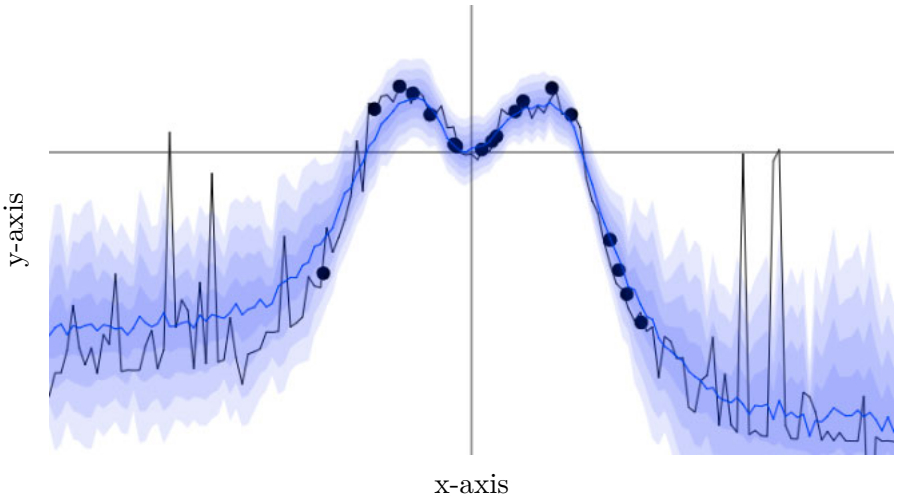


Figure 2.4: ANN predictions with dropout enabled for a simple regression problem [Gal and Ghahramani, 2016]

2.2.5 Bias-Variance Tradeoff

If multiple models are trained on independently collected datasets for a given system, each model will likely produce a different prediction for any given set of inputs. Errors can be classified in terms of bias and variance. In Figure 2.5, the target centres represent perfect model predictions. Predictions farther or away from the bulls-eye indicate larger errors. Each hit on the target represents an independently trained model. Ideally, a low bias and low variance is achieved.

A trade-off exists between reducing bias and variance. Bias measures the difference between the average of predictions and the correct value. The variance is the degree to which independent model predictions vary from each other. A high variance is the result of overfitting.

Bias and variance can be understood in terms of over-fitting and under-fitting (Figure 2.6). Increasing a model's complexity increases variance and reduces bias. A complex model tends to memorise the training dataset, leading to a low bias while the model is unable to generalise on unseen data (high variance). A balance between bias and variance is achieved by developing models with different levels of complexity and comparing the errors. A high bias can be identified when observing a high training error and the validation

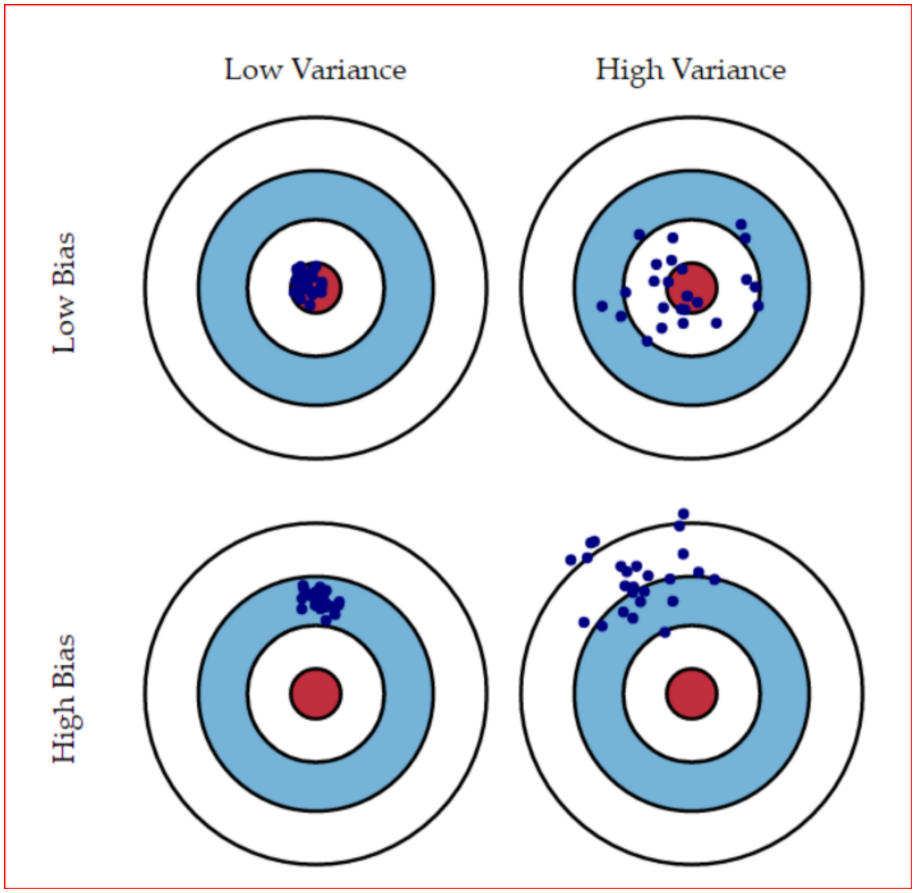


Figure 2.5: Graphical illustration of bias and variance [Fortmann-Roe, 2012]

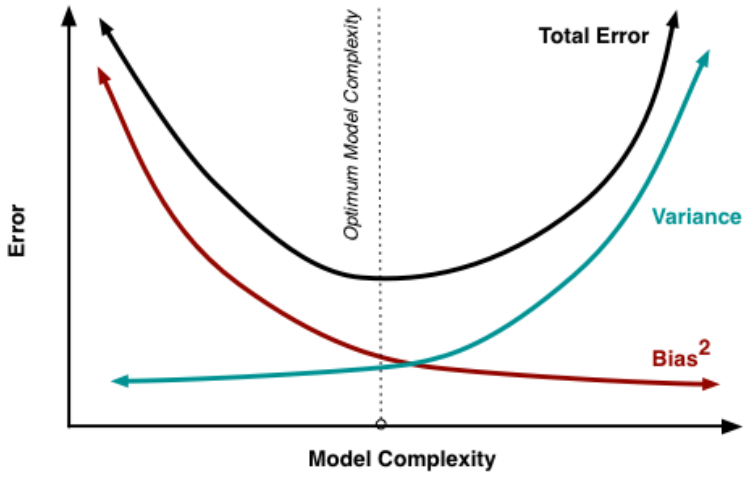


Figure 2.6: Bias and variance contribution to total error [Fortmann-Roe, 2012]

error is similar to the training error. A high variance is identified when a low training error is achieved but the validation error is high.

2.2.6 Ensembles

ANNs are trained using stochastic algorithms which leads to differences in predictions for different weight initialisations. If many identical ANN architectures are initialised with different weight settings and trained on the same dataset, each trained ANN may produce a significantly different mapping between inputs and outputs which can produce a high variance problem in predictions. One method for reducing the variance is to train multiple ANNs and combine the predictions, called ensemble learning. Each ANN is not likely to make the same mistakes when calculating predictions on unseen patterns. ANNs configured in ensembles helps reduce the variance of predictions and tends to perform better than any single ANN [Goodfellow, Bengio, and Courville, 2016].

2.3 Evolutionary Computation

Evolutionary Computation (EC) is a family of algorithms within the Computational Intelligence field that involves finding solutions to discontinuous and continuous optimisation problems using computation models and evolutionary processes [Engelbrecht, 2007]. Evo-

lutionary Algorithms (EA) is a subset of EC and involve algorithms based on natural evolution. In EA, a population of candidate solutions is evolved over many generations through biological mechanisms such as reproduction, selection, recombination and mutations. Solution candidates, referred to as individuals, survive or die based on evolved characteristics important for solving a particular problem. Fitter individuals are more likely to reproduce and pass on useful characteristics to their children.

2.3.1 Evolutionary Algorithms

An EA is a population-based optimisation algorithm that uses evolutionary principles in order to perform a stochastic search over a solution space. The pseudo-code of a generic EA is shown in Algorithm 1 [Engelbrecht, 2007].

An initial population of randomly generated individuals is created. Each individual consists of genes that encode traits that affect its characteristics or behaviours. An individual's ability to perform the chosen goal task is measured using a fitness function.

Algorithm 1 The Evolutionary Algorithm

Let generation $t = 0$
Initialize a population, $C(0)$, of n individuals
while no stopping condition(s) are met **do**
  Evaluate the fitness of each individual in $C(t)$
  Create offspring through reproduction operators
  Use selection operators to create a new population $C(t+1)$
  Advance to the next generation: $t = t + 1$
end while

All individuals in the population are evaluated and assigned a fitness value. Children are produced from the population by means of reproduction operators. Reproduction involves crossover operations between parents. Parents are selected for crossover operations through selection methods that favour higher fitnesses.

After the reproduction process, offspring can be mutated in order to introduce small changes to their genes and help increase diversity [Beasley, Martin, and Bull, 1993]. The probability that a particular gene will be mutated is called the mutation rate. In order to

avoid distorting existing good solutions, mutations are small and the mutation rate should not be too high. After the mutations are applied to the offspring, the current population is discarded and the offspring become the new population. Some of the best performing individuals from the old population can carry forward to the next generation. This process continues for many generations until some stopping condition is met.

The EA can terminate if certain convergence criteria have been met like no significant improvement can be found, there can be a limit to the total number of generations or the EA could terminate when an adequate solution is found.

2.3.2 Evolutionary Robotics

Evolutionary Robotics (ER) uses Darwinian principles in order to evolve a population of robotic controllers and/or robot morphologies [Floreano *et al.*, 2008]. Controllers receive input from robotic actuators and sensors, processes them and then produces a response as output, such as motor movements or changing a joint position. In order to produce controllers that generate target robot behaviours or solve goal tasks, controller settings need to be optimised using evolutionary techniques, such as Evolutionary Strategies [Schwefel, 1993], Evolutionary Programming [Koza, 1992] and Genetic Algorithms [Pratihar, 2003]. ER algorithms are not limited to evolving controllers but can also evolve robot morphologies [Lund, 2003].

The procedure followed for performing the ER process can be found in Section 2.3.2.1. A high level discussion regarding the current state of the ER field is given in Section 2.3.2.2.

2.3.2.1 Evolutionary Robotics Process

The ER process applied to an ANN-based controller is illustrated in Figure 2.7. An initial population of controllers is randomly generated. The initial population is randomly generated but future generations are created through a reproduction process (**A**). Individuals are made up of genes that are the weight parameters to a controller (**B**). The decoded genes are used to construct the actual ANN used to control robot behaviours (**C**). The example uses an ANN-based controller but other types of controllers are possible. The decoded individuals are evaluated on a real-world robot or behaviours are approximated

in simulation. Based on observed or simulated behaviours, fitness scores are assigned to individuals (**D**). Once all individuals have been evaluated and assigned a fitness, a new population of individuals is created through cross-over and mutation operators (**E**). This cycle is repeated for many generations until some termination state is reached.

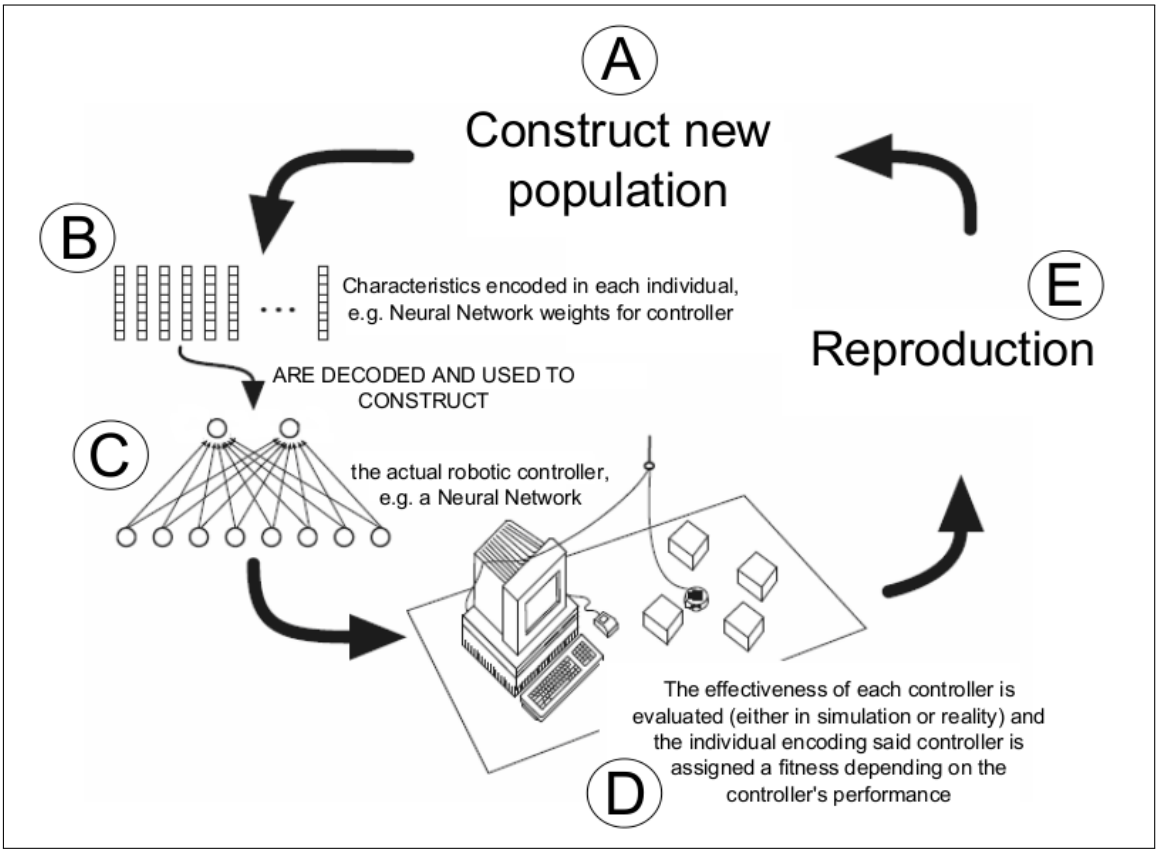


Figure 2.7: The Evolutionary Robotics Process applied to the evolution of an ANN based controller [Floreano *et al.*, 2008; Pretorius, 2010]

2.3.2.2 General State of the Evolutionary Robotics Field

Manually creating robot controllers becomes less feasible as tasks and robotic systems become more complex [Sofge, Potter, Bugajska, and Schultz, 2003]. A goal in ER is the automatic synthesis of control solutions and/or design of body plans for the specified task [Silva, Duarte, Correia, Oliveira, and Christensen, 2016]. ER research can be broadly divided into two primary categories. The first category involves cognitive science [Auerbach and Bongard, 2014; Harvey, Paolo, Wood, Quinn, and Tuci, 2005] and the second focus is

the study of ER as an engineering tool.

ER and neuroscience researchers can collaborate by study biologically inspired models of particular environments. A biological model of a target system can be developed and studied using ER techniques. For example, a central pattern generator of the swimming behaviours of Lamprey eels have been studied using ER techniques [Hallam and Ijspeert, 2003]. Experimental studies in biological evolution is usually practically impossible due to the long time periods required to observe evolution in a natural environment [Doncieux, Bredeche, Mouret, and Eiben, 2015]. Simulated versions of biological evolutionary processes is often the only viable method of investigation. ER has been used to investigate the evolutionary basis for altruism as a survival strategy [Montanier and Bredeche, 2011], species communication strategies [Wischmann, Floreano, and Keller, 2012], predator confusion through collective swarms [Olson, Hintze, Dyer, Knoester, and Adami, 2013] and how an environment can influence morphological complexity [Auerbach and Bongard, 2014].

For engineering applications, it is envisioned that ER approaches could replace inefficient manual controller/morphology development [Lipson and Pollack, 2000; Quinn, Smith, Mayley, and Husbands, 2003]. Traditional robotics involves human experts designing most aspects of a robotic system, such as the mechanics, sensors, control and morphology separately. These separate pieces are then put together to form the final solution. This manual process can be time-consuming, is subject to human bias, requires significant levels of specialised human knowledge and interventions. For example, the manual development of walking robot gaits can be replaced with an ER approach which can automate some of the controller development process [Hornby, Takamura, Yamamoto, and Fujita, 2005; Pretorius *et al.*, 2019]. Multiple aspects regarding a robotic system can be considered and optimised together as part of the ER process. Novel robot morphologies (modular robots [Zykov, Chan, and Lipson, 2007], soft robots [Cheney, Clune, and Lipson, 2014], swarms [Şahin, Girgin, Bayindir, and Turgut, 2008]) can be discovered and the solution spaces explored using ER approaches [Bongard *et al.*, 2006b].

ER techniques are of particular interest for the development of more autonomous and robust robots. For complex robotic systems, it is practically infeasible to account for all possible failure modes or changes in a target environment. ER approaches can be used to

automatically detect and recover from hardware damage or changes to the environment [Bongard, Zykov, and Lipson, 2006a; Cully *et al.*, 2015]. This would be particularly useful in scenarios where human interventions are difficult, such as robots operating on other planets or dangerous environments.

The ER field is having difficulties in scaling up current ER approaches and undertaking more complex problems. This is arguably one reason for the lack of mainstream adoption of ER techniques [Silva, Duarte, Oliveira, Correia, and Christensen, 2014]. There is currently a significant gap between ER and mainstream robotics fields [Silva *et al.*, 2016]. The difficulties in scaling ER approaches has likely caused current ER research to be more centred around theoretical contributions rather than practical applications.

The Evolutionary Computing field has developed standard benchmarks against which new techniques can be tested and compared. However, the ER field lacks standardised benchmarks, robots or environments against which to investigate ER approaches [Doncieux *et al.*, 2015; Silva *et al.*, 2016]. This makes comparisons between ER approaches difficult. Few ER approaches are rigorously studied across different robots and tasks. The effectiveness of certain ER approaches across different robotic systems may not be readily apparent based on the current literature.

Simulators are used to avoid damaging real-world hardware and to speed up the ER process. However, developing an effective simulator can become a challenging problem (Section 2.5). Behaviours developed in simulation often do not transfer well into reality. A trade-off between simulator accuracy and computational efficiency is often required. Alleviating some of the problems inherent in using robotic simulators in ER is an active and important research area (Section 2.5.3).

2.4 Snake Robot Morphologies

A robot's morphology can greatly affect the controller design, optimisation and difficulty level of the problem. The chosen robot morphology greatly influences real-world robot behaviours. ANNs can be used to simulate behaviours from different robot morphologies based on observed behaviours.

Many Snake robot locomotion modes are based on biological snake behaviours such as

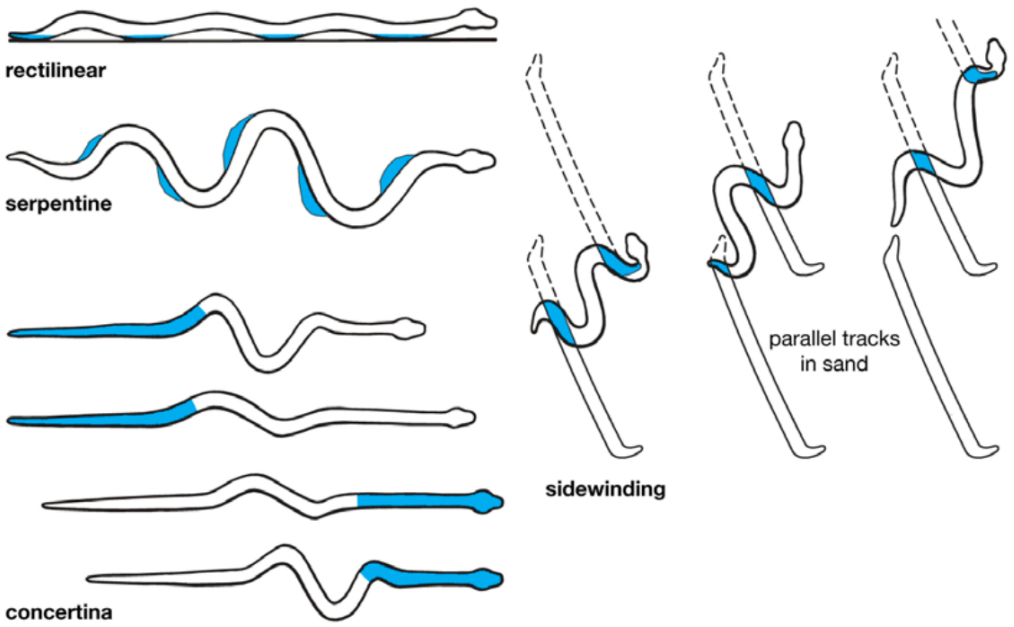


Figure 2.8: Snake locomotion modes
[Britannica, 2012]

slide-pushing, lateral undulation (serpentine locomotion), concertina, rectilinear motion, side-winding and various others [Dowling, 1996]. Some of these locomotion modes can be visualised in Figure 2.8. For biological snakes, lateral undulation is the most common locomotion mode used. Side-winding is when the snake body forms a sine-like wave while only two static contact points are made with the ground at any given time. Biological snake skins possess directional friction properties where forward movements are subject to less friction compared to moving backwards [Hu, Nirody, Scott, and Shelley, 2009]. Lateral undulation is less suitable for low-friction surfaces. Side-winding locomotion is capable of generating effective movements in low-friction environments [Shmakov, 2006].

Snake robot locomotion can be inspired by biological snakes. Figure 2.9 illustrates turning behaviours from a robot and biological snake where the robot is on a flat hard surface and the snake is on loose sand. Current Snake robot morphologies are unable to match the agility of current biological snakes [Gong, Travers, Astley, Li, Mendelson, Goldman, and Choset, 2016]. Non-biologically inspired locomotion modes include flapping and rolling motions. Robot Snake morphologies sometimes use wheels or directional friction mechanisms. Snake robots can be designed to operate in environments not suitable for

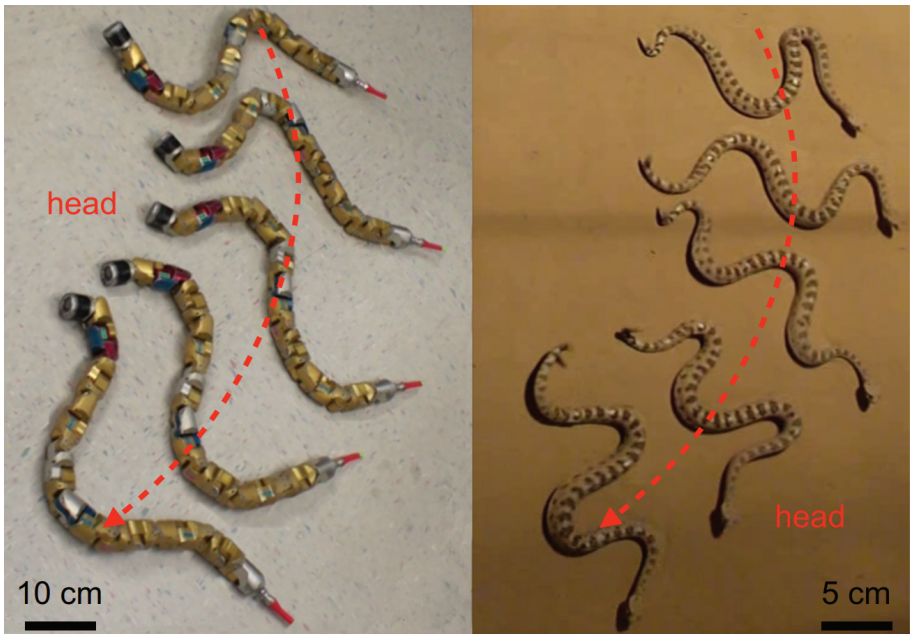


Figure 2.9: Snake robot (left) and rattlesnake (right) executing turning behaviours [Gong *et al.*, 2016]

wheeled motion, including tight spaces, swimming, climbing inside/outside pipes, stairs and many others [Melo *et al.*, 2012]. Robot Snakes without specialised friction generating mechanisms typically rely on locomotion modes similar to side-winding or sideways shifting [Woodford, du Plessis, and Pretorius, 2015].

The Snake robot morphology is theoretically appealing in terms of its potential but is difficult to use in practice. The study of most ER approaches on complex robots tend to be limited to limbed robot morphologies. Research into Snake-like robots in ER have been limited to relatively simple morphologies and problems [Au and Jin, 2016; Hasanzadeh and Akbarzadeh, 2013; Hasanzadeh and Tootoonchi, 2010; Kamimura, Kurokawa, Yoshida, Murata, Tomita, and Kokaji, 2005].

2.5 Simulators

As previously mentioned, simulators play a significant role in speeding up the ER process and avoiding problem inherent in evolving controllers in reality (Section 2.5.1). Background knowledge and prior research to consider when integrating simulators into ER are

covered in Section 2.5.2. Using simulators during the ER process can result in a mismatch between behaviours observed in simulation and reality. This is due to simulator inaccuracies or peculiarities that arise in simulation and not in reality. Many ER approaches have been designed, such that the simulator is improved or augmented during the ER process (Section 2.5.3). Most ER approaches make use of a physics-based simulator which come with significant disadvantages. SNNs do not suffer from many of the disadvantages seen in physics-based approaches. Prior research involving the use of SNNs in ER are covered in Section 2.5.4.

2.5.1 Evolving Controllers in Reality

Evaluating large numbers of controllers on real-world hardware can become infeasibly time-consuming and may result in significant damage to the robot through mechanical wear [Floreano and Mondada, 1994; Zagal and Ruiz-del Solar, 2007]. Evaluation times tend to increase as the complexity in a robotic system increases [Zagal and Ruiz-del Solar, 2007]. Erratic movements during the early stages of the ER process may damage robot hardware [Floreano *et al.*, 2008]. Frequent real-world controller evaluations can require significant manual human interventions, such as resetting the robot [Floreano *et al.*, 2008].

In order to escape the limitations of performing large numbers of real-world evaluations, controllers can be evaluated in simulation. The solution search space can be explored faster in simulation compared to reality. The number of real-world evaluations required to discover effective controller solutions can be reduced by using a simulator [Lund and Miglino, 1996].

2.5.2 Simulators in Evolutionary Robotics

A high level overview of the types of simulation strategies that can be followed are discussed in Section 2.5.2.1. Important aspects to consider when evolving controllers in simulation are discussed in Section 2.5.2.2. Lastly, simulators developed using Machine Learning techniques are covered in Section 2.5.2.3.

2.5.2.1 Simulation Strategies

Simulators can be constructed from expert knowledge, such as physics-based simulation techniques. Alternatively, a more machine learning approach can be followed and empirical models can be built from experimentally collected data. A hybrid simulation approach could combine different aspects of both an empirical and physics-based simulation approach.

Robotic systems can be modelled using either a knowledge-based or behavioural-based representation system, with varying levels in-between [De Nardi, 2010]. In knowledge-based representations, the underlying components governing a robotic system are modelled individually and integrated with one another, leading to clearly understood structures. Examples of components include kinetics, friction, gravity, weight distributions and many others. Knowledge-based representations require significant prior knowledge and human interventions to build. Behavioural-based representations ignore the direct modelling of underlying components and instead model behaviours directly. Little expert knowledge is required to model direct behaviours of a robot. However, many behavioural-based modelling techniques have the disadvantage of being black-boxes, whereas knowledge-based approaches have parameters and structures with real-world interpretations. As the complexity in robotic systems increase, the ability to construct simulators from human gained expert knowledge becomes more difficult.

The model taxonomy can be broken down in terms of the prior knowledge integrated into the simulator design [De Nardi, 2010]. Namely, simulators can be classified as white-box, black-box or grey-box models. White-box models, such as physics-based equations, are developed using specialised prior knowledge and physical insights. Certain machine learning approaches produce black-box models, such as ANNs. Grey-box models use both empirical data and prior knowledge.

2.5.2.2 Evolving Controllers in Simulation

There are challenges to overcome when using simulators in ER. If the ER process takes place in simulation only, solution controllers can develop behaviours in simulation that do not transfer well into reality, referred to as the *reality-gap* problem. A simulator design

could be oversimplified and lead to solutions that rely too heavily on simulated peculiarities that are absent in reality. Simulator inaccuracies are inevitable and even high fidelity simulators cannot perfectly model reality [Floreano *et al.*, 2008]. High fidelity simulators can reduce the *reality-gap* but can be computationally expensive to operate [Miglino, Nafasi, and Taylor, 1994; Mouret and Chatzilygeroudis, 2017]. Simulated controller evaluation times can increase substantially for more complex robotic system [Bongard, 2013; Matarić and Cliff, 1996].

The *reality-gap* problem can be reduced by injecting a reasonable amount of noise into simulated behaviours [Harvey, Husbands, Cliff, Thompson, and Jakobi, 1997; Jakobi *et al.*, 1995; Miglino *et al.*, 1995; Mouret and Chatzilygeroudis, 2017; Nolfi, Bongard, Husbands, and Floreano, 2016]. Small discrepancies between simulation and reality can accumulate over time and exacerbate the *reality-gap* problem. Even if the same controller is evaluated multiple times in reality, there are small differences in observed behaviours between evaluations. A simulator without noise does not account for the slight variations in behaviours seen in reality. Adding noise to simulated sensors and movements can prevent solution controllers from relying on peculiarities seen only in simulation. Noise can be added to a simulator by including sensor or movement data sampled directly from the real-world robot. Alternatively, a noise model or distribution can be estimated from empirically collected data. Injecting the correct type and level of simulator noise can be difficult and might require significant manual tuning [Mouret, Koos, and Doncieux, 2012]. As the complexity of a robotic system increases, it becomes increasingly difficult to inject large amounts of simulator noise [Bongard, 2013].

Physics engines such as the Open Dynamics Engine (ODE) [Smith, 2005], Bullet [C., 2019] and Dart [Lee, Grey, Ha, Kunz, Jain, Ye, Srinivasa, Stilman, and Liu, 2018] make use of algorithms for collision detection, rigid body dynamics and classical mechanics. Physics-based simulators have been used to evolve controllers that can transfer simulated behaviours into reality [Bongard and Lipson, 2004a; Jakobi *et al.*, 1995; Moeckel, Perov, Nguyen, Vespignani, Bonardi, Pouya, Sproewitz, van den Kieboom, Wilhelm, and Ijspeert, 2013; Pretorius *et al.*, 2019]. It is practically impossible to account for every possible physical component that makes up any given system, such as gravity, friction, kinetics, dynamics and many others. Developing a highly accurate simulator requires specialised knowledge,

can be time-consuming and financially costly [Floreano and Mondada, 1994; Wittmeier, Jäntsch, Dalamagkidis, and Knoll, 2011]. Significant resources can be spent evaluating existing simulation frameworks based on multiple criteria [Mouret and Chatzilygeroudis, 2017]. Simulation frameworks can become outdated quickly or are subject to significant changes over time [Mouret and Chatzilygeroudis, 2017].

In an attempt to reduce the complexities inherent in simulating real-world robotic systems, Jakobi [1998a] suggested that a *minimal simulation* be built. A simulation need not accurately simulate every component of a given system. Specific features of a system could be simulated while other aspects are randomly generated so that the ER process can only exploit predictable aspects. However, such an approach could require domain knowledge of the given robotic system and identifying the components to simulate may be difficult [Floreano *et al.*, 2008].

2.5.2.3 Simulator Development using Machine Learning Techniques

Simulation techniques that do not require significant expert knowledge of a given robotic system are possible using empirical models. Experimental data can be collected from a robotic system and standard regression or clustering methods used to build accurate behavioural models [De Nardi and Holland, 2008; Kamio and Iba, 2004; Pretorius *et al.*, 2019; Togelius, De Nardi, Marques, Newcombe, Lucas, and Holland, 2007; Woodford and du Plessis, 2018]. Empirical models can be built using lookup tables or interpolation techniques [Lund and Miglino, 1996]. More advanced modelling techniques include NARMAX polynomials [Nehmzow, Kerr, and Billings, 2009], Gaussian Processes [Lizotte, Wang, Bowling, and Schuurmans, 2007], ANNs [Lee, Nehmzow, and Hubbold, 1998, 1999; Nakamura, Saegusa, and Hashimoto, 2007; Pretorius *et al.*, 2017; Togelius *et al.*, 2007; Woodford *et al.*, 2015] and Genetic Programming [De Nardi and Holland, 2008; Schmidt and Lipson, 2009]. Most empirically developed simulators are for relatively simple robots with few degrees of freedom [De Nardi, 2010; De Nardi and Holland, 2008; Kamio and Iba, 2004, 2005; Lund and Miglino, 1996; Togelius *et al.*, 2007; Woodford *et al.*, 2016], however, recent work has demonstrated that empirical simulation approaches can be scaled up to more complex robots [Pretorius *et al.*, 2019; Woodford *et al.*, 2017].

2.5.3 Bidirectional Simulation Development

The importance of bidirectional ER approaches are covered in Section 2.5.3.1. The following sections discuss particular implementations of bidirectional ER approaches. Physics-based model parameters can be manually tuned through trial-and-error experimentation. Alternatively, model parameters and structures can be automatically optimised using the Anytime Learning Algorithm (Section 2.5.3.2), Estimation-Exploration Algorithm (Section 2.5.3.3) or the Back to Reality Algorithm (Section 2.5.3.4). Instead of improving an existing simulator, weaknesses could be identified and the ER process could evolve controllers that avoid unreliably simulated behaviours (Section 2.5.3.5). An existing simulator can be augmented by some mapping procedure, such as Intelligent Trial-and-Error Learning (Section 2.5.3.6). Many approaches require the development of a physics-based simulator based on prior knowledge. If a Machine Learning strategy is followed, empirical data can be collected and used to automatically model a given robotic system without specialised prior knowledge (Section 2.5.3.7).

2.5.3.1 Bidirectional Mechanism

The *reality-gap* problem can be avoided by evaluating all controllers in reality [Floreano and Mondada, 1994; Hornby *et al.*, 2005; Lipson, Bongard, Zykov, and Malone, 2006]. Alternatively, the ER process can initially evaluate controllers in simulation and later switch over to evaluating controllers in reality [Pollack, Lipson, Ficici, Funes, Hornby, and Watson, 2000].

The quality of solution controllers evolved in simulation is assessed through real-world evaluations. If all solutions fail to transfer well into reality, improvements to the simulator or fitness function could be necessary. The ER approaches described up to this point do not include a built-in mechanism for improving or augmenting the simulator during the ER process. Empirical data can be collected during the ER process for simulator improvement or augmentations that leads to the discovery of more transferable controllers. Bi-directional mechanisms to the ER process have been developed with feedback loops for reducing the *reality-gap* [Bongard *et al.*, 2006b; Cully *et al.*, 2015; De Nardi and Holland, 2008; Grefenstette and Ramsey, 1992; Mouret *et al.*, 2012; Zagal and Ruiz-del Solar, 2007].

2.5.3.2 Anytime Learning

The Anytime Learning approach is designed for controller optimisation in dynamically changing environments [Grefenstette and Ramsey, 1992]. The approach is illustrated in Figure 2.10 and consists of an execution system and a learning system.

The execution system runs the current best controller on real-world hardware. A monitor observes the real-world environment in order to identify significant changes. Controllers are made up of a decision maker and a knowledge base. The decision maker controls the robot based on strategies specified by the active knowledge base. The knowledge base is a set of rules governing behavioural strategies. The real-world environment and controller performance is monitored continually. The learning system is notified of significant changes to the environment.

The learning system is similar to the execution system, except that a simulator model is used as an alternative to the real-world environment. If the environment changes significantly, the simulation model is updated to better reflect reality. A population of knowledge base test strategies is evolved using a Genetic Algorithm (GA). The best knowledge base discovered is transferred to the execution system, swapping out the previous knowledge base.

The effectiveness of the Anytime Learning approach is highly dependent of the design of the decision makers. Specialised knowledge of the robotic system may be required in order to create and update the simulation model [Zagal and Ruiz-del Solar, 2007]. Measuring certain real-world phenomena related to the simulation model can be impractical.

The approach is appropriate for model-based parameter tuning. The Anytime Learning approach has been validated using a two dimensional cat-and-mouse game. Prior work has also successfully applied the Anytime Learning approach for gait optimisation on a Hexapod robot with 12 degrees of freedom [Parker, 2000, 2002].

2.5.3.3 Estimation-Exploration Algorithm

The Estimation-Exploration Algorithm (EEA) is a hybrid co-evolutionary algorithm that evolves populations of simulator models and controllers [Bongard and Lipson, 2004b]. The population of simulator models is evolved to better approximate the morphology of some

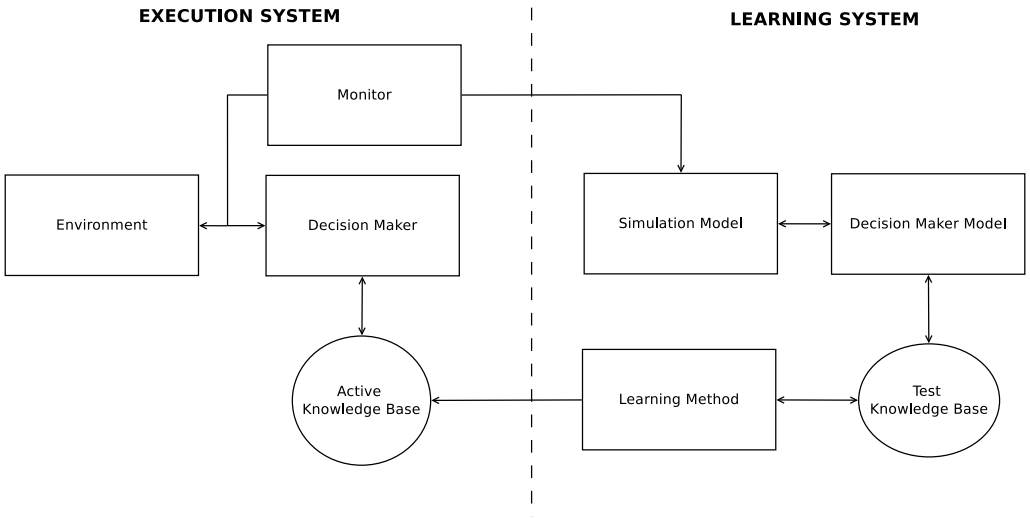


Figure 2.10: Anytime Learning System

target robot. Models consist of parameter settings and joint configurations, such as body dimensions, limb positions, weight distributions, gravity and others. A model's fitness is calculated by comparing simulated behaviours with those observed on the target robot. The population of controllers is evolved to either explore the simulation model search space or to maximise performance for a particular task [Bongard and Lipson, 2004b, 2005].

The approach consists of two distinct phases, namely, the estimation and exploration phases. During the estimation phase, controllers are evaluated on a real-world robot and behavioural data is collected. During the exploration phase, controllers are evolved to maximise the disagreement in behaviours observed when evaluated on the population of simulator models.

An illustration of the EEA is given in Figure 2.11. During the estimation phase, controllers are evaluated on the real-world robot (**A**). The behavioural data collected from these evaluation is compared to the behaviours produced by the models. The population of models is evolved in order to produce the behaviours observed in reality (**B**).

The exploration phase (**C**), evolves a population of controllers in order to maximise disagreement between the population of models. The optimisation process tries to discover controllers that demonstrate inconsistent behaviours when evaluated on the population of models.

The EEA alternates between the estimation and exploration phases until a termination

condition is triggered. Once an adequate model is found (**D**), controllers are evolved to solve a particular goal task, such as gait optimisation. The solution controller is validated on the real-world robot (**E**). Improved models, damage recovery can be achieved by alternating between the estimation and exploration phases or new solutions can be evolved with the existing model (**F**).

Prior work has validated the EEA using a four-legged, real-world robot with 8 degrees of freedom [Bongard *et al.*, 2006b; Liang and Xue, 2010]. Tilt and joint angle sensors were simulated. The approach has been used for gait optimisation problems, such as achieving forward locomotion. A Humanoid robot has been used to validate a variation of the EEA for evolving walking and kicking gaits [Iocchi, Libera, and Menegatti, 2007]. The Humanoid robot consisted of 22 degrees of freedom.

The EEA was also investigate using a Hexapod robots with 18 degrees of freedom for gait optimisation problems [Koos, Cully, and Mouret, 2013a]. However, the EEA failed to perform adequately well on a Hexapod robot platform. This failure might be due to the initial simulator not being powerful enough to capture the dynamics of the Hexapod robot or an insufficient number of real-world evaluations was performed.

The EEA can be used to allow a robot to quickly detect and recover from unanticipated damage [Bongard *et al.*, 2006a]. The approach can automatically fix or fine-tune inaccurate models. However, an initial simulator must be designed that is capable of simulating arbitrary robot morphologies. Developing a powerful simulator that is capable of simulating arbitrary robots morphologies or behaviours is a challenging problem.

2.5.3.4 Back to Reality Algorithm

The Back to Reality (BTR) algorithm co-evolves a population of controllers and simulator models [Zagal and Ruiz-del Solar, 2007]. Controllers are optimised for a particular task, such as developing fast gaits or kicking a ball. Controllers consist of parameter settings to parametrised mathematical equations. Models are optimised such that the difference between controller fitnesses observed in simulation and reality are minimised. Models consist of parameter settings to predefined physics equations. Examples of model parameter settings include mass distributions, friction properties, gravity, joint torques and others.

The BTR algorithm has been successfully validated on a physical four-legged Sony

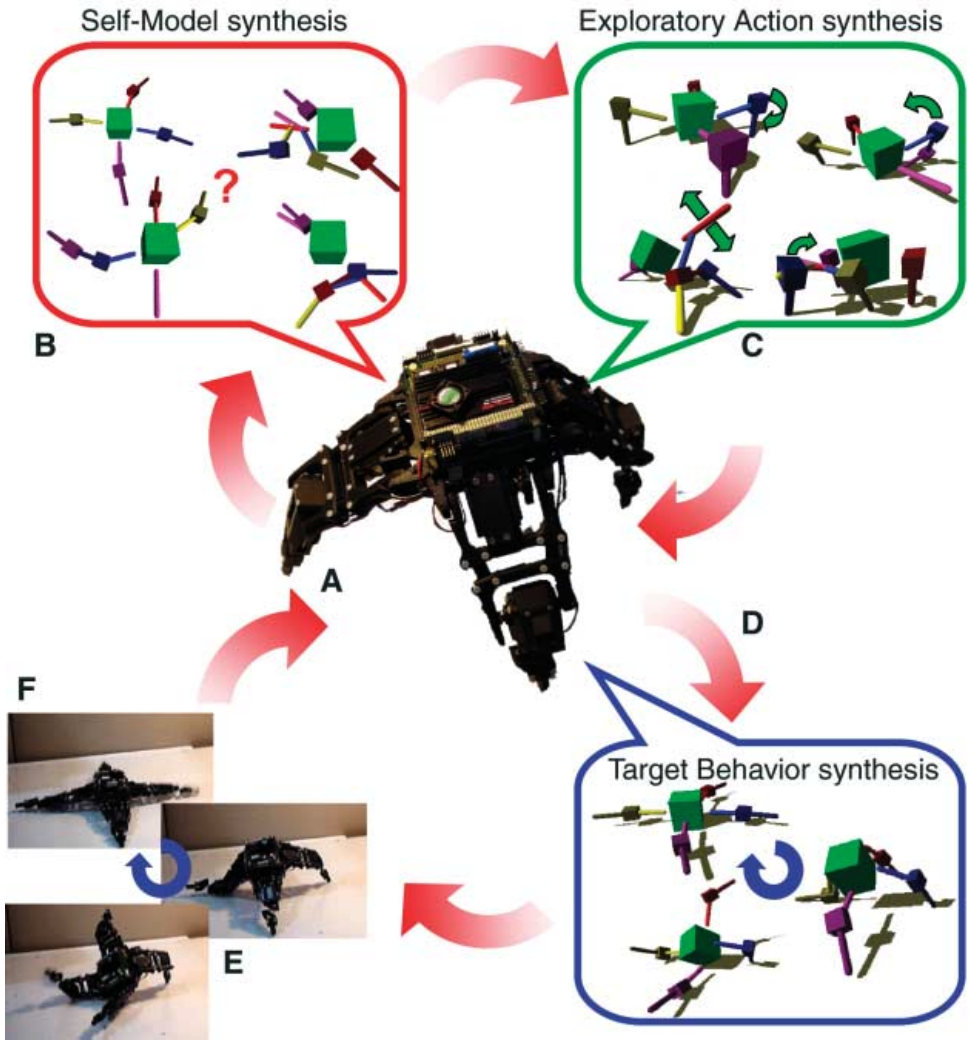


Figure 2.11: Estimation-Exploration Algorithm [Bongard *et al.*, 2006b]

AIBO robot with 12 degrees of freedom [Zagal and Ruiz-del Solar, 2007]. Controllers are optimised for either ball kicking or fast walking. Models consisted of 12 parameter settings. Depending on the problem, controllers consist of either 20 or 25 parameter settings.

The BTR algorithm has also been validated by generating walking behaviours for a humanoid soccer playing robot with 22 degrees of freedom [Zagal, Delpiano, and Ruiz-del Solar, 2009]. The controller consisted of 7 parameter settings. Simulator models consisted of 8 parameter settings.

The fitness function for assessing models is calculated as the average absolute difference between controller fitnesses in simulation and reality. The simulator optimisation process minimises $\triangle F$:

$$\triangle F = \frac{1}{m} \sum_{k=1}^{m} |f_{sk} - f_{rk}| \tag{2.6}$$

A GA is used to evolve the population of simulators. Each simulator's fitness is calculated using equation (2.6), where the best m controllers are selected and evaluated in simulation and reality. A controller's fitness in simulation is denoted as f_{sk} (s: simulator; k=1,...,m) and the corresponding real-world fitness is denoted as f_{rk} (r: reality; k=1,...,m).

The approach simply requires controller fitness assessments in simulation and reality. No explicit measurements of non-fitness related controller behavioural features are required. BTR can significantly reduce the number of real-world evaluations needed to evolve transferable controller solutions [Zagal and Ruiz-del Solar, 2007].

2.5.3.5 Transferability Approach

The Transferability approach does not optimise an existing simulator. Inaccurately modelled behaviours are identified and the ER process evolves solutions such that simulator weaknesses are avoided [Koos *et al.*, 2013b; Mouret *et al.*, 2012]. The ER process uses a multi-objective fitness function taking into account two factors. One factor estimates a controller's ability to solve the given task. The second factor, called the transferability function, estimates how well simulated behaviours will transfer into reality.

The transferability function approximates the size of the *reality-gap* by modelling dif-

ferences between chosen behavioural metrics in simulation and reality. These behavioural metrics can include differences in 3D-trajectories, joint positions, leg contact times with the ground and others. The transferability function can be implemented using standard machine learning techniques, such as ANNs or Support Vector Machines. The transferability function could map a controller's genotype to a particular transferability score, however, this type of mapping can become difficult to learn for high dimensional genotypes. Alternatively, mapping a relatively low dimensional simulated behaviour (such as leg contact times with the ground) to a transferability measure can be more effective. Expert domain knowledge might be required to identify appropriate transferability mappings and measuring particular behaviours in reality can be difficult.

The Transferability approach has been validated by optimising walking gaits on a quadruped robot with eight degrees of freedom [Mouret *et al.*, 2012]. Controllers consisted of sinusoidal mathematical equations with parameter settings requiring optimisation. Experimental work demonstrated that the Transferability approach can significantly reduce the *reality-gap* problem and improve performance outcomes. The Transferability approach has also been adapted (T-Resilience Algorithm) for use in optimising walking gaits on an undamaged and damaged Hexapod robot with 18 degrees of freedom [Koos *et al.*, 2013a].

2.5.3.6 Intelligent Trial-and-Error Learning

The Intelligent Trial-and-Error Learning approach (Figure 2.12) computes an initial behavioural performance map before the ER process can begin [Cully *et al.*, 2015]. The behavioural performance map is also updated during the ER process based on real-world feedback. A controller's evaluated behaviour can be measured in terms of behavioural metrics, such as leg contact times with the ground. The approach has been validated on a six-legged robot. The measured contact times of each leg with the ground (6-dimensional behavioural space) can be mapped to a single performance measure, such as the walking gait speed. The high dimensional search space of a controller is reduced to a low dimensional behaviour space (**A & B**). An optimisation algorithm can discover a large number of controller solutions using a high fidelity simulator and construct a behavioural performance map based on simulated performance. This initial pre-computed behavioural performance map requires an existing simulator to be built.

A behavioural confidence mapping is also required. The confidence level is a measure of the reliability of a performance prediction. Initially, low confidence levels are assigned over the entire behavioural confidence mapping due to the lack of real-world fitness assessments. Confidence levels improve when controllers are evaluated in reality.

The performance and confidence level mappings are updated as controllers are evaluated in reality (**C & D**). Nearby mappings are adjusted with performance and confidence levels proportional to their proximity to the behaviours observed in reality. After a map update, a new controller is selected and evaluated in reality, producing further updates. This process continues until a sufficiently good solution is found (**E**).

Validation experiments investigated the Intelligent Trial-and-Error approach on a Hexapod robot with 18 degrees of freedom [Cully *et al.*, 2015]. Effective walking gait controllers were discovered in less than a minute for the undamaged configuration and less than 3 minutes for damaged configurations [Cully *et al.*, 2015]. Additional validation experiments were conducted on a physical robotic arm with 9 degrees of freedom. The robot arm results demonstrated success for problems involving placing balls into bins.

The approach can effectively discover viable controller solutions quickly. If the robot is damaged, a lengthy self-diagnosis period might be avoided and the simulator does not need to be corrected. However, a relatively high fidelity physics-based simulator is developed before the behavioural performance map can be compiled. The approach requires millions of controller evaluations in simulation in order to produce the behavioural performance map. Expert knowledge could be required in order to identify useful behavioural measures used to form the behavioural performance map.

2.5.3.7 Model-fitting based on Empirical Data

Surrogates or meta-models can be used to help approximate the fitness of individuals in Evolutionary Algorithms (EAs). Surrogate models can be constructed from empirically collected data and machine learning techniques. Surrogate modelling is often used when a high fidelity simulator becomes too computationally expensive to perform the high number of evaluations required in an EA. A surrogate model is used to help approximate the fitnesses of many candidate solutions while the computationally expensive simulator is used to selectively evaluate only the most promising solutions. A surrogate model can

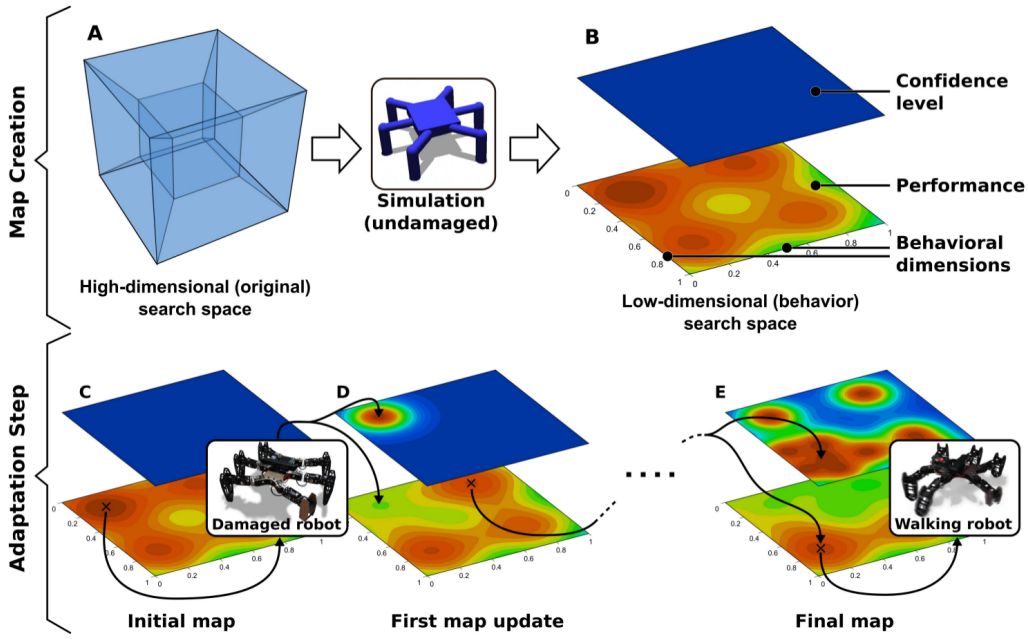


Figure 2.12: Intelligent Trial-and-Error Learning [Cully *et al.*, 2015]

also serve as a complete replacement to a high fidelity simulator. Surrogate models can be built before or during the optimisation process [Mainini and Willcox, 2015; Zavoianu, Lughofer, Bramerdorfer, Amrhein, and Klement, 2013].

Robotics systems can be modelled using equation (2.7). The current state of the environment (s_t), the action or command (a_t) to be applied and the transition state (s_{t+1}) after an action is applied . The state and action search spaces can be either discrete or continuous. Physics models create mathematical functions, f, using theoretical physical knowledge of a robotic system.

$$s_{t+1} = f(s_t, a_t) \tag{2.7}$$

A more autonomous, machine learning approach can be used to develop f. A real-world robot generates behavioural data that can be collected. The collected behaviour data could be used to train a model using machine learning techniques. Models can be constructed using machine learning and empirically collected data with little prior knowledge of the dynamics of the given robot system [De Nardi and Holland, 2008; Kamio and Iba, 2004; Pretorius *et al.*, 2019; Woodford *et al.*, 2017].

Prediction models can be developed through reinforcement learning approaches [Kamio and Iba, 2005], evolutionary algorithms [De Nardi and Holland, 2008], SNNs [Pretorius *et al.*, 2017; Woodford *et al.*, 2017] and many others. Modelling empirical data directly can be easier than modelling a system using physics-based principles [Pretorius *et al.*, 2017, 2019].

De Nardi and Holland [2008] used Genetic Programming to model a miniature rotorcraft from empirically collected data. The robot consists of four propellers. The dynamics of the rotorcraft were successfully modelled without specialised domain knowledge.

Kamio and Iba [2004] successfully validated that Genetic Programming and cluster approximation techniques can adequately model certain behaviours of a humanoid robot with 20 degrees of freedom. However, the controller design is simplified and utilises a significant degree of prior knowledge. Commands consisted of discrete actions such as sidesteps, forward/backwards, turning and various combinations of actions. Controllers were trained to move a box to a specific goal region.

SNNs can simulate different robot morphologies, such as wheeled, Snake and Hexapod robots [Pretorius *et al.*, 2014, 2019; Woodford *et al.*, 2015, 2016]. Bidirectionally developed SNNs have also been investigated [Woodford *et al.*, 2016, 2017]. These approaches are discussed in detail in the following sections.

2.5.4 Simulator Neural Networks

Some of the proof of concept work regarding SNNs was performed by the current author [Woodford *et al.*, 2015, 2016, 2017]. Other researchers have also shown that SNNs can serve as an alternative to physics-based simulators [De Nardi, 2010; Nakamura and Hashimoto, 2007; Pretorius *et al.*, 2009; Wang, 2005]. Controllers have been evolved with the help of SNNs for problems such to gait optimisation, trajectory planning, obstacle avoidance, balancing and light following problems [Pretorius, du Plessis, and Cilliers, 2013; Pretorius *et al.*, 2017, 2019; Woodford *et al.*, 2015]. SNNs can handle noisy data, possess good generalisation abilities and can accurately model complex relationships. SNNs can be simpler to construct compared to physics-based approaches [Pretorius *et al.*, 2017, 2019]. Prior research has found that SNNs can be more accurate, computationally efficient and transferable compared to certain physics-based approaches [Pretorius *et al.*, 2014, 2019].

An explanation of the behavioural components simulated using SNNs is given in Section 2.5.4.1. A controller design can be simplified with built-in prior knowledge of robotic locomotion or the design can purposely not rely on any prior knowledge (Section 2.5.4.2). The platforms used to evaluate the controllers in this research are discussed in Section 2.5.4.3. The SNS and BNS approaches are presented in Sections 2.5.4.4 and 2.5.4.5, respectively. Two robot platforms are explored in this thesis, namely, a Hexapod robot (Section 2.5.4.6) and Snake robot (Section 2.5.4.7).

2.5.4.1 Behavioural Components

A robot's behaviours can be broken down into smaller components (Figure 2.13). Robot behaviours can be measured according to a single or group of commands, depending on the controller design.

A command is a set of actions that can be applied to a robot's actuators, such as changes to joint angles or specified motor speeds. Behaviours can be broken down into physical components. The relative change in the position of a robot after executing a command is measured as Δx, Δy and Δa. Depending on the controller design, physical components could be measured for a group of commands instead of a single command. The x-axis represents lateral (sideways) movement of the robot and the y-axis represents forward/backward movement. Changes in position are relative the centre point on the robot's head. The relative change in the robot's position with respect to the perpendicular of the initial heading is denoted as Δx. The relative change in the robot's position parallel to the initial heading is denoted as Δy. The relative change in the robot's heading is expressed as Δa.

SNNs can be designed to take as input the command sent directly to a robot's actuators. Alternatively, SNNs can take as input the parameter settings used to generate a sequential list of commands. SNNs are designed to take as input either a single command or a set of parameter settings, then output the behavioural components, Δx, Δy and Δa. A single SNN can be configured to predict either one or all physical components as output. Prior work has found that multiple SNNs, each predicting a single physical component tends to have better accuracy [Pretorius *et al.*, 2013]. For any given controller, a sequential list of behavioural components is predicted (Δx, Δy, Δa). These predictions

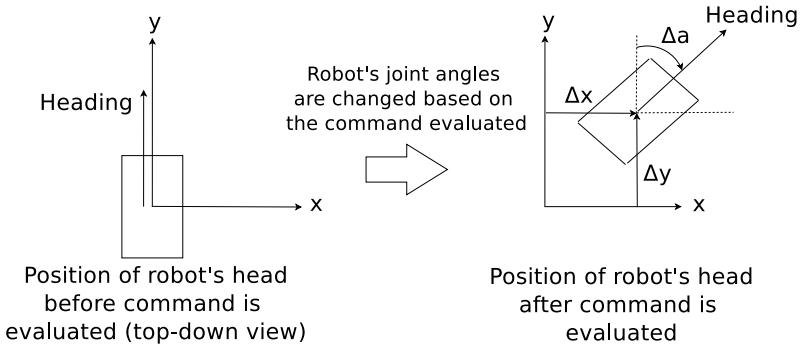


Figure 2.13: Behavioural components of a robot

accumulate to form an overall approximation of a controller's total behaviour, such as the path followed by the robot or the values of a sensor over time. A behavioural dataset is defined in this work as a collection of command/parameter inputs and corresponding behavioural component outputs used to form the training, validation and test datasets. The behavioural dataset is constructed by sending many commands to a robot and observing the resulting behavioural components generated.

SNNs can be trained either before [Pretorius *et al.*, 2017; Woodford and du Plessis, 2018, 2019; Woodford *et al.*, 2015] or during the ER process [Woodford *et al.*, 2016, 2017]. The Static Neuro-Simulation (SNS) approach trains SNNs before the ER process begins and SNNs remain static during the ER process (Section 2.5.4.4). Section 2.5.4.5 discusses the Bootstrapped Neuro-Simulation (BNS) approach where SNNs are trained during the ER process.

2.5.4.2 Level of Prior Knowledge

Table 2.1 lists some robot morphologies that have been studied using the SNS and BNS approaches. Wheeled robot morphologies have been investigated for tasks such as trajectory planning, light approaching behaviours and obstacle avoidance [Pretorius, 2010; Woodford *et al.*, 2016]. A balancing robot was used to study the inverted pendulum stabilisation problem [Pretorius *et al.*, 2017]. The Hexapod robot platform has been used to study gait optimisation problems [Pretorius *et al.*, 2019]. A Snake robot has been used to investigate trajectory planning scenarios [Woodford *et al.*, 2015, 2017].

A controller design can have built-in prior knowledge of the robotic system or the design

can avoid using existing human gained knowledge of the problem. Scripted controllers consist of a sequential list of low level commands sent to the robot directly, such as sequences of joint angle changes or wheel speeds. Scripted or ANN-based controllers do not use prior knowledge to help guide and simplify the solution search space. Parametrised controllers use mathematical functions or central patten generators to help generate the low level commands sent to a robot based on specialised knowledge of the robotic system. For a parametrised controller design, the parameter settings applied to a mathematical function constructed by a human are optimised. The checkmarks in Table 2.1 indicate that the particular robot and controller design have been investigated in prior work for the SNS and BNS approaches. With the exception of the Snake robot, all prior work investigating the SNS or BNS approaches do not rely on controller designs with built-in prior knowledge of the robotic problem. Due to the complexity of using the Snake robot, the controller was simplified through prior knowledge.

The SNS approach has been investigated for Wheeled, Balancing and Hexapod robots using controller designs without prior knowledge. The Snake robot uses a parametrised controller design for the SNS and BNS approaches. The BNS approach has not been investigated for a Balancing or Hexapod robot morphology. The viability of using the SNS approach on different classes of complex robot morphologies has been demonstrated through experimental work on both the Hexapod and Snake robots. However, for complex robot morphologies, the BNS approach has only be demonstrated for a Snake robot using a controller design reliant on a significant level of prior human knowledge. The generalisability of the BNS approach to work on different classes of complex robots using controller designs without prior knowledge has not been established.

2.5.4.3 Evaluation Platforms

Controllers are evaluated either in simulation or reality. Figure 2.14 illustrates the different controller evaluation platforms (Static SNNs, Dynamic SNNs, Real-World Robot) that will be referenced throughout this thesis. Two different simulation approaches are explored, namely Static and Dynamic SNNs. Static and Dynamic SNNs simulate the real-world behaviours of controllers. The real-world robot determines the actual performance of any given controller. Static SNNs are trained from randomly generated behavioural

	SNS		BNS	
Controller	***Prior Knowledge***	***No Prior Knowledge***	***Prior Knowledge***	***No Prior Knowledge***
Wheeled robot		✓		✓
Balancing robot		✓		
Hexapod robot		✓		
Snake robot	✓		✓	

Table 2.1: Robot morphologies investigated using the SNS and BNS approaches

data collected from real-world robot evaluations. Static SNNs are used as part of the SNS approach. Dynamic SNNs are developed during the ER process and are trained using behavioural data generated from evaluations of select controllers from the evolving controller population. Dynamic SNNs are used as part of the BNS approach.

2.5.4.4 The SNS Approach

The SNS approach [Pretorius *et al.*, 2009] is illustrated in Figure 2.15. A behavioural data collection phase is performed before SNNs can be trained and utilised in the ER process. During the data collection phase, commands are randomly generated and executed on a real-world robot. The behavioural data collection phase continues until enough data is available to adequately represent the behavioural search space. The number of patterns required to adequately simulate a given robotic system is determined experimentally.

Before SNN training can begin, the appropriate SNN architectures need to be chosen. The number of hidden layers and nodes required to accurately simulate behavioural components is initially unknown for any given robotic system. Appropriate SNN architectures are normally determined through a benchmarking process that compares the accuracy of different SNN architectures. Evolving SNN architectures and weight parameters using the Neuroevolution of Augmenting Topologies (NEAT) algorithm has been investigated but the accuracy of produced SNNs tended to be relatively poor [Pretorius *et al.*, 2017].

Once good SNN architectures have been found, a training phase is completed before controller evolution can begin. Behavioural data variables are standardised before training so that each variable's distribution has a mean of zero and a standard deviation of one.

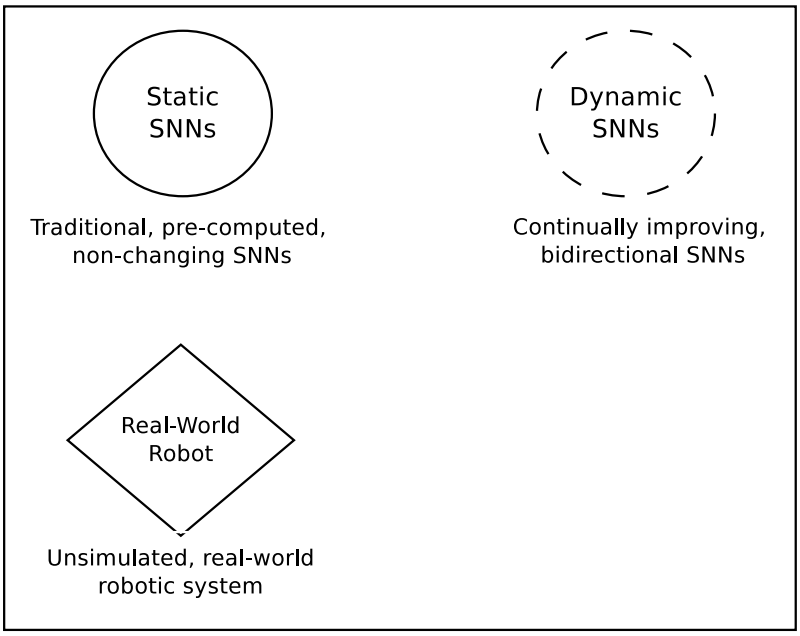


Figure 2.14: Controller evaluation platforms

Training SNNs on standardised data tends to improve accuracy. The behavioural data is split into three groups, namely, training, validation and test datasets. SNNs are trained using the training dataset while the validation dataset is used to prevent over-fitting. The test dataset is not available during training and is used to estimate the real-world prediction accuracy of the trained SNNs. The training process produces Static SNNs. These Static SNNs do not change in any way after the training phase is complete.

For all simulated controller evaluations performed during the ER process, simulator noise is typically injected into SNN predictions. Simulator noise is sourced from a Gaussian distribution with a mean of zero. The standard deviation is equal to the standard deviation of the prediction errors between the trained simulator and the test dataset.

The Static SNNs simulate robot behaviours during controller evolution and serve as an alternative to real-world controller evaluations. Controllers are evolved for many generations until a stopping condition is met. The best controller in the last generation is selected as the final controller solution.

The pendulum swing-up problem has been simulated using SNNs [Nakamura *et al.*, 2007]. The dynamics of miniature rotorcraft and vehicles have been simulated using SNNs [De Nardi, 2010]. Differentially-steered mobile robot platforms have been used to

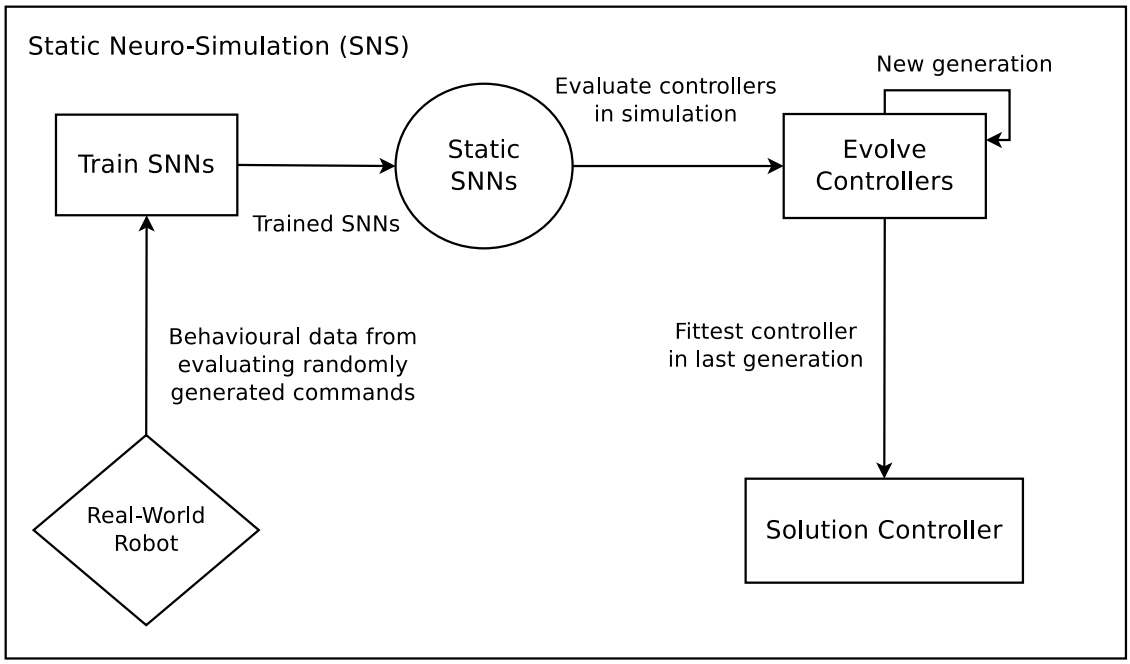


Figure 2.15: Static Neuro-Simulation (SNS) approach

study the SNS approach on tasks such as obstacle avoidance, trajectory planning, light approaching behaviour and inverted pendulum stabilisation [Pretorius *et al.*, 2013, 2014, 2017]. However, these robot morphologies are relatively simple and have a low number of degrees of freedom.

Prior work using Snake and Hexapod robot platforms have validated that the SNS approach is viable on complex robot morphologies [Pretorius *et al.*, 2019; Woodford *et al.*, 2015]. A Snake robot has been simulated using SNNs for trajectory planning problems [Woodford *et al.*, 2016]. However, the prior work use a parametrised Snake controller design with built-in knowledge of biological snake locomotion. SNNs were also specialised to only be compatible with the chosen parametrised Snake controller design. Prior work on the Snake robot is discussed in Section 2.5.4.6.

A Hexapod robot platform has been simulated using SNNs and controllers evolved for a gait optimisation problem [Pretorius *et al.*, 2019]. Controllers were evolved to maximise the total Euclidean distance travelled. Unlike prior work on the Snake robot, the Hexapod robot did not use a parametrised controller design with prior knowledge. The Hexapod controller design was scripted and did not include any built-in prior knowledge of Hexapod

locomotion modes. The Hexapod simulator was designed to be compatible with any controller design, scripted or parametrised. The prior research specific to the Hexapod robot is covered in greater detail in Section 2.5.4.7.

2.5.4.5 The BNS Approach

The BNS approach was proposed and demonstrated to be viable by the current author in prior work [Woodford *et al.*, 2015, 2016, 2017]. The BNS approach performs data collection, simulator training and controller evolution concurrently. The ER process can begin without a fully trained simulator. A population of controllers is continually evolved to solve a particular task. Behavioural data is sourced through real-world evaluations of controllers selected from the evolving controller population. The behavioural data used to train SNNs are candidate controllers seen during controller evolution.

The BNS approach is illustrated in Figure 2.16. Controllers are continually selected from the controller population and evaluated on the real-world robot. Selecting a controller from the controller population and evaluating it on a target robot is referred to as a *sampling evaluation*. At the time of selection, a fixed number of controllers is randomly chosen from the latest controller population and the fittest controller is evaluated on the real-world robot. The resulting behavioural data is collected and used to improve a set of Dynamic SNNs.

All newly sourced behavioural data is added to a training dataset. Splitting patterns into training and validation datasets have not yet been investigated. The Dynamic SNNs are continually trained while periodically integrating newly acquired training data. Dynamic SNNs are used to help evaluate controllers during the controller evolution process. The continual optimisation of controllers and SNNs is performed until some stopping condition is met. The fittest controller in the final controller population is produced as the final solution controller.

For the SNS approach, the parameter settings used to standardise the behavioural dataset or generate simulator noise is calculated based on the already collected behavioural dataset. However, an initial large behavioural dataset is not available during the early stages of the BNS process. Prior work on the BNS approach used the standardisation and noise generation parameter settings calculated as part of the SNS approach. For the

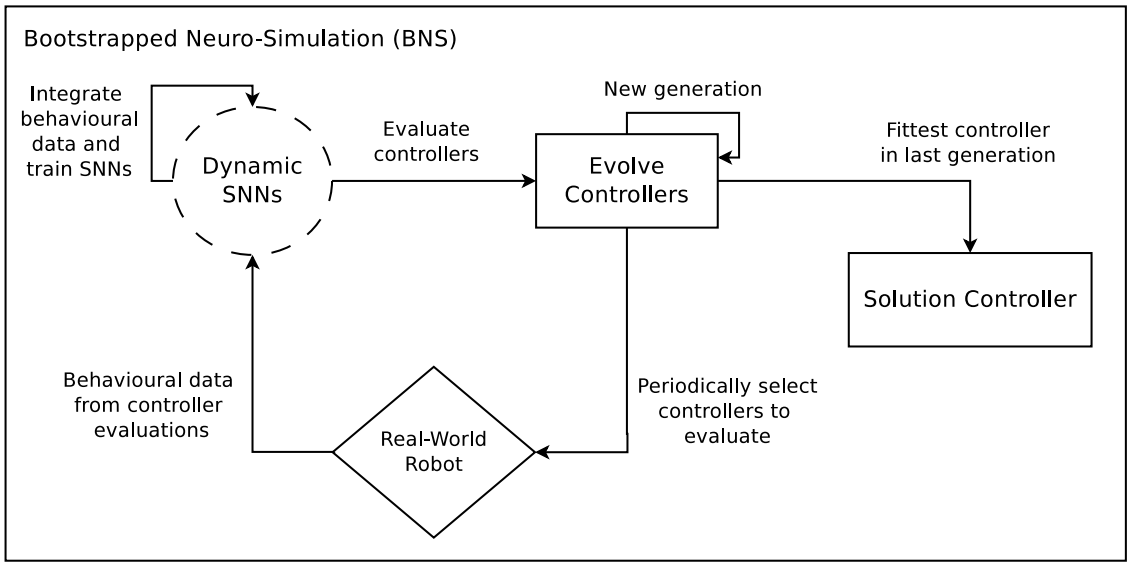


Figure 2.16: Bootstrapped Neuro-Simulation (BNS)

BNS approach to be practically applied to robotics systems without an initial behavioural dataset collection phase, the noise and standardisation parameter settings would need to be calculated during the BNS approach. A method for dynamically standardising the behavioural dataset during the BNS approach is yet to be investigated. A method for dynamically calculating the simulator noise parameter settings during the BNS approach has not been investigated.

A robot's morphology or environment could change under certain scenarios. Any significant changes might render an existing simulator inaccurate. The data collection process would need to be restarted and a new simulator trained. Training a new simulator using the SNS approach can be time-consuming due to the lengthy data collection process. The BNS approach is designed to reduce the amount of behavioural data collected compared to the SNS approach [Woodford, 2016].

A simulator developed using the BNS approach specialises in simulating behaviours seen during controller evolution. As a consequence, the simulator might not generalise well to unseen behaviours. No research has investigated the re-usability of simulators developed using the BNS approach.

The BNS approach has been validated on a simple differentially-steered Khepera robot for trajectory planning problems [Woodford *et al.*, 2016]. For complex robot morphologies,

the BNS approach has only been validated on a Snake robot platform with a parametrised controller design [Woodford *et al.*, 2017]. Snake robot controllers were evolved to solve trajectory planning problems. The Snake robot controller and simulator designs were simplified in order to reduce the complexity of the controller and simulator search spaces. In the prior research, the Snake robot controller design used a mathematical function that simplified the generation of snake-like locomotion modes. However, the Snake robot simulator was only capable of simulating behaviours generated by the chosen controller design. Due to this specialisation, the Snake robot simulator was not compatible with other alternative Snake robot controller designs. Prior work specific to the Snake robot is covered in greater detail in Section 2.5.4.6.

In prior research, the relationship between BNS simulated and real-world fitnesses were studied [Woodford *et al.*, 2016]. Estimating the real-world fitness of solutions over time for a single run of the BNS approach is practically infeasible using a real-world robot due to the large number of controller evaluations involved. It was necessary to use a Static Simulator as an alternative to a real-world robot (substitute real-world) in order to estimate the relationship between BNS simulated and real-world fitnesses over the lifetime of the BNS approach. Figure 2.17 illustrates the substitute real-world (Static Simulator) and BNS simulated fitnesses of solution controllers over the lifetime of a single run of the BNS approach. Higher values indicate better fitness. The dotted line represents the substitute real-world fitness of solutions over time. The solid line represents the simulated (Dynamic SNNs) fitnesses of solution controllers over time. Early fitness assessments are inaccurate and volatile but gradually stabilise as accuracy improves over time. The substitute real-world fitnesses are almost always overestimated by the BNS simulator. Towards the end of the BNS approach, the substitute real-world fitnesses converge towards the BNS simulated fitnesses and the robot is able to solve the given robotic problem. In summary, solution controllers solve the problem in the inaccurate BNS simulator and as the BNS simulator becomes more accurate, the real-world solutions slowly converge towards the solved behaviour.

No studies have investigated the BNS approach on complex robots with a scripted controller design. A scripted controller design consists of low level, arbitrary commands not constrained by expert knowledge, such as a mathematical function modelling biological

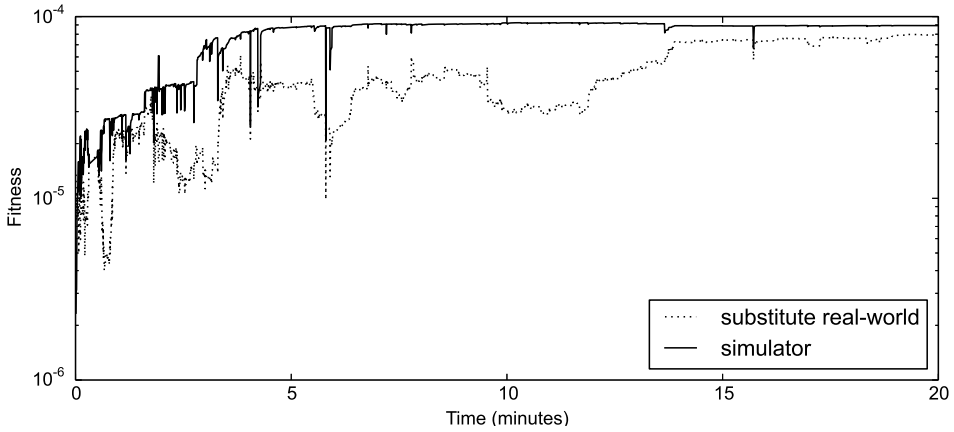


Figure 2.17: Simulated and real-world fitness over time for BNS approach [Woodford *et al.*, 2016]

snake locomotion. A scripted controller design requires the use of a generalised simulator design capable of simulating low-level robot commands. A generalised simulator is compatible with any controller design.

In prior work investigating the BNS approach, no validation dataset is used to avoid over-fitting during SNN training. The SNNs used in previous studies had a low number of input features and the SNN architectures were relatively small. The continual addition of new training data during the BNS approach, low number of input features, combined with simple SNN architectures helped the training process avoid over-fitting. Over-fitting is of greater concern when studying more complex robot morphologies.

2.5.4.6 Simulating Snake Robots using SNNs

A Snake robot can be considered a complex morphology due to the high number of degrees of freedom and dynamics involved. Prior work has investigated the SNS [Woodford *et al.*, 2015] and BNS [Woodford *et al.*, 2017] approaches on a Snake robot platform. The prior work used a simulator design only compatible with the chosen parametrised controller design. The controller design used parametrised equations [Melo *et al.*, 2012] (equations (2.8) and (2.9)) that generates joint angles mimicking biological snake locomotion. The equation can generate angles at discrete time steps t. The angle $\phi(n,t)$ of the n^{th} joint, at time step t is calculated using equation (2.8). The parameter setting values $A_{lateral}$,

$A_{vertical}$, ω and α affect the locomotion modes generated.

$$\phi(n,t) = \begin{cases} A_{lateral} \cdot sin(\theta + \alpha), & n \text{ is even} \\ A_{vertical} \cdot sin(\theta), & \text{otherwise} \end{cases} \tag{2.8}$$

$$\theta = \omega n + 2\pi t/12 \tag{2.9}$$

The simulator predicts changes in position for two tracked positions on the Snake robot. These two tracked positions are 65cm apart when the Snake robot is completely straight. The heading of the robot is calculated as the direction from one tracked position to the other.

The controller design consisted of a sequential list of parameter settings to the parametrised equations (Figure 2.18). Each cycle consisted of unique parameter setting values. The SNNs were designed to predict the behavioural outcomes of the two tracked positions, per cycle. SNNs took as input the parameter settings to equations (2.8) and (2.9) and predicted the changes in position for the first ($\Delta x_1(t)$ and $\Delta y_1(t)$) and second ($\Delta x_2(t)$ and $\Delta y_2(t)$) tracked positions, where t is the time index. The heading of the robot was simulated as the direction from the first tracked position to the second. The trajectory of the robot was calculated as the midpoint of the tracked markers over time. When all cycles have been simulated, the trajectory followed by the robot can be calculated and controller fitness assessed.

SNNs that take as input parametrised features are not generalisable to other controller designs. The simulator ignored changes in position related to transitions between cycles which introduced inaccuracies in overall simulated behaviours. Simulating two separate tracked positions resulted in the simulated distances between the tracked markers becoming unrealistically too far or close to each other over time. No prior work has simulated sensors on a Snake robot for the SNS or BNS approaches.

2.5.4.7 Simulating Hexapod Robots using SNNs

Pretorius *et al.* [2019] demonstrated that the SNS approach is viable for a Hexapod robot. The SNS approach has been compared to standard physics-based simulation techniques for a Hexapod robot platform [Pretorius *et al.*, 2019]. A scripted controller design was

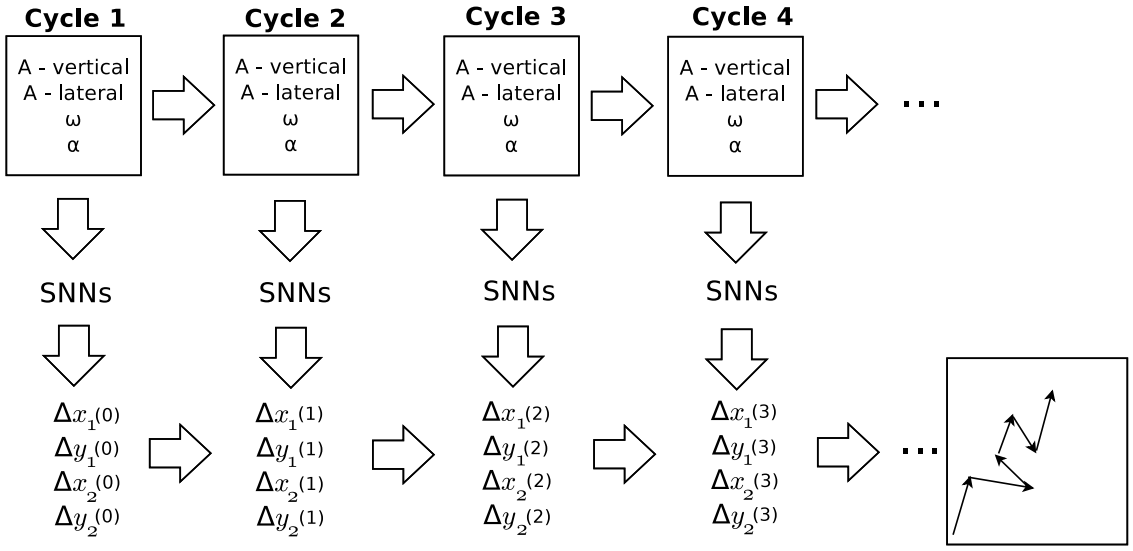


Figure 2.18: Snake controller and simulator design

used with no built-in prior knowledge of known Hexapod locomotion modes. The Hexapod SNNs were designed to be low level and compatible with arbitrary controller designs. Validation experiments found that the Hexapod SNNs are relatively simple to construct, computationally efficient and are an accurate alternative to certain physic-based approaches [Pretorius *et al.*, 2019].

A controller evaluation in simulation is illustrated in Figure 2.19. A scripted controller consists of a sequential list of commands. Each command contains angle changes that are applied to each joint on the Hexapod robot. Joints are indexed numerically. The starting and transition joint angles for the i^{th} command and j^{th} joint is represented by α_{ij} and β_{ij}, respectively (for $i = 1, 2, ..., n$ and $j = 1, 2, ..., m$). SNNs take as input the starting and transition joint angles of each command and output the predicted behavioural components $\Delta x(t)$, $\Delta y(t)$ and $\Delta a(t)$ where t is the time step. The accumulated predictions are used to calculate the overall trajectory and heading of the Hexapod robot over time.

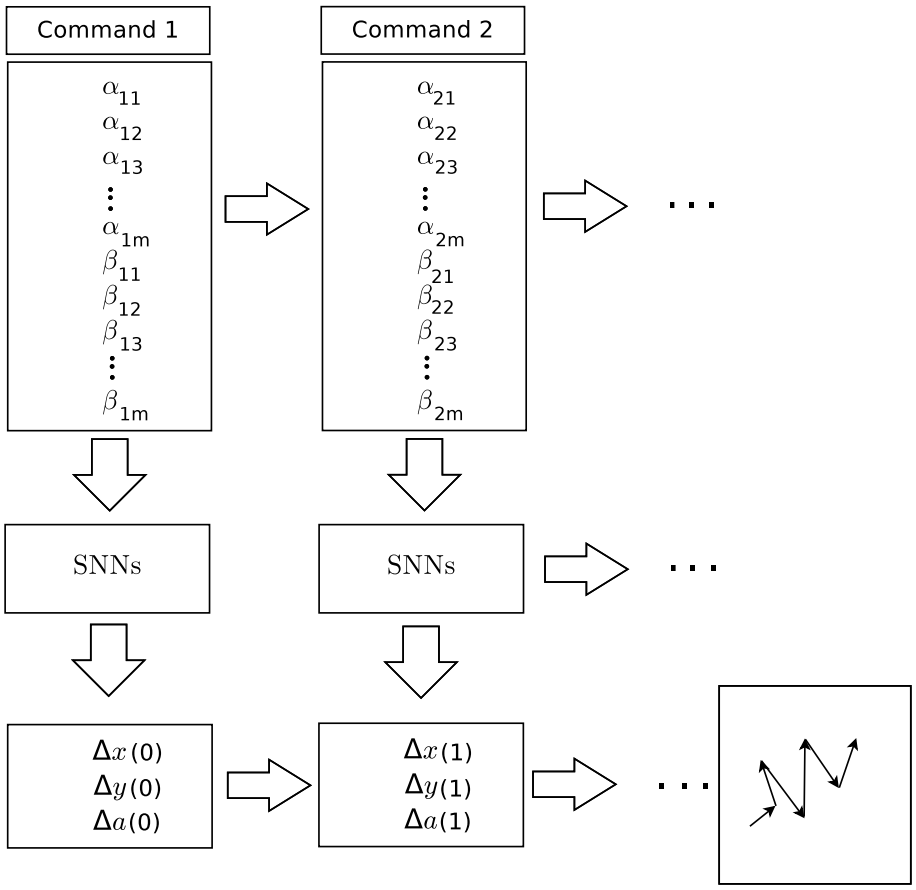


Figure 2.19: Hexapod controller and simulator design

2.6 High Level Comparison of ER Approaches

Natural biological evolution happens automatically and without the help of human interventions. An ideal ER approach would be able to evolve robotic controllers for complex robots without the help or expertise of humans while also not taking an infeasibly long time to produce effective solutions. Ideal ER approaches would require few human interventions or specialised knowledge, can produce effective solutions in a reasonable amount of time and still be effective when increasing the complexity of the robot morphology.

This section speculates as to which ER approaches have the most potential in terms of the outlined ideal ER properties. Currently, no existing ER approach perfectly achieves all ideal properties but some ER approaches have demonstrated significant potential in performing well in most aspects.

ER Approach	Scalability	Generalisability	No pre-developed simulator	Little specialised knowledge	Narrowing reality-gap	Minimises data collection
Reality	✓	✓	✓	✓	N/A	
Simulation	✓	✓				
Minimal Simulation	✓					
Anytime Learning	✓				✓	✓
Estimation-Exploration	✓				✓	✓
Transferability	✓				✓	✓
Intelligent Trial-and-Error	✓				✓	✓
Back to Reality	✓	✓			✓	✓
Genetic Programming			✓	✓	✓	✓
Static Neuro-Simulation (SNS)	✓	✓		✓		
Bootstrapped-Neuro Simulation (BNS)	✓		✓	✓	✓	✓

Table 2.2: Comparison of ER approaches

In Table 2.2, the rows contain different ER approaches and the columns represent important goals for finding an ideal ER approach. Approaches that have been demonstrated to exhibit particular properties are marked with a ✓ in the table.

In order to determine if an ER approach can handle more complex robots, two aspects are studied. Is the ER approach scalable for use on more complex robots? Secondly, can the ER approach generalise for use on more than one class of complex robots? Many ER approaches are only validated on a single complex robot morphology. The **generalisability** of ER approaches to work across different classes of complex robot morphologies is typically not studied. The complexity of a robot can be categorised according to the number of degrees of freedom involved in locomotion (Table 2.3). At least in this study, only robots with at least 12 degrees of freedom of movement are considered complex robot

morphologies. An ER approach is defined as **scalable** if it has been validated on at least one complex robot morphology without drastically simplifying the controller design.

The time taken for an ER approach to develop effective solutions is important. **Not requiring an initial simulator** to be developed before the ER process begins eliminates a lengthy simulator development process. ER approaches that reduce the number of controller evaluations on real-world hardware, **minimising data collection**, can greatly reduce the time taken to develop effective controller solutions.

Different ER approaches require varying degrees of human interventions. **Requiring little specialised knowledge** to develop a robotic simulator reduces the need for human interventions and the simulator development process can be automated. Physics-based simulators require significant human understanding to use effectively. SNNs are based on Machine Learning principles and are relatively easy to automate.

Simulators will inevitably contain weaknesses or encounter unseen behaviours that end up being exploited by the *reality-gap* problem. These problems can be resolved through human interventions. However, some ER approaches have a mechanism for integrating real-world feedback for simulator improvement or augmentation. Bi-directional ER approaches attempt to automatically **narrow the reality-gap** during the ER process without human interventions.

Evolving controllers directly on complex robots can be time-consuming but still feasible [Hornby *et al.*, 2005; Lipson *et al.*, 2006]. Alternatively, evolving controllers in simulation tends to produce solutions with poor transferability unless the simulator is sufficiently accurate. Prior work has demonstrated that evolving controllers in a physics-based simulator is possible and solutions can transfer well into reality for complex robots [Belter and Skrzypczyński, 2010; Hong and Lee, 2017; Pretorius *et al.*, 2019]. Many ER approaches have been validated to be scalable, namely, Minimal Simulation [Jakobi, 1998b], Anytime Learning [Parker, 2002], Estimation-Exploration [Iocchi *et al.*, 2007], Transferability [Koos *et al.*, 2013a], Intelligent Trial-and-Error Learning [Cully *et al.*, 2015], Back to Reality [Zagal and Ruiz-del Solar, 2007; Zagal *et al.*, 2009], SNS approach [Pretorius *et al.*, 2019; Woodford, 2016] and BNS approach [Woodford *et al.*, 2017].

Most ER approaches have not been validated on significantly different classes of complex robot morphologies. For example, an ER approach could be viable on a Hexapod

ER Approach	Morphology class	Degrees of freedom	Complex	Generalisable
Reality	Double Stewart Platform	12	Yes	Yes
	Quadruped	19	Yes	
Simulation	Biped	20	Yes	Yes
	Hexapod	18	Yes	
Minimal Simulation	Octopod	16	Yes	No
Anytime Learning	Hexapod	12	Yes	No
Estimation-Exploration	Biped	22	Yes	No
	Starfish	8	No	
Transferability	Hexapod	18	Yes	No
	Quadruped	8	No	
Intelligent Trial-and-Error	Hexapod	18	Yes	No
	Arm	9	No	
Back to Reality	Biped	22	Yes	Yes
	Quadruped	12	Yes	
Genetic Programming	Biped (simplified)	20	No	No
	Quadcopter	4	No	
Static Neuro-Simulation (SNS)	Hexapod	18	Yes	Yes
	Snake	12	Yes	
Bootstrapped Neuro-Simulation (BNS)	Snake	12	Yes	No

Table 2.3: Validated robot morphologies per ER approaches

robot for limbed behaviours but non-viable on a Snake robot. An ER approach is demonstrated to be generalisable if more than one class of complex robot morphologies have been used validate the approach in reality. Classes that differ significantly from each other are Hexapod, Bipedal, Crawling and Flying robots. In Table 2.2, some ER approaches could potentially be generalisable but no validation studies could be found.

Robot morphologies validated for each ER approach are given in Table 2.3. Evolving controllers completely in reality has been validated on a nine-legged double Stewart platform (robot body connected with many linear actuators) [Lipson *et al.*, 2006] and Quadruped robots [Hornby *et al.*, 2005]. Evolving controllers completely in simulation has also been validated on Hexapod [Pretorius *et al.*, 2019] and Bipedal [Hong and Lee, 2017] robots. For the other ER approaches, only Back-to-Reality and the SNS approach have been validated on significantly different real-world, complex robot morphologies.

ER approaches that do not require an initial simulator to be developed before the ER process can be advantageous. A simulator can be built and trained during the ER process, such as in BNS approaches. For the Genetic Programming approach, the simulator can be develop before or during the ER process [De Nardi, 2010]. Other ER approaches require at least an initial simulator development process.

Physics-based simulation approaches require a significant amount of prior knowledge. Minimal simulations are reliant on specialised domain knowledge. Behavioural-based modelling techniques do not rely on prior knowledge but instead use empirically collected data and machine learning regression and classification techniques. The Genetic Programming, SNS and BNS approaches do not require significant amounts of specialised knowledge to develop a simulator.

An ER approach has a narrowing reality-gap if inaccurately simulated behaviours are avoided or corrected during the ER process. All bidirectional ER approaches achieve a narrowing reality-gap using real-world feedback. The Transferability [Koos *et al.*, 2013a] and Intelligent Trial-and-Error [Cully *et al.*, 2015] approaches do not improve an existing simulator but construct a surrogate of the transferability or performance during the ER process which is used to reduce the reality-gap.

The development, augmentation or tuning of a simulator requires the collection of significant amounts of real-world data. Behavioural data can be collected by evaluating

randomly generated commands on a real-world robot. Many of these randomly generated behaviours are not required for training an effective simulator. Smarter data collection techniques ideally focus on collecting behavioural data related to simulator weaknesses or are specific to the chosen problem. All bidirectional ER approaches rely on smart, non-randomised data collection techniques specific to the given problem.

Few researchers are actively investigating Machine Learning based simulation techniques and their integration into the ER process. Genetic Programming based ER approaches appear promising, however, no existing work has demonstrated the approach on more complex robot morphologies. The SNS approach has been demonstrated to be generalisable and require little specialised knowledge but is lacking in terms of the other important goals. The BNS approach is closest to performing well in all specified ideal goals in Table 2.2, however, the BNS approach has not been demonstrated to generalise as of yet.

In the following chapters, the BNS approach is demonstrated to generalise across two significantly different complex robot morphologies. The SNS and BNS approaches are closely related to each other and differences between these approaches are worth investigating.

2.7 Conclusions

Physics-based simulators require prior expert knowledge, can be time-consuming to build and are difficult to automate. The feasibility of developing high fidelity simulators decreases as robotic systems become more complex. Physics-based simulators tend to be fine-tuned and specialised to model a particular robotic system and tend to not generalise without significant human interventions.

ANNs are becoming more advanced and capable of handling complex relationships. Machine learning approaches have helped automate the development of solutions to problems, such as video game playing AI and self-driving cars. It is not inconceivable that a machine learning approach to simulator development could one day outperform traditional approaches in certain domains. Simulating complex robotic systems can be greatly simplified through the use of SNNs. Investigating approaches that reduce the need for hu-

man interventions in the development of simulators could greatly increase the autonomy of future robotic systems.

SNNs have not been thoroughly studied on complex robots and high dimensional controller designs without built-in prior knowledge. The BNS approach combines advantages seen in both bidirectional and SNS ER approaches. No studies have investigated the BNS approach on complex robots with scripted controller designs. No experimental comparisons have been performed between the SNS and BNS approaches on significantly different classes of complex robots.

Chapter 3

EXPERIMENTAL METHOD

3.1 Introduction

This research studies the use of SNNs as an alternative to physic-based approaches. Two classes of ER approaches using SNNs have been proposed in prior work. Namely, the SNS and BNS approaches. Research into the SNS and BNS approaches have been largely limited to simple robots or problems. This work attempts to scale up and generalise these ER approaches for more complex robots (Section 3.2).

The ER process optimises controllers for a particular robotic task. Before conducting any experimental work, a controller design needs to be decided upon. A controller design can be high level, consisting of parameters specific to a mathematical function with built-in knowledge on movement strategies. Alternatively, a low level controller design consists of raw joint angle changes over time. A low level controller design uses no specialised prior knowledge that can simplify the controller solution space.

A low level controller design is chosen for the experimental work which significantly increases the difficulty level of the optimisation process due to the high dimensionality of the search space. SNNs are also required to simulate low level commands and are not able to take advantage of a parametrised controller design. The controller design and data collection process are discussed in Sections 3.3 and 3.4, respectively.

The real-world implications of certain experimental methodologies are not readily apparent and careful planning is required. There are practical implications and limitations

in dealing with real-world robotic systems. Challenges experience in this research are discussed in Section 3.5.

In order to perform statistically significant comparisons between proposed variations to the SNS and BNS approaches, experiments are designed to gather a quantitative level of data. Two different classes of robot morphologies are selected in order to validate the generalisability of the SNS and BNS approaches. Few studies investigate ER approaches on different classes of complex robots. The proposed improvements to the SNS and BNS approaches are tested in order to identify if improvements in transferability and performance outcomes are possible. The experimental methodology is covered in Section 3.6.

A detailed explanation of the proposed adaptations applied to the SNS and BNS approaches are discussed in Section 3.7. Lastly, conclusions are presented in Section 3.8.

3.2 Robot Morphologies

Two different classes of robot morphologies are studied. Hexapod walking gaits require coordination between multiple limbs. The Snake robot relies on crawling behaviours. Dynamics involved in simulating behaviours for the Snake and Hexapod robots are significantly different.

These two robots are investigated on order to study the generalisation potential of the SNS and BNS approaches. The successful application of the SNS and BNS approaches on both robots morphologies, in addition to applications in previous studies, would strongly indicate that the tested approaches could generalise across many other classes of robots. The Hexapod and Snake robot morphologies are described in Sections 3.2.1 and 3.2.2, respectively.

3.2.1 Hexapod Robot

Hexapod robot platforms are commonly used in robotics research. The Hexapod robot used in this research is shown in Figure 3.1. The robot consists of 18 Dynamixel AX-12 joint motors. Each leg consists of 3 joint motors. Motors are controlled using a Arbotix-M micro-controller. No mechanical wheel mechanisms are present on the robot. Commands are sent to the robot over a Bluetooth serial interface. The robot is powered through a

tethered connection. Tethered power ensures stable, consistent power without practical issues related to battery technologies. No sensors are used for the robot morphology.

The robot operates on a horizontal, flat surface. Locomotion is achieved through the coordinated movement of six limbs. The robot has two coloured markers placed on its back. The different coloured markers allows for the colour-based tracking system to discern heading. The position of the robot is calculated as the midpoint of the tracked markers. A camera is placed above the robot in order to record the position of tracking marker positions.

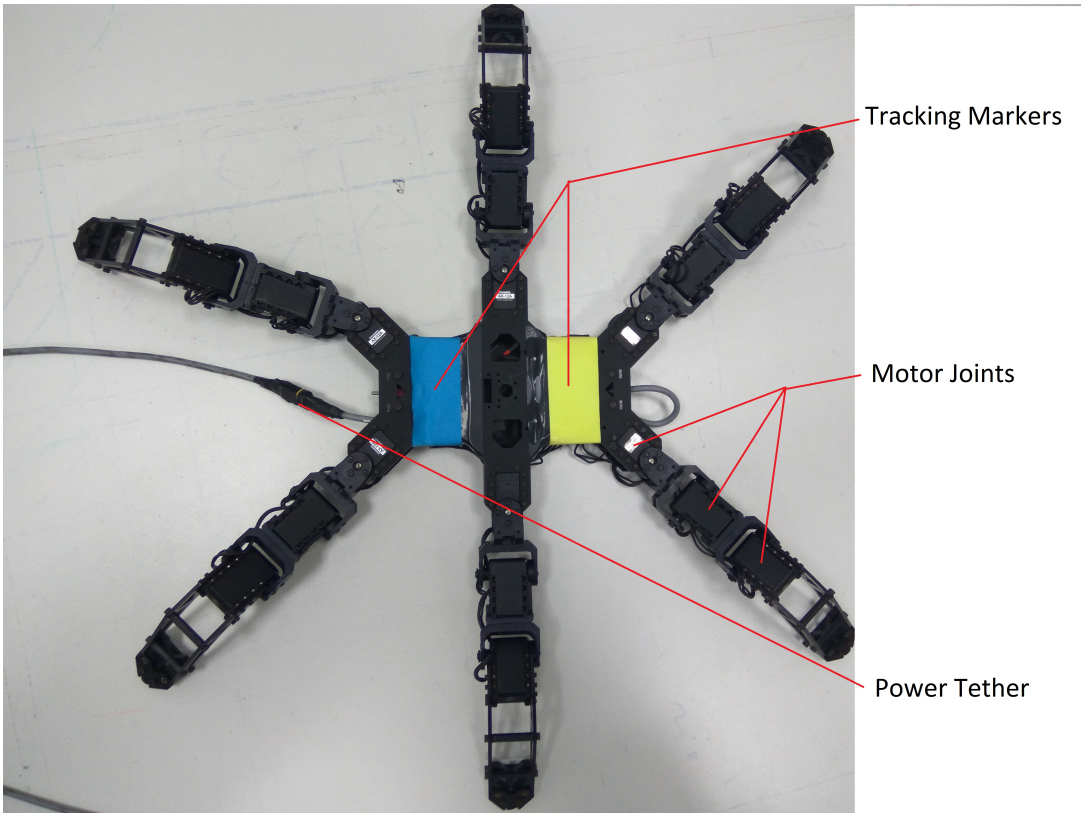


Figure 3.1: Hexapod robot

3.2.2 Snake Robot

No known work has investigated bi-directional ER approaches using a Snake robot. The Snake robot used in this work has more degrees of freedom than prior work in the ER field. Most controller designs for Snake robots are simplified by being based on mathematical

equations, curve fitting procedures or Central Pattern Generators. The controller design used in this work is not simplified and do not make use of prior knowledge of snake locomotion. This research specifically utilised a Snake robot morphology due to the lack of existing work in ER and the high complexity and novelty of the dynamics involved. The Snake robot is arguably more complex compared to most limbed morphologies.

The weight distribution properties produced by controller solutions are important due to friction with the working surface greatly affecting the robot's performance. Taking into account all the factors required to adequately simulate the Snake robot using a physics-based approach would be a significant challenge. SNNs are easier to build in comparison to a physics-based simulator.

The Snake robot used in this research is shown in Figure 3.2. The Snake robot consists of 12 Dynamixel AX-12 joint motors connected serially. Joint motors are controlled using a Arbotix-M micro-controller. Joints are designed to alternate between moving the robot vertically and laterally. The height and width of the robot is approximately 5 cm and has a length of 114 cm. Communication is achieved using a serial data cable. Power is supplied using a tether in order to void issues related to battery technologies. No snake-like skins or mechanical wheel mechanisms are used.

The Snake robot operates on the same flat, horizontal surface as the Hexapod robot. Movement is achieve through a coordinated series of joint angle changes which produce crawling behaviours. Behavioural data is collected using camera-based tracking of the coloured markers located on the head of the robot. Yellow and blue coloured markers are located on the head. Positions are calculated as the midpoint of the tracked markers. The heading is calculated based on the different coloured markers. The orientation of the head relative to the ground is determined using an orientation sensor located on the head of the robot. Effective solution controllers require balanced behaviours that keep the head relatively upright, otherwise, the robot could roll onto its back. The tracking system is unable to track the robot if the head is upside down.

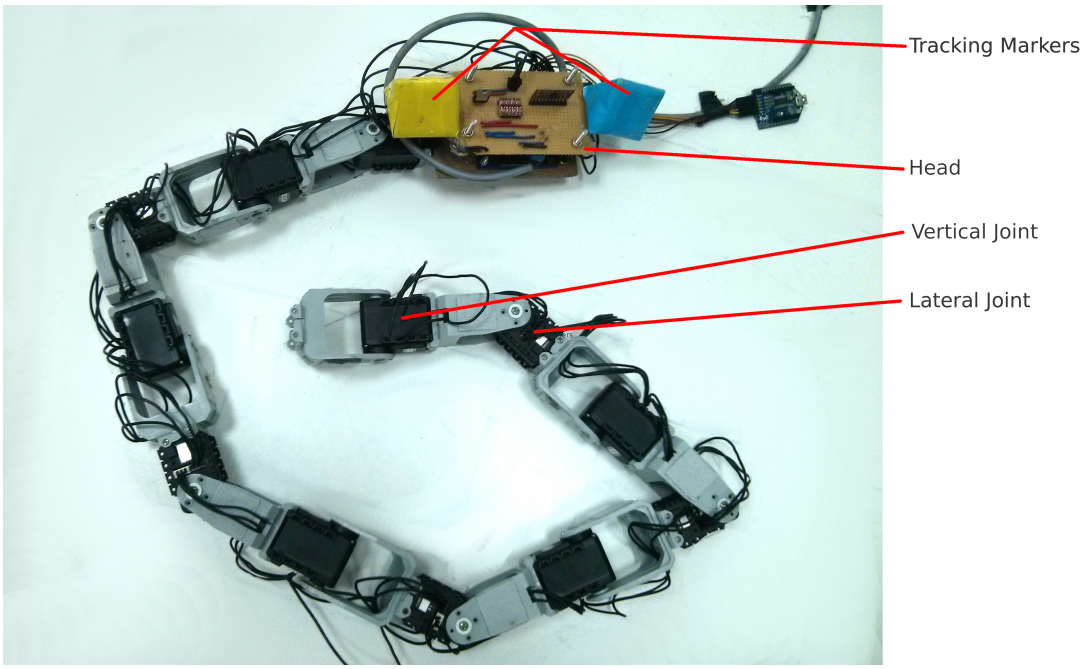


Figure 3.2: Snake robot

3.3 Controller Design

At the beginning of every controller evaluation, all joint angles start off in a default position. The Snake robot starts off completely straight. The starting position for the Hexapod robot is when all joint positions are straight and all feet touch the ground.

Each controller consists of a sequential list of commands (Figure 3.3). A command contains a set of joint angle changes to be applied to a robot's joint angles. The j^{th} joint is changed by α_{ij} units when the i^{th} command is executed. Commands are sent to the robot sequentially and joint angles are changed accordingly.

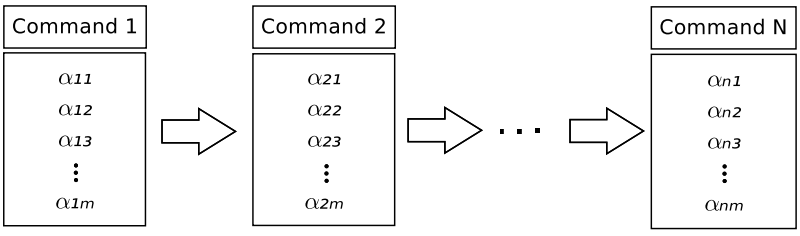


Figure 3.3: Controller example

The controller design is purposely an open-loop controller with no feedback mecha-

nisms. During controller evaluations, robots cannot perform corrections in their positions over time. This allows for the study of how accurate the developed simulators are at predicting the accumulated consequences of simulated behaviours over many commands. For controllers to be useful in practice, future work might be required to transition to a close-loop controller design.

3.4 Data Collection

For each command sent to the robot, a particular behaviour is observed. The behaviour specific to each command sent, broken down into behavioural components is recorded. The motion breakdowns, sideway displacement (Δx), forwards/backward displacement (Δy), heading displacement (Δa) and head orientation displacement (Δo) will be referred to as behavioural components of the robot. The behavioural components are illustrated in Figure 3.4. Changes in behaviour for a given command are calculated relative to the starting position before the command is executed.

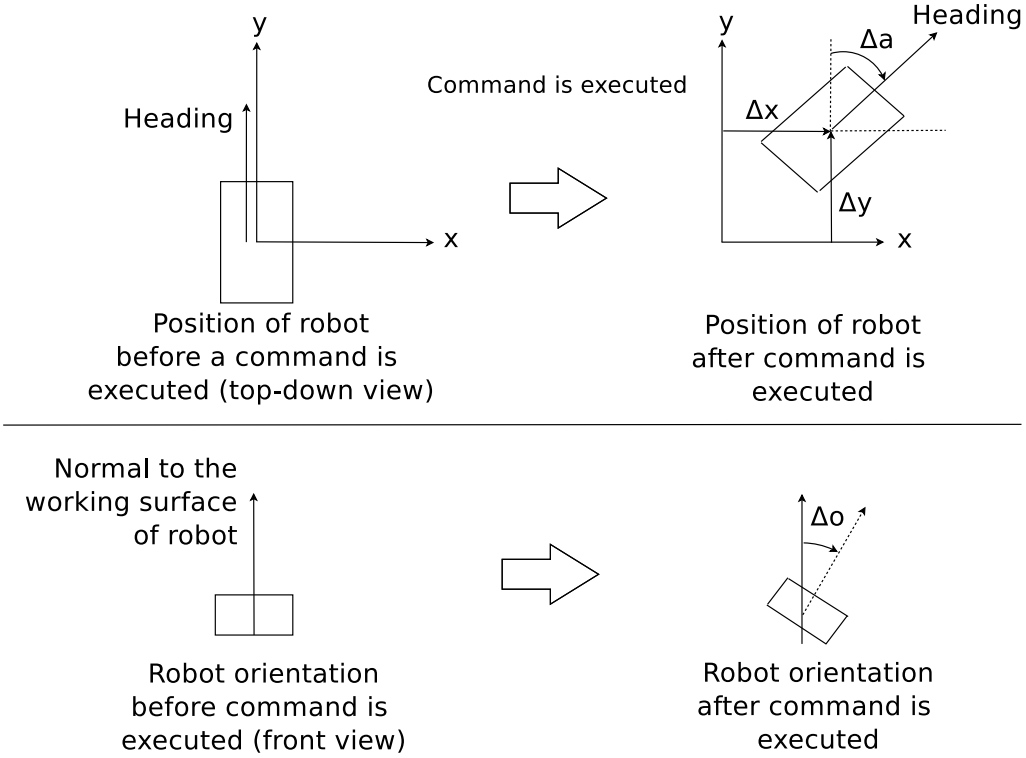


Figure 3.4: Behavioural Components

A camera mounted above the robot records changes in position and heading of the tracking markers. Positions are calculated as the midpoint between coloured markers. Energy, torques, power usage and speed of controllers are not measured or simulated in any way.

The execution of a controller and related data collection is illustrated in Figure 3.5. The robot has a total number of m joint motors. All joint angles are considered to be zero in the default starting position. Commands are sequentially executed on the robot. A single command consists of a list of joint angle changes to be applied to the robot's joint angles (represented as α_{ij}, where i is the command being evaluated and j is the joint number). The starting and final joint positions are captured for every command. Starting and final joint positions are represented by β_{ij} and δ_{ij}, respectively. A controller consists of n commands. For the i^{th} command, the robot's relative change in position in the perpendicular direction (Δx_i), parallel direction (Δy_i) and heading (Δa_i) are captured. Only the Snake robot has an orientation sensor. The snake's head orientation relative to the ground at the beginning of the i^{th} command evaluation is captured as θ_i. The change in the Snake robot's head orientation relative to the ground (Δo_i) for the i^{th} command is also captured. Simulating only the head orientation relative to the ground is computationally more efficient than simulating the yaw, pitch and roll separately.

3.5 Challenges

Currently, no ER approaches achieve a level of automation required to develop a truly autonomous robotic system. Human interventions are still required in the planning and execution of the ER approaches. This section discusses some practical challenges encountered.

A tethered power connection is used in order to avoid problems associated with portable power solutions. For untethered robots, this work would require Lithium Polymer batteries. Batteries would significantly increase the weight on the robots, adding additional strain on motors which increases the likelihood of damage. Robot behaviours are less consistent if power levels fluctuate based on the charge level of batteries. A power tether guarantees consistent, stable power without the need to recharge batteries.

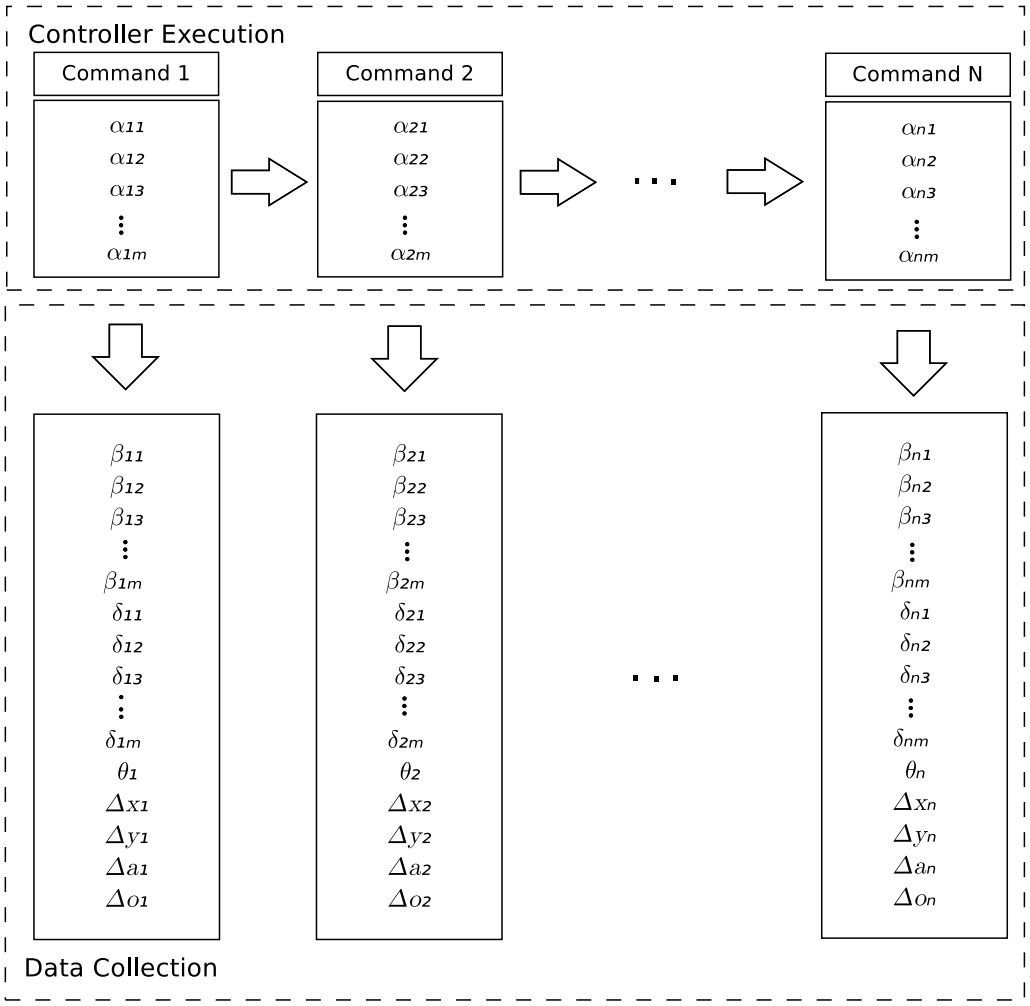


Figure 3.5: Data Collection

Executing the exact same controller in reality multiple times produces slightly different behaviours. The power tether introduces a small force on the robots when dragged on the working surface. Depending on the positioning of the tether, there can be significant differences in robot behaviours. Fortunately, scenarios where tether positioning greatly influences behaviours are not a common occurrence. The tether positioning or drag force is not directly simulated in this research and is only indirectly simulated through observed behaviours.

Human intervention is required in order to effectively collect behavioural data from commands executed on the robot hardware. Robots are manually repositioned once outside the bounds of the working surface. Manual interventions may be required to avoid

the robot stepping on the power tether or becoming entangled. Unforeseen problems may occur during the data collection process that may require human intervention. Examples include motor failures, overheating, loose connections or the tracking system might not pick up certain robot states. The tracking system is designed to rely on robots remaining fairly upright. Frequent pauses in robot operation are required in order to avoid overheating of joint motors.

Joint motors are manually replaced due to damage caused by overuse. The most common cause of motor failure is gear damage. The gears inside the motors consist of plastic. Significant levels of torque over a period of time wears down these gears until certain gear positions can no longer be maintained.

If the temperatures of motors are ignored, a slow degradation in performance is observed. Robot behaviours become less reliable and the frequency at which motors fail increases. Overheating is significantly worse on the Hexapod robot when compared to the Snake robot. This is probably due to the motors on the Hexapod robot moving faster than the Snake robot. Motor joints on the Hexapod robot typically carry the weight of the body above the working surface. The weight of the Snake robot's body is usually more distributed on the working surface.

All joint motors have torque limits. If a motor maintains a certain position and a high torque is experienced for too long, the motor will shut off in order to prevent damage. Torque limits mainly affect motors on certain parts of the robot. On the Hexapod robot, the middle joint motors on each leg are most affected by torque failures. For the Snake robot, joint motors close to the centre of the robot are most likely to fail.

Certain robot behaviours are more likely to trigger torque limit failures than others. In order to reduce the likelihood of motors reaching torque limits on the Snake robot, the range of motion for the vertical joints are half that of the lateral joints. This reduces how high the body can lift off the ground, thereby reducing the strain on motors. Scheduled pauses in robot operation reduces the likelihood of motors failing. It should be noted that the simulator does not model torque values or motor failures.

Unlike the Hexapod robot, the same sequence of commands executed on the Snake robot can result in drastically mixed behavioural outcomes if the starting head orientations are different. The Snake robot requires the modelling of the head orientation while

the Hexapod robot is less sensitive to orientation states. The camera-based tracking system is not designed to take into account rolling behaviours for the Snake robot. If the Snake robot rolls onto its back during a controller evaluation, the controller is considered a failure. In order to avoid rolling behaviours, Snake robot controllers are evolved to remain relatively upright throughout controller evaluations. If the head orientation is not accurately simulated, it is likely that many solution controllers will fail.

3.6 Methodology

Three stages of experimental work are conducted. The first stage, Methodology A (Figure 3.6), investigates proposed enhancements (adaptations) applied to the SNS approach. The second stage, Methodology B (Figure 3.7), investigates proposed adaptations to the BNS approach completely in simulation, without the use of a real-world robot. The last stage, Methodology C (Figure 3.8), demonstrates and validates the promising adaptations discovered during the Methodology B experimental work. Adaptations can consist of modifications to the simulator configurations, training, data collection and controller evolution process.

The SNS approach can be modified in various ways (Section 3.7). Adaptations to the SNS approach are investigated in Methodology A (Figure 3.6). The adaptations are investigate through a series of experiments. The experimental work produces a set of controller solutions associated to each adaptation tested. Comparing these sets of controllers against one another provides insight into differences between the tested adaptations.

Methodology B investigates the BNS approach and is conducted completely within a simulated environment in order to benchmark the large number of proposed adaptations. A statistically rigorous analysis of many adaptations is practically infeasible on real-world robots. Performing such a large number of experiments in reality would take over a year to complete. It would also be financially costly in terms of repairs required from the wear and tear. However, conducting the experimental work for Methodology B completely in simulation over a large number of computers is achievable. A quantitative analysis of many adaptations is only feasibly by replacing the real-world robot with the simulated alternative developed in Methodology A (Figure 3.6). Static SNNs are used as a replacement

for a real-world robot in Methodology B. Simulator noise is always added to sampling evaluations performed using the Static SNNs in order to represent noise present in reality. Conducting experimental benchmarks completely in simulation due to the infeasibility of a large number of real-world experiments has been seen in prior robotics research [Klaus, Glette, and Tørresen, 2012].

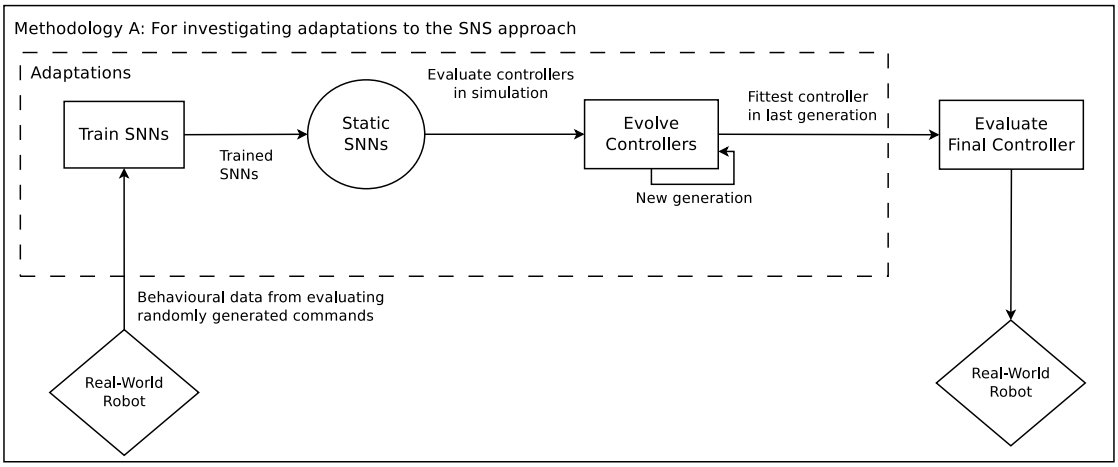


Figure 3.6: Methodology A: Adaptations to the SNS approach

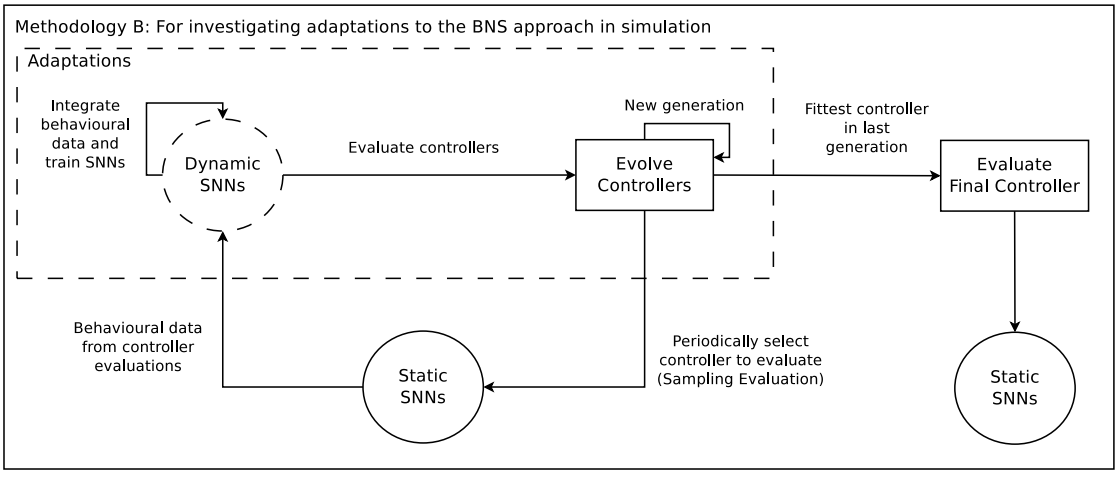


Figure 3.7: Methodology B: Adaptations to the BNS approach

The third stage of experimental work is required in order to validate and demonstrate that top performing BNS adaptations discovered in Methodology B are viable for a real-world robot. Methodology C (Figure 3.8) selects the top performing BNS adaptations based on the Methodology B results and repeats the experimental work on a real-world

robot. Only a small number of adaptations can be feasibly investigated in reality. In Methodology C, adaptations are studied on a qualitative level.

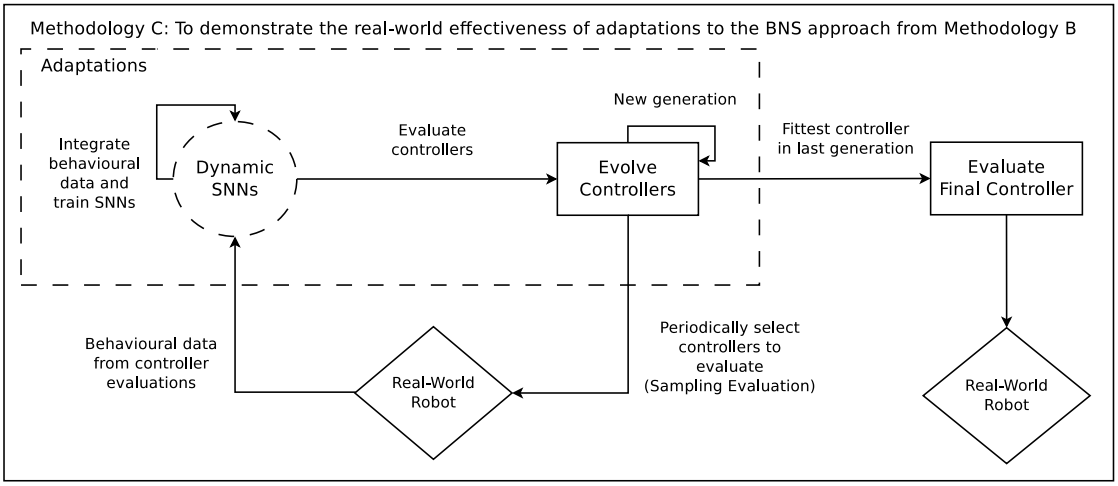


Figure 3.8: Methodology C: Real-world validation of adaptations

The problems chosen for the experimental work is intentionally open-ended. There exists a large number of possible good solutions. Robots are required to maximise the total distance travelled after a fixed number of commands. The performance and transferability of solution trajectories are calculated.

The simulated and real-world trajectories of solution controllers are recorded. The robot's final positions in simulation and reality are denoted by $(x_s,\ y_s)$ and $(x_r,\ y_r)$, respectively. The starting position is denoted by $(o_x,\ o_y)$. The angle between the final simulated and real-world position vectors is denoted as ϕ. Controller performance is calculated as the distance of the projection of the real-world final position vector onto the simulated final position vector (equation (3.1)). The performance metric measures the real-world distance travelled by the robot relative to the simulated final position. Controllers ideally travel a great distance, however, trajectories significantly different to the simulated direction are considered bad. The chosen performance metric captures the distance travelled in the correct trajectory.

$$F = \sqrt{(o_x - x_r)^2 + (o_y - y_r)^2} \times \cos\phi \tag{3.1}$$

A transferability metric is designed to study differences between the transferability

properties of the tested adaptations. The transferability metric is defined as the Euclidean distance between the final positions of the robot in simulation and reality, divided by the Euclidean distance between the starting and final positions of the robot in reality (equation (3.2)). The transferability metric can be thought of as having two components. Firstly, an error measure for the gap between simulation and reality. The other component is a scaling factor that adjusts the error measure according to the real-world distance travelled. Good transferability indicates that the simulated and real-world final positions are relatively close when taking into account the real-world distance travelled.

$$T = \frac{\sqrt{(x_s - x_r)^2 + (y_s - y_r)^2}}{\sqrt{(o_x - x_r)^2 + (o_y - y_r)^2}} \tag{3.2}$$

To compare the transferability and performance differences between adaptations, pairwise Mann-Whitney U tests are performed. The Mann-Whitney U test is a non-parametric test and does not assume normally distributed datasets. This test is also a more conservative comparison of differences between distributions than other statistical tests. No p-value adjustments are performed which will result in a small percentage of the pairwise comparisons having a Type 1 error. The Mann-Whitney U test is already quite conservative and many pairwise comparisons share common adaptations which means that adjusted p-values would become overly conservative. No more than 10 groups are compared when using pairwise comparisons. All experiments are planned beforehand and no decisions or reporting are performed based on the pairwise comparison results. Statistical outcomes are used to confirm prior theoretical predictions. Theories and statistical outcomes are also analysed and reported through multiple experiments across different robot morphologies.

3.7 Adaptations

The proposed enhancements to the SNS and BNS approaches are called adaptations. Certain adaptations can be applied to both the SNS and BNS approaches while others are only relevant to the BNS approach. Figure 3.9 is a Venn diagram that represents the SNS and BNS approaches. The top circle represent the SNS approach while the bottom circle represents the BNS approach. Each adaptation tested is listed within these circles.

The first adaptation, Simulator Configuration, investigates different simulator architectures (Section 3.7.1). The second adaptation, Simulator Noise, is a technique where noise is intentionally added or excluded in controller evaluations in simulation (Section 3.7.2). These first two adaptations can be applied to both the SNS and BNS approaches.

The following three adaptations are only relevant to the BNS approach. For the BNS approach, controller evolution and simulator training are conducted concurrently. This research proposes that periodically resetting the controller population and/or simulator weights during the BNS approach could be beneficial. Controller Resetting and Simulator Resetting procedures are explained in Sections 3.7.3 and 3.7.4, respectively. The last adaptation, Sampling Strategies, is the selection technique used to pick controllers from the controller population for behavioural data acquisition purposes (Section 3.7.5).

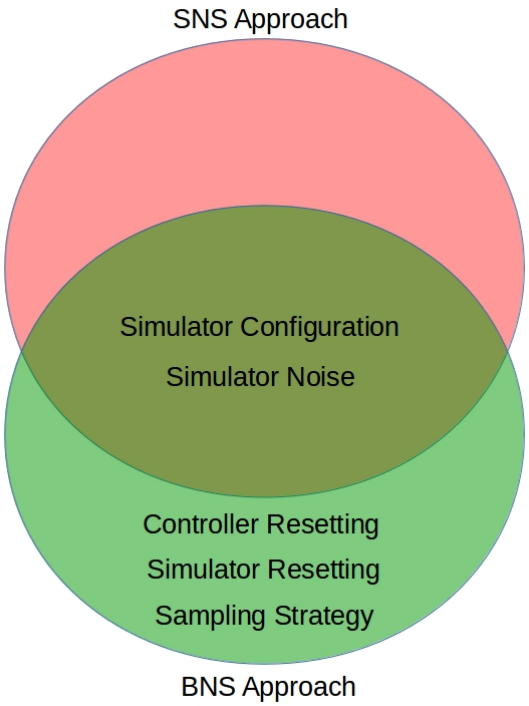


Figure 3.9: Venn Diagram of SNS and BNS adaptations

3.7.1 Simulator Configurations

SNNs have to deal with the bias-variance trade-off problem (Section 2.2.5). Training ensembles of SNNs to simulate particular behaviours could potentially help reduce the bias and variance experienced. Training multiple independent SNNs to predict the same behavioural components may provide significant improvements to the SNS and BNS approaches.

An additional improvement to consider is uncertainty estimation (Section 2.2.4). Disagreement between SNN predictions could be used to create an uncertainty measure. Any high levels of uncertainty could be penalised during controller evolution. The ER process could take into account the accumulated uncertainty of controller behaviours and avoid inaccurately simulated controllers. This would ideally direct the evolution process into more accurately simulated regions of the search space.

Prior work determined that robot behaviours are most accurately simulated when each SNN only models a single behavioural component as output. SNNs with multiple behavioural component outputs have not been investigated for complex robots. This work investigates SNN configurations that simulate all behavioural components at once. No prior work has used SNNs to simulate the head orientation of a Snake robot.

The Simulator Configurations investigated are:

- Basic (Section 3.7.1.1)
- Dropout (Section 3.7.1.2)
- Ensemble (Section 3.7.1.3)
- Basic Multi-output (Section 3.7.1.4)
- Ensemble Multi-output (Section 3.7.1.5)

Configurations that generate uncertainty information are the **Dropout**, **Ensemble** and **Ensemble Multi-output** simulator configurations. The two techniques for producing uncertainty information is enabling dropout layers during predictions (**Dropout** configuration) and grouping SNNs into ensembles (**Ensemble** and **Ensemble Multi-output** configurations).

3.7.1.1 Basic Configuration

The **Basic** configuration (Figure 3.10) consists of either three (Hexapod) or four (Snake) SNNs, one for each of the behavioural components simulated. This simulator configuration has been investigated in prior studies [Pretorius *et al.*, 2013; Woodford *et al.*, 2015, 2016].

The architecture of each SNN is illustrated in Figure 3.11. The size and number of hidden layers used depends on the robot and behavioural component simulated. The input layer takes as input the current and transition joint angles of a particular robot command. The Snake robot additionally takes as input the starting orientation of the head of the robot relative to the working surface before the command is executed. Both robot morphologies require SNNs predicting the robot's sideways trajectory (Δx), forward-backwards trajectory (Δy) and the change in heading (Δa). An additional SNN is required for the Snake robot in order to predict the change in the robot's head orientation relative to the ground (Δo).

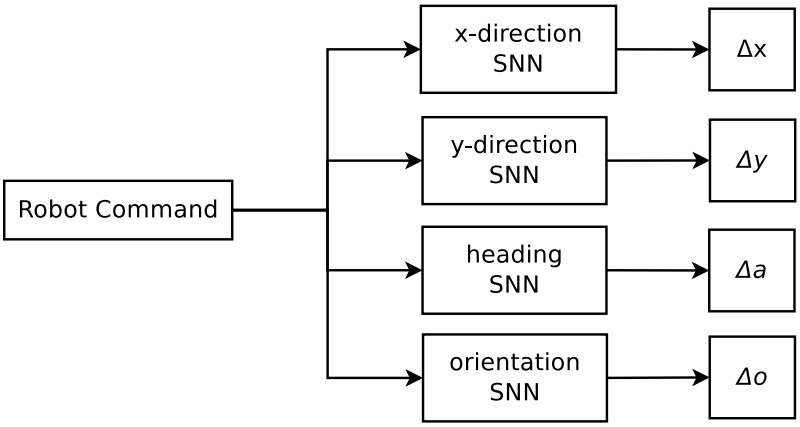


Figure 3.10: Basic Configuration

The dropout layers seen in Figure 3.11 are used to reduce over-fitting during training. During the training phase, hidden layer nodes are randomly enabled or disables. Dropout nodes have a dropout rate of 50%. The ReLU activation function is used for all hidden and output layer nodes. Once a SNN is trained, all nodes are utilised during controller evaluations. The **Basic** configuration does not produce uncertainty information during controller evolution.

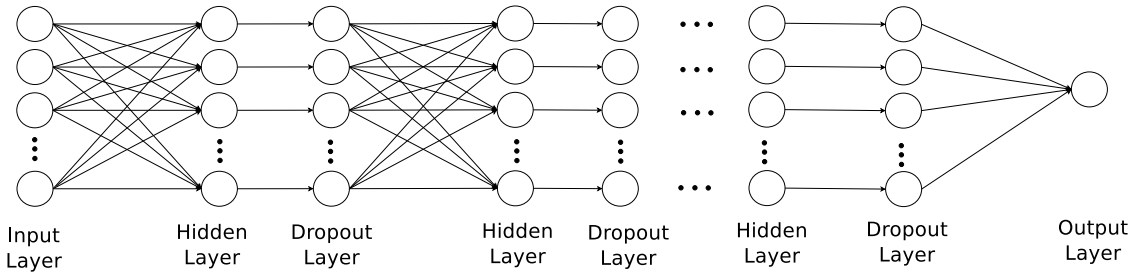


Figure 3.11: SNN Architecture

3.7.1.2 Dropout Configuration

The **Dropout** configuration is identical to the **Basic** configuration, except that the dropout layers are also enabled during controller evaluations. Multiple forward passes of the same command produce different outputs due to hidden layer neurons being randomly turned on or off. For each SNN in the **Dropout** configuration (Figure 3.12), multiple predictions for the same command are evaluated with the dropout layers enabled, generating multiple predictions per behavioural component. The mean and standard deviation of these predictions are calculated. The mean predictions become the simulated behaviour and the standard deviations are a measurement of uncertainty. The fitness function is designed to penalise the accumulated uncertainty measurements for controllers. Highly penalised controllers are less likely to survive during controller evolution. Enabling dropout layers during controller evaluations is a novel proposal for the SNS and BNS approaches.

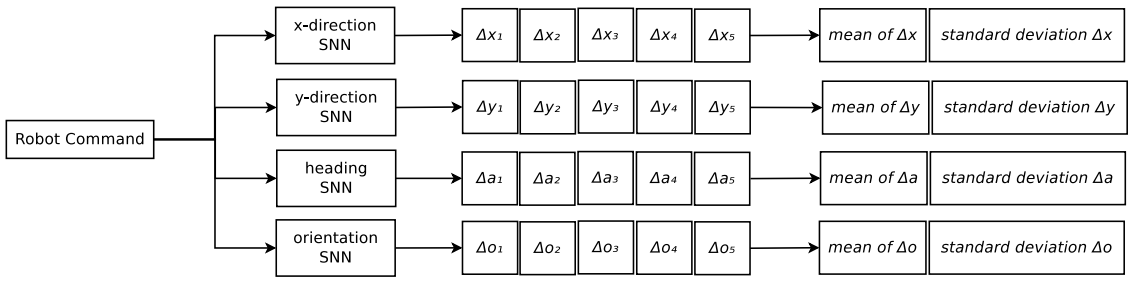


Figure 3.12: Dropout Configuration

3.7.1.3 Ensemble Configuration

An **Ensemble** configuration (Figure 3.13) involves training multiple **Basic** configurations and grouping the SNNs according to the behavioural components modelled. Multiple **Basic** configurations simulate each behavioural component. This is the most computationally expensive simulator configuration tested. Every SNN is randomly initialised with a different set of weights. SNNs are trained independently using the same behavioural dataset. The trained SNNs contain different weight settings even when simulating the same behavioural component. Configuring SNNs into ensembles is a novel proposal for the SNS and BNS approaches.

Each SNN will output a different behavioural component prediction for the same command due to different weight settings. This produces a distribution of predictions for each behavioural component. The mean and standard deviation for each behavioural component is calculated. The means become the simulated behaviour and the standard deviations are utilised in fitness penalisations. The standard deviation in predictions is used as a source of uncertainty information (Section 2.2.4). A group of SNNs simulating the same behavioural component reduces the variance of predictions at no cost to the bias (Section 2.2.5). Ensembles rely on simulating behaviours from multiple SNNs and averages out the simulated errors (Section 2.2.6).

3.7.1.4 Basic Multi-output Configuration

The **Basic Multi-output** configuration is illustrated in Figure 3.14. It is the most computationally efficient configuration tested. The **Basic Multi-output** configuration simulates all behavioural components using a single SNN. The SNN architecture used is shown in Figure 3.15. A multi-output SNN architecture has three or four outputs depending on the number of behavioural components simulated. Dropout is only used during training. The configuration does not produce any uncertainty information.

Prior work has found that this configuration tends to model behaviours less accurately compared to single-output SNN architectures [Pretorius *et al.*, 2009]. However, multi-output SNNs have not been investigated for complex robots. This research is the first time that multi-output SNN architectures are investigated on complex robots.

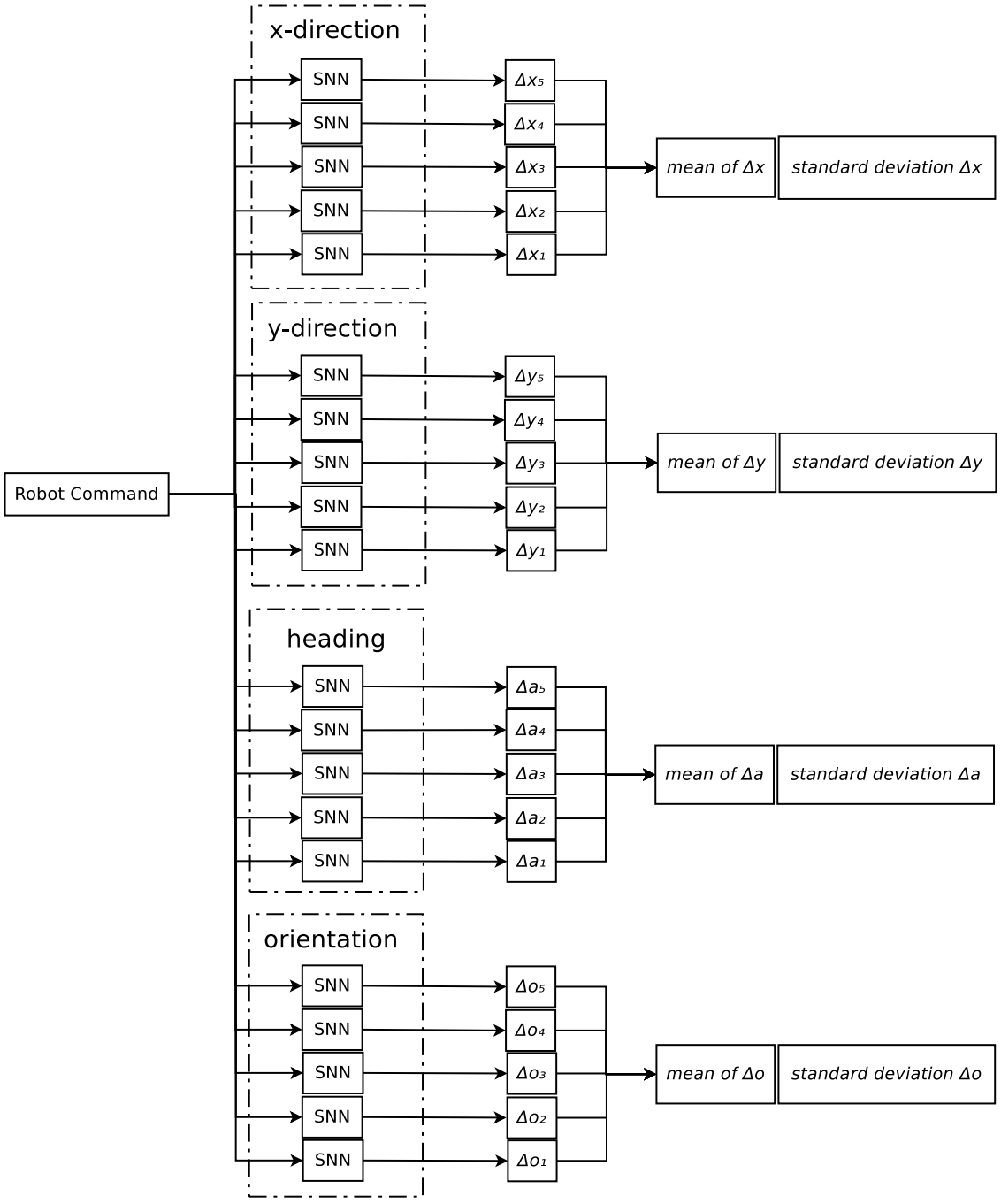


Figure 3.13: Ensemble SNN Configuration

3.7.1.5 Ensemble Multi-output Configuration

The **Ensemble Multi-output** simulator configuration (Figure 3.16) consists of many **Basic Multi-output** simulator configurations in an ensemble. The configuration is a

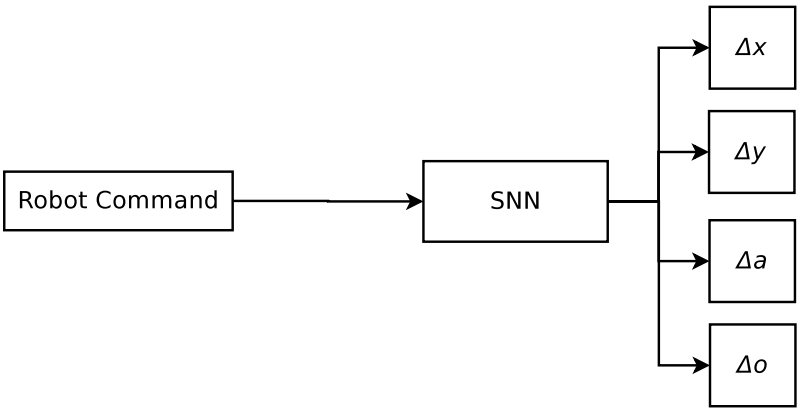


Figure 3.14: Basic Multi-output SNN Configuration

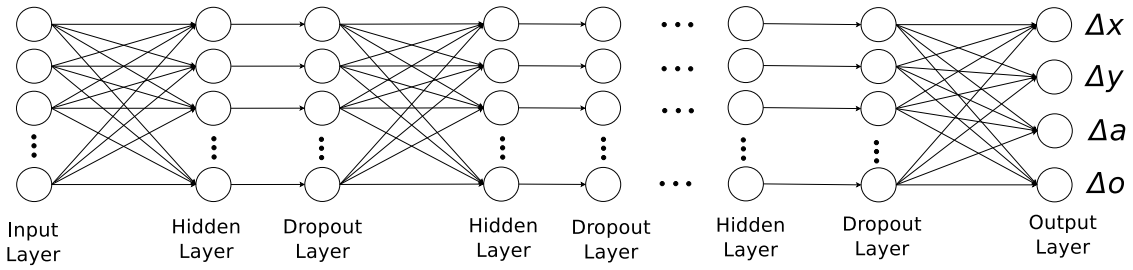


Figure 3.15: Multi-output SNN architecture

mixture of the **Ensemble** and **Basic Multi-output** simulator configurations. SNNs are independently initialised with different randomised weight settings and trained using the same behavioural dataset. Each SNN predicts all behavioural components. For any given command, a distribution of predictions is obtained and predictions are grouped into their respective behavioural components. The mean and standard deviation of each behavioural component is calculated. The mean values become the simulated behaviours and the standard deviation are used for uncertainty penalties (Section 2.2.4). Similar to the **Ensemble** configuration, ensembles of SNNs helps reduce the variance of predictions at no cost to the bias (Section 2.2.5). Simulated errors are averaged out over multiple SNN predictions (Section 2.2.6). Multi-output SNNs configured in ensembles is a novel proposal of this thesis.

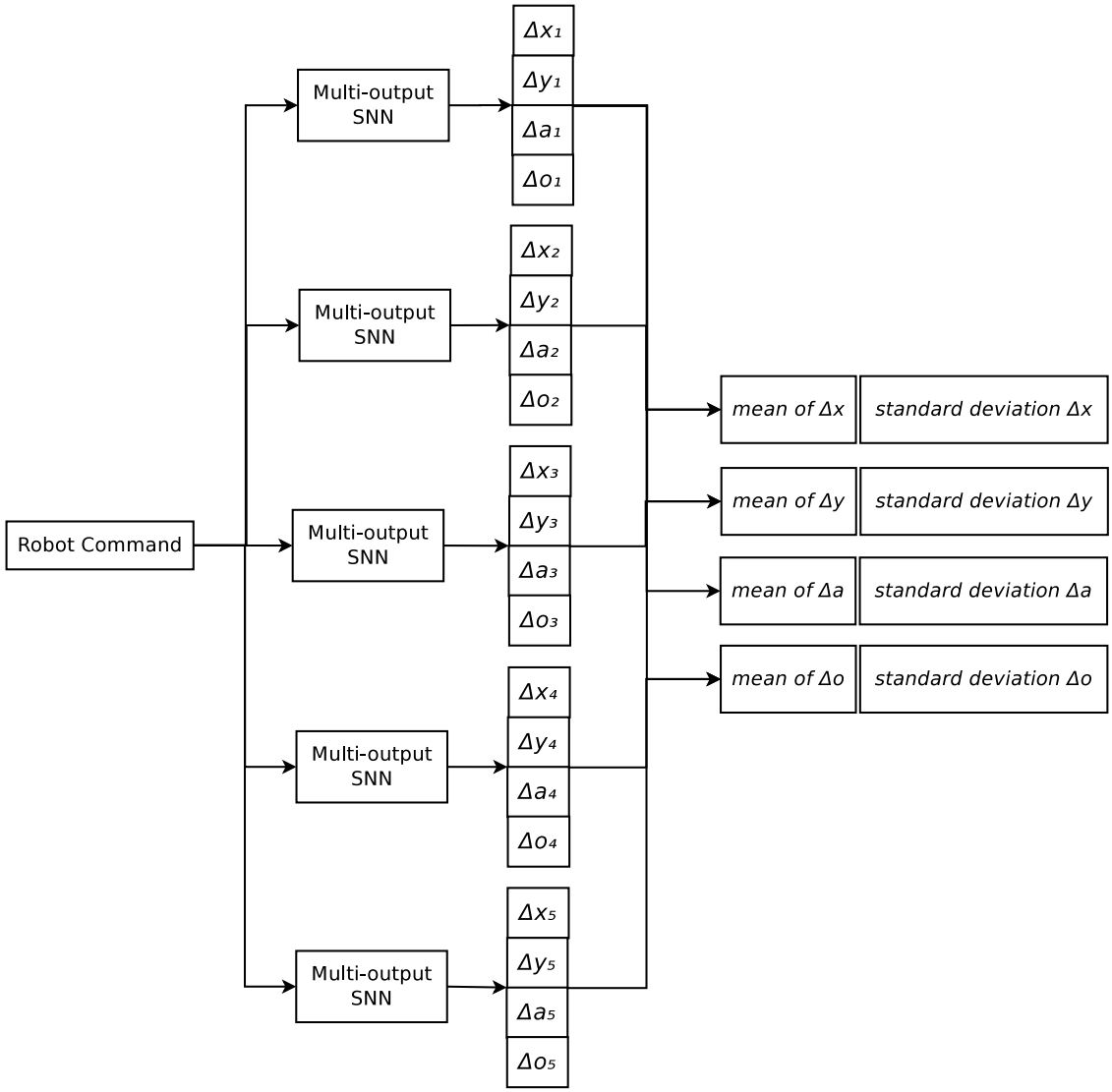


Figure 3.16: Ensemble Multi-output SNN Configuration

3.7.2 Simulator Noise

Real-world robotic systems contain noise which can result in significant differences in behaviours even when actions applied are identical. The noise component of reality is often modelled and integrated into simulators. The inclusion of simulator noise has been shown to improve the transferability of evolved behaviours from simulation to reality [Jakobi, 1997; Jakobi *et al.*, 1995]. The addition of simulator noise is commonly used in ER [Bongard, 2013; Bongard and Hornby, 2013; Doncieux and Mouret, 2014; Miglino

et al., 1995]. The inclusion or exclusion of simulator noise is an adaptation investigated in this research.

Controllers are evaluated multiple times in simulation with noise, each evaluation produces slightly different behaviours and different fitness estimates. The average fitness over multiple evaluations is used during controller evolution. Noise is injected into predictions of each behavioural component. A Gaussian distribution is used to generate noise. For the SNS approach, simulator noise is included or excluded for Static SNN predictions. For the BNS approach, simulator noise is included or excluded for Dynamic SNN predictions. Importantly, in order to more accurately represent reality, simulator noise is always added to the Static SNN predictions in Methodology B.

For the Hexapod robot, prior work has demonstrated that not using simulator noise for the SNS approach can lead to improved performance outcomes for the robotic task investigated [Pretorius *et al.*, 2019]. For the Snake robot, this research is the first time that the inclusion or exclusion of simulator noise is compared for the SNS approach. This thesis presents a first time investigation comparing the inclusion or exclusion of simulator noise for the BNS approach.

3.7.3 Controller Resetting

During the early stages of the BNS approach, the controller population typically converges towards poor solutions due to a largely untrained simulator. Over time the simulator improves as more behavioural data is collected and used to improve the simulator. The controller population converges towards better solutions as the simulator becomes more accurate. However, premature convergence during the early stages of controller evolution might impede progress towards a better convergence point as the simulator improves. Periodically restarting the controller evolution process as the simulator improves is an adaptation worth investigating. Periodically resetting the controller population during the BNS approach is a novel proposal of this research.

Controller resetting periodically eliminates the entire controller population and randomly generates a new one. The controller evolution process simply continues to evolve the new population after the reset. Other resetting procedures not investigated include partial population resetting, dynamic restart strategies based on a lack of progress and

optimising separate populations. Age-Layered Population Structure (ALPS) Evolutionary Algorithms optimise individuals based on their age and continually generates younger individuals. This research is limited to studying full resetting and no resetting procedures. Partial resetting and island-based population optimisation techniques are more advanced and may be more computationally expensive. Future work should investigate a wider variety of resetting procedures.

3.7.4 Simulator Resetting

Slowly increasing the size of the behavioural dataset used to train SNNs may have disadvantages. The Dynamic SNNs could become bias towards modelling behaviours encountered during the early stages of the data acquisition process. Learning newer behavioural patterns might take longer than necessary. Periodically reinitialising the Dynamic SNNs and retraining with all available behavioural data could help eliminate training bias. Resetting ensures that all behavioural data collected is equally considered during retraining. Periodically resetting the simulator during the BNS approach is a novel proposal of this research.

3.7.5 Sampling Strategy

During the BNS approach, controllers are selected from the controller population and are evaluated on the target robot in order to collect behavioural data. The technique used to select controllers from the controller population is called the Sampling Strategy.

The baseline Sampling Strategy used in prior work is called **High Fitness** sampling, where a tournament selection process is followed. A certain number of controllers are randomly selected from the controller population. The fittest controller from the selection is evaluated in order to obtain new behavioural patterns. Prior work found that the tournament size does not greatly influence the performance of the BNS approach and higher tournament sizes perform marginally better than small tournament sizes [Woodford *et al.*, 2016].

Controllers with the highest degree of accumulated uncertainty is chosen for real-world evaluations. A controller's accumulated uncertainty is the summation of all its prediction

standard deviations over all commands and behavioural components. Evaluations from controllers with a high accumulated uncertainty would ideally contain more valuable information compared to behavioural data collected from a **High Fitness** sampling strategy. The **Most Uncertain** sampling strategy is a novel proposal of this research.

3.8 Conclusions

The generalisability of the SNS and BNS approaches can be inferred based on the experimental outcomes of this work. Demonstrating the SNS and BNS approaches are viable on completely different classes of robots would provide strong evidence these approaches might generalise for use on other classes of robots not investigated in this research.

The controller designs used in this work do not heavily rely on specialised human knowledge. Controller solutions are not biased by human understanding. The controller solution search space is not simplified by some mathematical functions that mimics biological behaviours. The ER process evolves low-level arbitrary joint angle changes in robot joints which means that controllers have a high level of flexibility and freedom to generate arbitrary behaviours.

Experiments are designed to investigate improvements applied to the SNS and BNS approaches for both the Hexapod and Snake robots. Methodology A investigates proposed improvements for the SNS approach. Investigating the large number of possible improvements for the BNS approach is only feasible in a completely simulated environment (Methodology B). Promising improvements are selected based on observations generated by Methodology B and are validated on the real-world robots (Methodology C).

Chapter 4

HEXAPOD STATIC NEURO-SIMULATION

4.1 Introduction

Prior research has already validated that the SNS approach is viable on a Hexapod robot [Pretorius *et al.*, 2019]. This chapter builds upon prior work by investigating novel adaptations to the standard SNS approach. The experimental procedures used to investigate adaptations are discussed in Section 4.2. Experimental configurations tested consist of standard and novel adaptations which are described in Section 4.3.

Experimental results achieved in this chapter are presented and used to determine if any proposed adaptations improve upon the standard SNS approach (Section 4.4). Finally, conclusions to the work are presented in Section 4.5.

4.2 Experimental Procedure

This chapter consists of experimental work specific to the SNS approach and the Hexapod robot. Methodology A illustrated in Figure 4.1 visually demonstrates the experimental procedure followed in this chapter. Methodology A is discussed in greater detail in Section 3.6.

Methodology A requires the use of a real-world Hexapod robot only to collect behavioural data and validate the final solution controllers. Details of the chosen controller

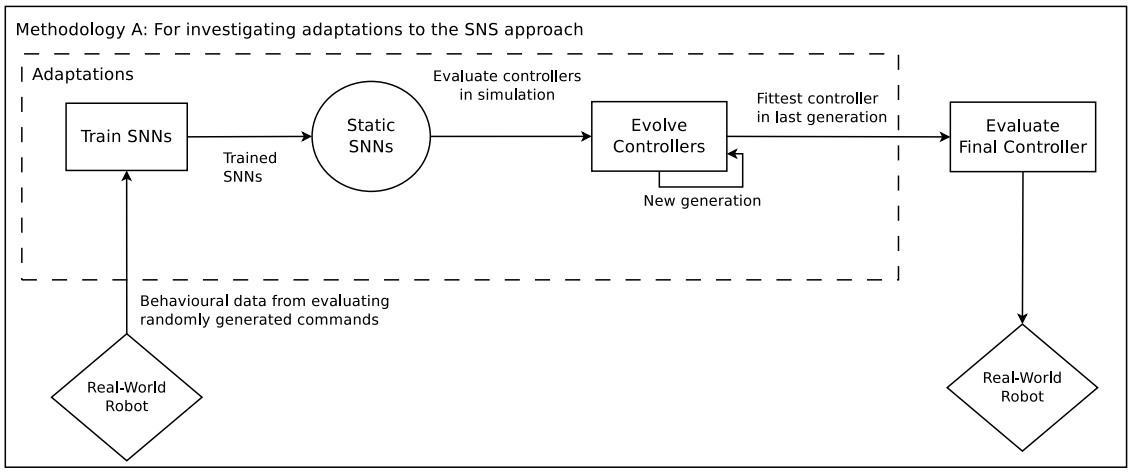


Figure 4.1: Methodology A: Adaptations to the SNS approach

design and data collection process are covered in Section 4.2.1. This chapter uses the same controller design, Hexapod robot and randomly collected behavioural data from a prior study [Pretorius *et al.*, 2019]. However, only the **Basic** simulator configuration is investigated in the prior study. Specifics regarding the tested simulator configurations and training procedures are discussed in Section 4.2.2.

4.2.1 Controllers

The controller design is also discussed in Section 3.3. Specific design decisions related to the Hexapod robot are covered in this section. Controllers are developed for an open-loop gait optimisation problem. No feedback is provided to a controller about the robot's current position or state during evaluation. The ER process optimises controllers in order to maximise the distance travelled by the robot in any particular direction. Controllers consist of a sequential list of eleven commands. A command contains the joint angle changes that need to be applied to each of the robot's 18 joints. Each command evaluation requires that all the robot's joint angles are adjusted according to the change specified by the command. Once all the joints have reach their specified positions, the next command can be executed. Joints are limited in their range of motion in order to eliminate the possibility of limbs colliding with each other or the body of the robot. Each joint can move between -26 and 26 degrees relative to the start position of a controller evaluation.

4.2.2 The Simulator

The SNN architectures, namely, the number of hidden layers and nodes for each behavioural component are based on experimental benchmarks conducted in prior research [Pretorius *et al.*, 2019]. Normally the ideal SNN architectures for a particular robotic platform would be unknown until behavioural data is collected and various SNN architectures are tested. Since proven SNN architectures have been demonstrated in prior work, this study avoids benchmarking a large number of SNN architectures. The prior study investigating the SNS approach for the Hexapod robot found that a single hidden layer per SNN is the best architecture [Pretorius *et al.*, 2019].

The **Basic**, **Dropout** and **Ensemble** simulator configurations have separate SNNs for each behavioural component simulated. All SNNs consist of a single hidden layer. The number of hidden layer neurons for the Δx, Δy and Δa behavioural components are 100, 200 and 100 neurons, respectively. The **Ensemble Multi-output** and **Basic Multi-output** simulator configurations consist of multi-output SNNs. Every multi-output SNN consists of a single hidden layer and is expected to simulate all behavioural components as output. Each multi-output SNN contains 200 hidden layer neurons.

A total number of 4942 behavioural patterns were collected by evaluating randomly generated commands on the real-world Hexapod robot. From the patterns collected, 80% were used to train the SNNs, 10% formed the validation set and 10% formed the test dataset. The features and predicted outputs of all behavioural patterns are standardised to have a zero mean and unit standard deviation. The whole training dataset is used for gradient calculations and batch-based training methods are not used. Training is performed using the Adam optimisation algorithm. The SNNs are implemented using the Deep Learning Libraries called Keras [Chollet, 2015] and Tensorflow [Abadi *et al.*, 2015]. After each training iteration, the weights are saved whenever the validation error improves. SNNs are trained for 4000 iterations. Nodes in the dropout layer have a dropout rate of 50%. Dropout is enabled for all SNNs during training but only enabled during controller evolution for the **Dropout** Simulator Configuration. Over-fitting is avoided by saving the weights associated with the lowest validation error. The saved weights are used for the final SNN models.

4.3 SNS Experiments

During the ER process, controllers are evaluated in simulation. A simulator predicts the path of each controller. The simulated path is a list of positions reached by the robot in simulation. A fitness function (Algorithm 2) assigns controller fitness scores during the ER process. The fitness function takes as input the controller being assessed and the corresponding simulated path of the controller. The Euclidean distance between the simulated starting and final positions of the controller evaluation is calculated.

Simulator configurations not producing uncertainty information will have a standard deviation of zero and will not contribute any penalties. For each command and behavioural component simulated, the normalised standard deviation of the predictions is determined. The normalised standard deviation is multiplied by a penalty factor and added to an accumulation of penalties. The normalisation is based on the maximum standard deviations observed for predictions on the test dataset. Normalisation is necessary in order equally consider the uncertainty weight of each behavioural component. A controller's fitness score becomes the ratio of the final displacement of the robot and the accumulated penalties. The evolution process tries to minimise the penalties and maximise the distance travelled in Algorithm 2.

Algorithm 2 Hexapod Controller fitness evaluation

Require: *positions* ← Simulated path list and *commands* ← List of controller commands

```
distance ← Distance between start and end of positions
penalties ← 1
c ← Penalty factor
for each command in commands do
    for for each behavioural component of prediction do
        v ← Normalised standard deviation of the prediction
        penalties ← penalties + c × v
    end for
end for
fitness ← distance/penalties
return fitness
```

Parameter settings used during the ER process were established in prior work [Pretorius *et al.*, 2019] and are shown in Table 4.1. An initial population of 100 controllers is generated from a uniform distribution. Parents are selected through a tournament selection process using a tournament size of 10% of the controller population size. The crossover method used is Simulated Binary Crossover (SBX). The probability of mutating any particular joint angle change in a controller by a random amount is 10%. The size of a mutation is a random number between -6 and 6 degrees. The ER process is carried out for 1000 controller generations. The fittest controller in the final generation is selected as the solution. Thirty independent trial runs of the SNS approach (Methodology A) per unique adaptation are performed.

Controller Population Size	100
Initialization	Random from a uniform distribution
Selection	Tournament (Tournament size 10%)
Cross-over Method	Simulated Binary Crossover (SBX)
Mutation Rate	10%
Mutation Method	Random Component Perturbation
Controller Generation Limit	1000

Table 4.1: Parameters for controller evolution

Adaptations investigated consist of changes to the simulator configuration and the inclusion or exclusion of simulator noise. A total number of 10 unique adaptations are investigated (Table 4.2). Each adaptation is given an encoded name which is listed in the first column of Table 4.2. The first letter, "**H**", stands for Hexapod which indicates the robot platform. The second letter represents the simulator configuration. Letters **B**, **D**, **E**, **M** and **S** represent the **Basic**, **Dropout**, **Ensemble**, **Ensemble Multi-output** and **Basic Multi-output** simulator configurations, respectively. The third letter indicates if noise is injected into simulator predictions during the ER process (**E** indicates that simulator noise is excluded while **N** indicates that noise is present).

The procedure followed in each trial run of the SNS approach is as follows:

1. A set of adaptations is chosen for the experimental run.

2. The process described in Methodology A is performed (Figure 4.1).

3. The simulated and real-world paths of the solution controller is collected as observational data.

Experiment	Simulator Configuration	Simulator Noise
HBE	Basic	No
HBN	Basic	Yes
HDE	Dropout	No
HDN	Dropout	Yes
HEE	Ensemble	No
HEN	Ensemble	Yes
HME	Ensemble Multi-output	No
HMN	Ensemble Multi-output	Yes
HSE	Basic Multi-output	Yes
HSN	Basic Multi-output	No

Table 4.2: SNS Experimental Adaptations

Once all trial runs are complete, the performance distributions of the tested SNS adaptations are studied. The transferability and convergence properties of the tested adaptations are also investigated.

4.4 The SNS Experiment Results

An example solution controller is presented in Section 4.4.1. The behavioural dataset and SNN training is covered in Section 4.4.2. Once SNNs are trained, they can be used to evaluated candidate controllers during the ER process. For each proposed adaptation in Table 4.2, thirty solution controllers are independently produced.

Performance and transferability distributions of each tested adaptation are presented in Sections 4.4.3 and 4.4.4, respectively. The solution trajectories of solutions are highly

dependent on the adaptation used (Section 4.4.5). Finally, the real-world and simulated paths of the best performing solution to each adaptation is presented in Section 4.4.6.

4.4.1 Demonstration

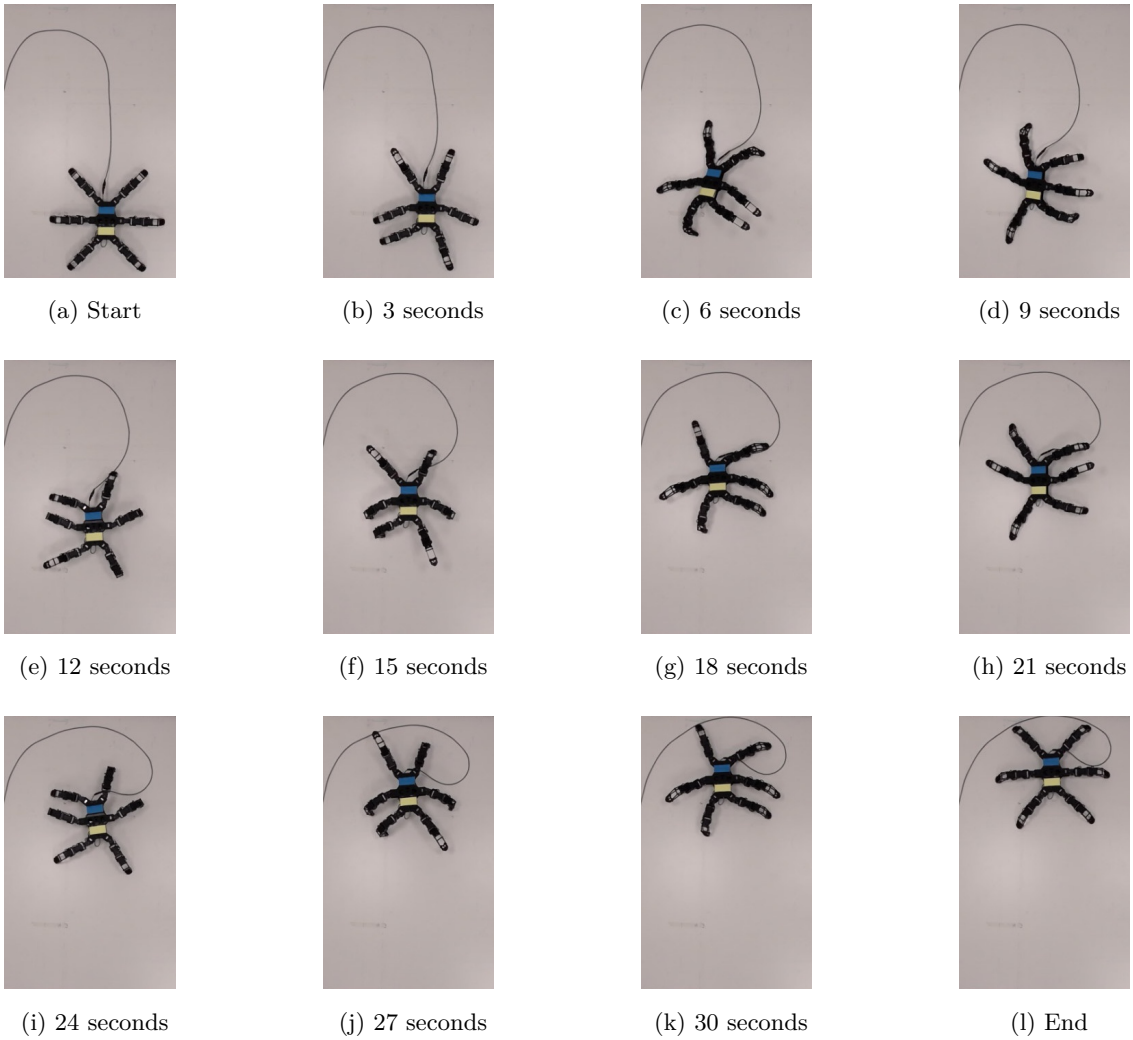
The SNS approach is successfully used to develop effective gaits for the Hexapod robot. The trained SNNs can accurately simulate Hexapod robot behaviours over many commands without real-time feedback. No sensors are simulated in any way. Effective solutions are developed without the simulator explicitly taking into account body orientations or leg contact times with the ground. The success rate for producing adequate solutions is high and the SNS approach can be considered a fairly robust method when applied to the Hexapod robot. This section demonstrates a single example solution produced by the SNS approach.

The chosen solution controller is demonstrated[1] in Figure 4.2. The robot successfully traverses a significant distance upwards. The robot's initial position before the first command is executed is shown in Figure 4.2a. During controller evaluation, the robot's position is captured at 3 second intervals. The final position of the robot upon completion of the controller evaluation is given in Figure 4.2l.

The simulated and real-world trajectories of the chosen solution controller is also illustrated in Figure 4.3. The robot is situated on the origin facing the positive y-direction before the evaluation begins. The solid line represents the real-world trajectory of the robot. Similarly, the dashed line illustrates the simulated trajectory.

The positions achieved by the robot (Figure 4.3) are annotated to approximately correspond to the sub-figures in Figure 4.2. The start and end positions are represented by the letters 'a' (Figure 4.2a) and 'l' (Figure 4.2l), respectively. The robot moves slightly upwards from the starting position, reaching 'b' (Figure 4.2b). Movement shifts upwards and to the left to reach the the position annotated 'c' (Figure 4.2c) and to the upper right towards 'e' (Figure 4.2e). Significant movement upwards towards 'g' (Figure 4.2g) is achieved, followed by a slight shift towards the position annotated 'i' (Figure 4.2i). The next 6 seconds entails the robot moving upwards to the position annotated as 'k' (Figure 4.2k), followed by the robot achieving its end position which is annotated as 'l' (Figure

[1] https://youtu.be/5z7JgWLb9rg



(a) Start (b) 3 seconds (c) 6 seconds (d) 9 seconds

(e) 12 seconds (f) 15 seconds (g) 18 seconds (h) 21 seconds

(i) 24 seconds (j) 27 seconds (k) 30 seconds (l) End

Figure 4.2: Solution controller demonstration

4.2l).

The movement strategy involves a subset of limbs reaching forwards while other limbs keep the robot's main body off the floor. After certain limbs have reached sufficiently far forwards, the limbs stop reaching forwards and instead stabilise the robot while other limbs reach forwards. Limbs alternate between support and reaching forward in a cyclical fashion without being designed to do so. The robot also slightly shifts left and right while moving forwards.

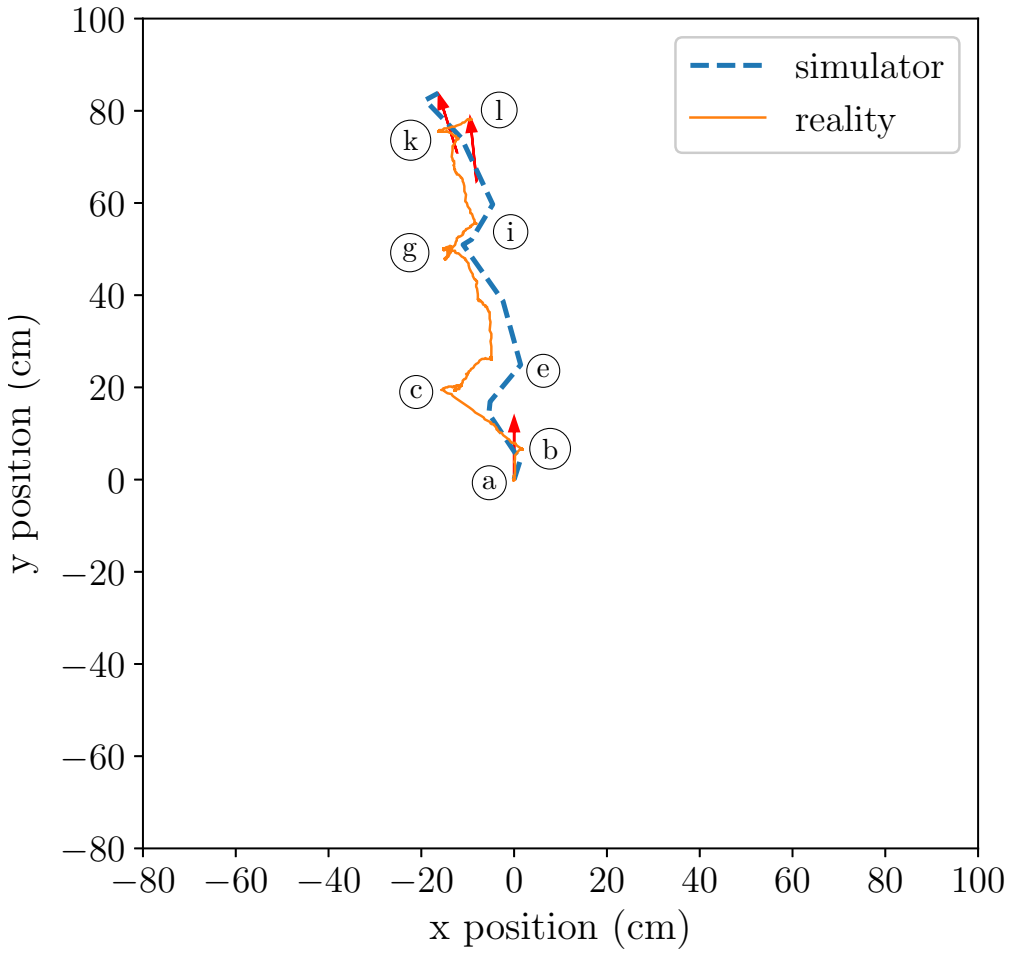


Figure 4.3: Simulated and real-world trajectory paths for solution controller

4.4.2 Static SNN Results

This section returns to the training of the SNNs based on the collected behavioural data, which is required before controllers can be evolved in the SNS approach. Section 4.4.2.1 presents an analysis of the collected behavioural data. Once the SNNs have been trained, the accuracy of the proposed simulator configurations are discussed in Section 4.4.2.2.

4.4.2.1 Behavioural Data

The Δx and Δy behavioural components for the training dataset are visualised as a scatter plot in Figure 4.4. Each point represents the final position of the robot after evaluating a single command. The x and y displacements represent the relative change in the robot's position due to the execution of the associated command. It is clear that most commands have little effect on the distance travelled by the robot. The observed displacements are not uniformly distributed but more closely resemble a Gaussian distribution.

Density plots comparing the robot's displacements and changes in heading for the training dataset are given in Figures 4.5 and 4.6, respectively. The interquartile range (IQR) for the x displacements are between -1.87 and 1.83 centimetres. The IQR for the y displacements are between -2.61 and 2.55 centimetres. This indicates that a large portion of the training data patterns have displacements relatively close to zero. Both the x and y displacements have a mean close to zero. The standard deviation for the x and y displacements are 3.41 and 4.47 centimetres, respectively.

A Levene's statistical test for assessing equality of variance is used to compare the variances between the x and y displacement distributions. The variance for the y displacements is significantly greater than that of the x displacements with a p-value of 5.7×10^{-50}. Physically this indicates that the Hexapod robot has greater mobility moving forwards and backwards relative to the robot's heading. The distribution for the heading displacement is Gaussian with a mean close to zero. The standard deviation of the heading displacements is 10.24 degrees. The IQR is between -7.00 and 6.69 degrees.

4.4.2.2 Training Errors

The MSE and IQR per simulated behavioural component and simulator configuration tested for the training and test datasets are given in Tables 4.3 and 4.4, respectively. The first and third quartiles of squared errors are represented by $\mathbf{Q}_1$ and $\mathbf{Q}_3$, respectively.

The lowest Δx training MSE values are achieved by the **Basic** and **Dropout** simulator configurations. This is followed closely by the **Ensemble** simulator configuration. For the training dataset, the MSE values for the Δx **Ensemble Multi-output** and **Basic Multi-output** simulator configurations have noticeably higher MSEs compared to the

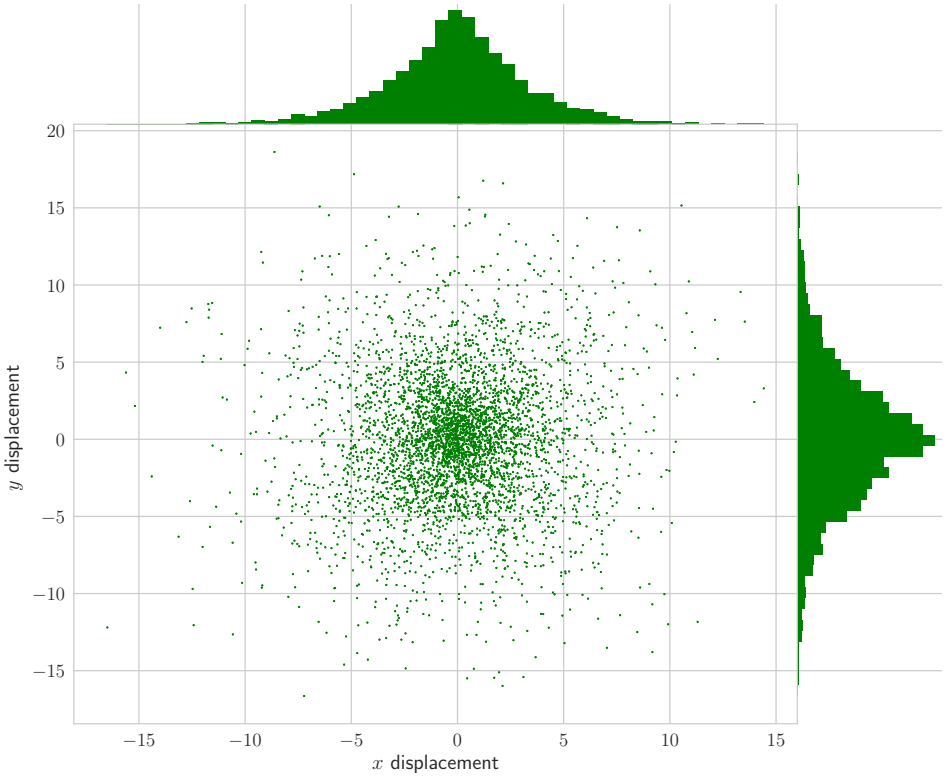


Figure 4.4: Scatter plot for the training data Δx and Δy components

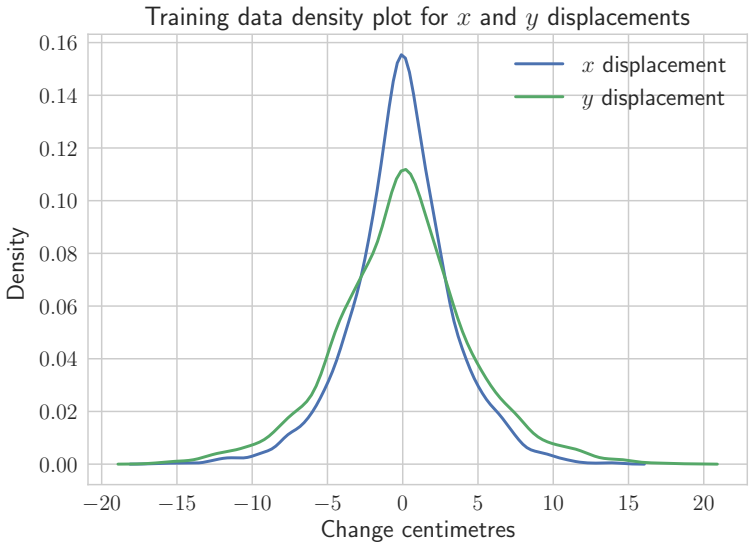


Figure 4.5: Density plot for the training data Δx and Δy components

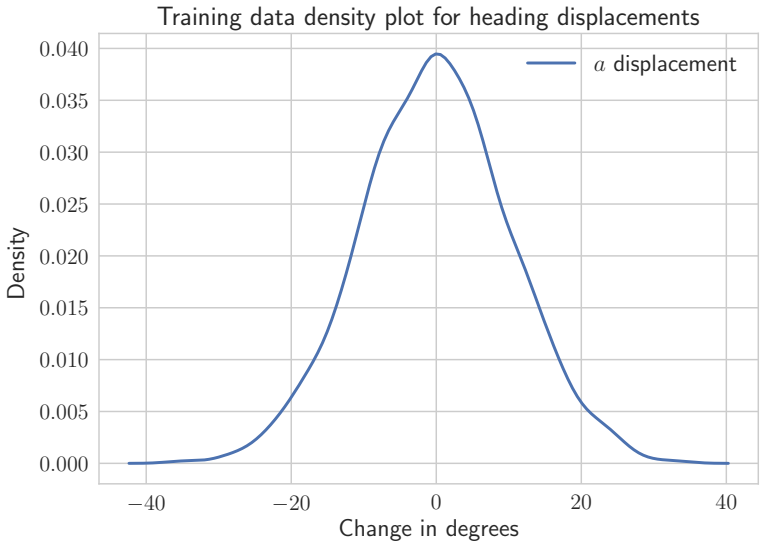


Figure 4.6: Density plot of the training data Δa component

other configurations.

The lowest training dataset MSE for the Δy component is the **Ensemble** simulator configuration followed closely by the **Basic** and **Dropout** simulator configurations. The **Basic**, **Dropout** and **Ensemble** simulator configuration have Δy MSEs within 4% of each other. The **Ensemble Multi-output** and **Basic Multi-output** simulator configurations have a more than 24% higher Δy training dataset MSE compared to other configurations.

The Δa training dataset MSE values for all configurations are within 7% of each other. The **Ensemble** simulator configuration has the lowest MSE and IQR compared to all other configurations. The **Basic Multi-output** simulator configuration has the highest MSE and IQR followed by the **Ensemble Multi-output**, **Dropout** and **Basic** simulator configurations.

The squared errors observed on the test dataset are a more realistic indicator of predictive accuracy compared to the training dataset. Differences in MSE values between simulator configurations are less extreme for the test dataset. The MSE values are within 8%, 3.2% and 1% of each other for the Δx, Δy and Δa behavioural components, respectively. The **Ensemble Multi-output** and **Basic Multi-output** simulator configurations have lower Δx and Δy MSE values compared to the **Basic** and **Dropout** simulator con-

Training Dataset	Δx			Δy			Δa		
	MSE	$\mathbf{Q_1}$	$\mathbf{Q_3}$	**MSE**	$\mathbf{Q_1}$	$\mathbf{Q_3}$	**MSE**	$\mathbf{Q_1}$	$\mathbf{Q_3}$
Basic	*3.93*	0.26	4.30	6.27	0.47	7.08	33.42	2.72	37.15
Dropout	3.97	0.28	4.37	6.33	0.47	7.07	33.54	2.68	37.30
Ensemble	4.27	0.29	4.83	*6.12*	0.46	6.65	*32.18*	2.49	35.80
Ensemble Mulit-output	5.26	0.37	6.04	7.85	0.54	9.03	33.63	2.73	37.96
Basic Multi-output	5.44	0.39	6.18	8.05	0.56	9.32	34.44	2.95	38.13

Table 4.3: Training dataset MSE and IQR for each simulator configuration

figurations.

The **Ensemble** simulator configuration has the lowest Δx test dataset MSE but the **Ensemble Multi-output** configuration has the lowest IQR. The **Dropout** and **Basic** simulator configurations have the highest MSE and IQR for the Δx test dataset.

The MSE values for the test dataset Δy component are lowest for the **Ensemble Multi-output** and **Basic Multi-output** simulator configurations followed closely by the **Basic** and **Dropout** simulator configurations. The **Ensemble Multi-output** simulator configuration has the lowest Δy test dataset IQR. The **Ensemble** simulator configuration has the highest MSE and IQR.

The **Ensemble Multi-output** simulator configuration has marginally the lowest test dataset Δa MSE while the **Basic Multi-output** and **Dropout** simulator configurations have the highest MSE values. The MSE values for the test dataset Δa behavioural component are all within 1% of each other.

Multi-output SNNs modelling all behavioural components have a smaller gap between the training and test MSE values. The multi-output SNNs are likely less capable of overfitting the training dataset in comparison to single output SNN configurations.

The best simulator configuration should be judged based on the test dataset MSE distributions. Simulator configurations modelling the Δx behavioural component have the largest differences between test dataset MSE distributions. The **Ensemble Multi-output** simulator configuration is the best choice for the Δx and Δy behavioural components. Simulator configurations modelling the Δa behavioural component are relatively more similar to each other. There is no clear best simulator configuration choice for

Test Dataset	Δx			Δy			Δa		
	MSE	$\mathbf{Q}_1$	$\mathbf{Q}_3$	**MSE**	$\mathbf{Q}_1$	$\mathbf{Q}_3$	**MSE**	$\mathbf{Q}_1$	$\mathbf{Q}_3$
Basic	7.45	0.64	8.57	10.77	0.77	12.42	47.86	4.02	53.64
Dropout	7.65	0.69	8.76	10.78	0.76	12.39	48.34	4.30	52.05
Ensemble	*7.09*	0.55	7.95	11.02	0.74	12.87	47.87	4.44	51.43
Ensemble Mulit-output	7.18	0.55	7.78	10.71	0.69	12.39	*47.82*	4.28	51.68
Basic Multi-output	7.23	0.57	7.93	*10.68*	0.76	12.56	48.20	4.02	50.23

Table 4.4: Test dataset MSE and IQR for each simulator configuration

modelling the Δa behavioural component.

4.4.3 Performance

Thirty independent trial runs are conducted for each SNS adaptation tested, producing 30 controller solutions per SNS adaptation (Table B.1). The solutions controllers are evaluated on the real-world Hexapod robot and the resulting paths generated are recorded. The performance metric calculation is given in equation (3.1).

A summary of the performance statistics for adaptations is given in Table 4.5. The performance distributions are illustrated as standard box-and-whisker plots in Figure 4.7. Not adding noise to the simulator during the ER process always performs significantly better than including noise. Adaptations without simulator noise are observed to have higher standard deviations compared to their noiseless counterparts. This could be due to simulator noise significantly decreasing the performance which leads to lower possible standard deviations.

For all box-and-whisker plots in this thesis, the box plot rectangle spans the IQR with the segment inside the IQR box representing the median. The whiskers outside the IQR box represent values 1.5 times the IQR above third quartile and below the first quartile. Whiskers extend up to the last datum within the 1.5 times IQR extensions. Values outside the whisker range are considered outliers.

All statistical comparisons use a significance level of 5%. The Kruskal-Wallis H test is used to determine whether the performances of the tested adaptations originate from the same distribution. The performance distributions between the adaptations were found

Approach	Mean (cm)	Median (cm)	Q_1 (cm)	Q_3 (cm)	Std. Dev. (cm)
HBE	53.8	53.6	47.5	60.5	10.0
HBN	41.7	40.8	37.1	48.6	9.1
HDE	52.1	52.9	44.8	59.1	12.1
HDN	39.9	39.7	33.4	46.1	10.8
HEE	53.4	53.1	48.3	60.7	10.4
HEN	39.0	38.0	33.4	45.8	9.3
HME	*60.2*	*61.1*	54.4	66.8	8.7
HMN	40.3	41.0	33.8	47.3	8.0
HSE	55.7	55.0	50.9	60.0	8.6
HSN	41.1	41.2	34.9	47.6	7.6

Table 4.5: Performance statistics for the SNS adaptations

to be significantly different with a p-value of 1.15×10^{-22}. This indicates that there are significant differences between the performance distributions of the tested adaptations. A post hoc analysis is performed using pairwise Mann-Whitney U (also called the Mann-Whitney-Wilcoxon) tests to determine which adaptations are significantly different from each other. The Mann-Whitney U test is a non-parametric test and does not assume normally distributed datasets. This test is also a more conservative comparison of differences between distributions than other statistical tests. The p-values for the post hoc analysis are shown in Table 4.6. Rows and columns represent the adaptations listed in Table 4.2.

No significant differences in performance are found between adaptations that add noise to the simulator (Adaptations HBN, HDN, HEN, HMN and HSN). Adaptations with simulator noise perform significantly worse compared to all noiseless adaptations. Not adding simulator noise results in better likely performance outcomes for solutions. The only adaptation that performed significantly better than all others excludes simulator noise and uses an **Ensemble Multi-output** simulator configuration (HME). The combined effect of using ensembles with multi-output SNNs improves performance outcomes significantly more than if only the **Ensemble** (HEE) or a **Basic Multi-output** (HSE) simulator configuration is used alone.

The time taken to complete a single trial run of the SNS approach is highly dependent

	HBE	**HBN**	**HDE**	**HDN**	**HEE**	**HEN**	**HME**	**HMN**	**HSE**
HBN	1.0×10^{-5}	-	-	-	-	-	-	-	-
HDE	0.64377	0.00016	-	-	-	-	-	-	-
HDN	4.5×10^{-6}	0.40629	6.7×10^{-5}	-	-	-	-	-	-
HEE	0.83150	2.2×10^{-5}	0.78601	1.6×10^{-5}	-	-	-	-	-
HEN	6.2×10^{-7}	0.16351	8.7×10^{-6}	0.70819	9.0×10^{-7}	-	-	-	-
HME	0.00765	2.0×10^{-10}	0.007 30	4.1×10^{-10}	0.00697	9.1×10^{-12}	-	-	-
HMN	5.1×10^{-7}	0.53250	1.2×10^{-5}	0.91237	1.3×10^{-6}	0.55200	2.5×10^{-12}	-	-
HSE	0.58188	1.3×10^{-7}	0.24781	2.6×10^{-7}	0.47614	4.0×10^{-9}	0.03194	6.3×10^{-10}	-
HSN	1.6×10^{-6}	0.74123	5.1×10^{-5}	0.63327	2.4×10^{-6}	0.31362	5.3×10^{-12}	0.75235	2.4×10^{-9}

Table 4.6: Comparisons between the performances of adaptations for the SNS approach

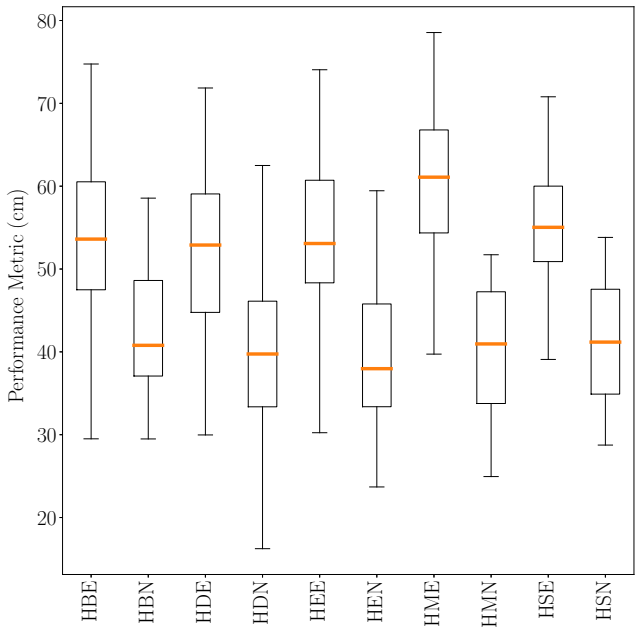


Figure 4.7: Performance distributions for SNS adaptations

on the simulator configuration used. Controller evaluations are slower on more complex simulator configurations. The estimated runtime for executing a single trial run of the SNS approach for adaptations using the tested simulator configurations are given in Table 4.7. Durations are estimated using a mid-range desktop computer without using a Graphics Processing Unit (GPU). The shortest runtime durations are seen for trial runs consisting of **Basic Multi-output** and **Basic** simulator configurations. SNS trial runs using the **Basic** and **Basic Multi-output** simulator configurations take approximately 2 and 1.5 minutes to complete, respectively. The **Ensemble Multi-output** simulator configuration takes approximately 3 minutes to complete. The **Ensemble** and **Dropout** simulator configurations take the longest to complete at 15 and 8 minutes, respectively.

4.4.4 Transferability

A summary of the transferability statistics for adaptations is given in Table 4.8. Transferability values for trial runs are given in Table B.2. The transferability between the

	Time (minutes)
Basic	2
Dropout	8
Ensemble	15
Ensemble Multi-output	3
Basic Multi-output	1.5

Table 4.7: SNS trial run durations

various adaptations of the SNS approach are illustrated in Figure 4.8. Lower values indicate better transferability and a closer correspondence between simulation and reality. The Kruskal-Wallis H test is used to determine whether the transferability distributions of each tested adaptation originates from the same distribution. The transferability distributions between adaptations were found to be significantly different from each other. The p-value for the Kruskal-Wallis H test is 3.41×10^{-16}. A post hoc analysis is conducted using Mann-Whitney U pairwise comparisons. The p-values for the post hoc analysis are shown in Table 4.9.

Adaptations with simulator noise do not generally have significantly better transferability distributions compared to adaptations without simulator noise. The only adaptations that have significantly better transferability distributions with simulator noise use either the **Ensemble Multi-output** or **Basic Multi-output** simulator configuration (Adaptations HMN and HSN). However, the overall results indicate that simulator noise has little effect on the likely transferability of solutions.

Adaptations HBE, HBN, HDE, HDN, HEE and HEN have significantly worse transferability distributions compared to adaptation using multi-output SNNs (HME, HMN, HSE and HSN). For configurations using multi-output SNNs, the transferability distribution for the HSE adaptation is significantly worse compared to the HME, HMN and HSN adaptations.

The most significant contributor towards better transferability is the use of multi-output SNNs. The transferability distribution upper quartiles for adaptations HME, HMN, HSE and HSN are close to the lower quartile values of the single-output SNN

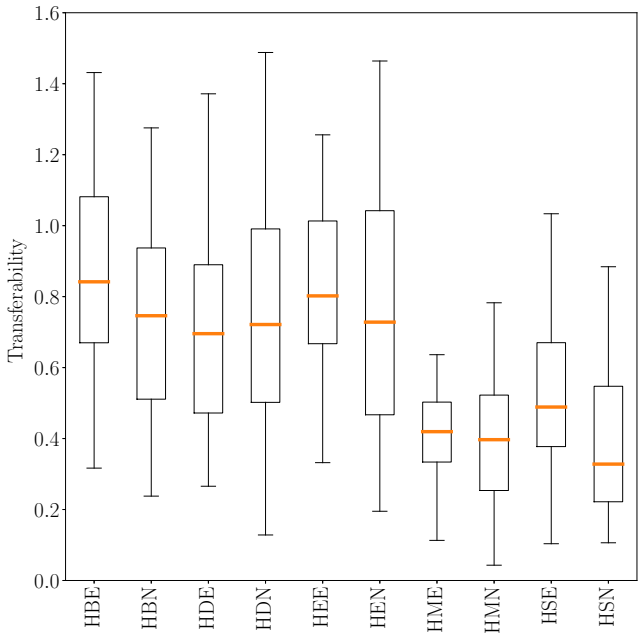


Figure 4.8: Transferability distributions for the SNS adaptations

configurations. The standard deviations for the multi-output SNN configurations are also relatively low. The HME, HMN and HSN adaptations, statistically have similar transferability distributions. The HME adaptation has both the best in performance distribution and one of the best transferability distributions.

Section 4.4.5 examines the behavioural trends of solution controllers in simulation and reality for each adaptation tested. Some of the trends observed indicate why certain adaptations have significantly better transferability distributions than others. The simulators for adaptations HBE, HBN, HDE, HDN, HEE and HEN visibly overestimate the real-world distances travelled. This result in their being a large error between the simulated and real-world final positions of solutions, leading to a high transferability values.

Simulators for adaptations HME, HMN, HSE and HSN tend not to overestimate the real-world distances travelled. The **Basic Multi-output** simulator configuration with noise (HSN) has the best overall transferability, followed closely by the **Ensemble Multi-output** simulator configuration with and without noise (HMN, HME). The **Basic Multi-output** simulator configuration without noise (HSE) demonstrated worse transferability

Approach	Mean	Median	Q_1	Q_3	Std. Dev.
HBE	0.92	0.84	0.67	1.08	0.38
HBN	0.79	0.75	0.51	0.94	0.49
HDE	0.84	0.70	0.47	0.89	0.55
HDN	0.84	0.72	0.50	0.99	0.54
HEE	0.87	0.80	0.67	1.01	0.37
HEN	0.78	0.73	0.47	1.04	0.37
HME	0.42	0.42	0.33	0.50	0.18
HMN	0.41	0.40	0.25	0.52	0.25
HSE	0.52	0.49	0.38	0.67	0.20
HSN	0.39	0.33	0.22	0.55	0.22

Table 4.8: Transferability statistics for the SNS adaptations

compared to the HSN, HMN and HME adaptations. The HSE adaptation has a significantly better transferability compared to adaptations using a **Basic** (HBE, HBN), Ensemble (HEE, HEN) or **Dropout** (HDE, HDN) simulator configuration.

4.4.5 Simulated and Real-world Trajectories

The simulated and real-world paths generated by solution controllers for adaptations are visualised and discussed in this section. Figures 4.9 and 4.10 illustrate the solution paths obtained for the **Basic** and **Basic Multi-output** simulator configurations. Similarly, solution paths for the **Ensemble**, **Dropout** and **Ensemble Multi-output** simulator configurations are shown in Figures 4.11 and 4.12. The sub-figures on the left illustrate the simulated paths. The corresponding real-world paths are shown by the sub-figures on the right. Having the simulated and real-world paths side-by-side helps visualise differences between simulation and reality.

The distances travelled by controllers in simulation tend to be relatively similar within each adaptation. Simulators tend to overestimate the distances travelled. The addition of simulator noise noticeably reduces the simulated distances travelled. Noise likely complicates the controller search space due to random changes in the fitness landscape, making

	HBE	**HBN**	**HDE**	**HDN**	**HEE**	**HEN**	**HME**	**HMN**	**HSE**
HBN	0.0965	-	-	-	-	-	-	-	-
HDE	0.1342	0.9707	-	-	-	-	-	-	-
HDN	0.2188	0.8660	0.7191	-	-	-	-	-	-
HEE	0.4853	0.3208	0.3280	0.5133	-	-	-	-	-
HEN	0.2132	0.8545	0.9474	0.9474	0.4147	-	-	-	-
HME	4.6×10^{-9}	1.3×10^{-5}	1.5×10^{-5}	2.8×10^{-5}	3.1×10^{-9}	4.1×10^{-5}	-	-	-
HMN	3.4×10^{-8}	2.4×10^{-5}	2.6×10^{-5}	4.7×10^{-5}	6.6×10^{-8}	7.7×10^{-5}	0.5920	-	-
HSE	2.7×10^{-6}	0.0021	0.0043	0.0041	6.8×10^{-6}	0.0067	0.0479	0.0385	-
HSN	5.9×10^{-9}	8.7×10^{-6}	6.8×10^{-6}	1.1×10^{-5}	1.2×10^{-8}	1.5×10^{-5}	0.4063	0.9007	0.0234

Table 4.9: Comparison between transferability distributions of adaptations for the SNS approach

the exploitation of known good solutions more difficult.

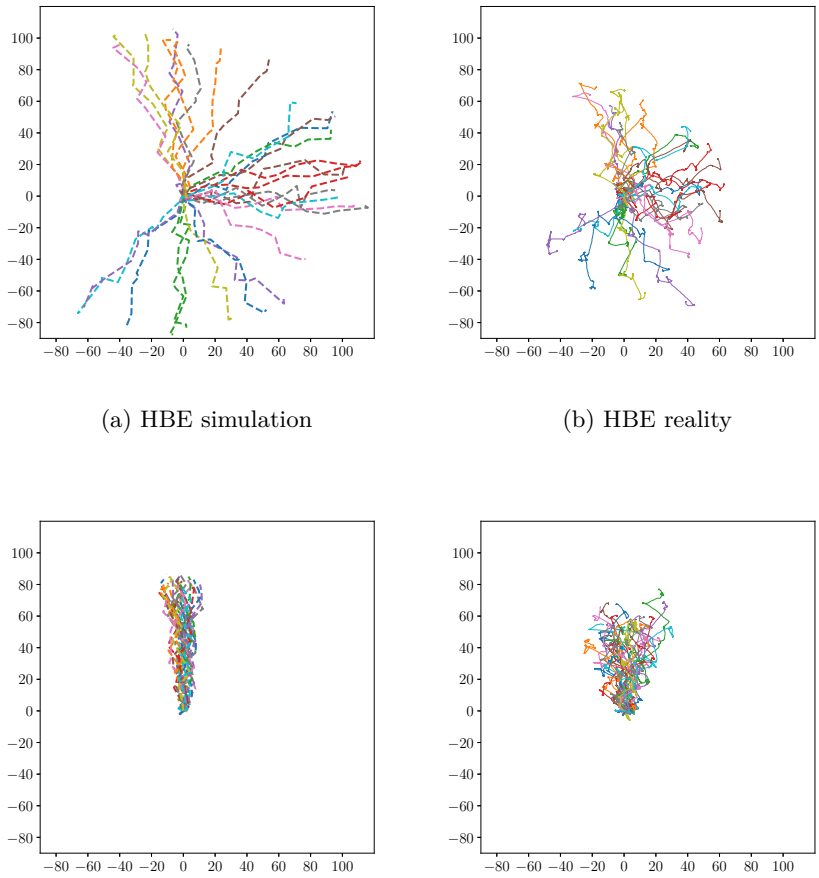
A significant difference in evolved behaviours is observed between the **Basic** and **Basic Multi-output** simulator configurations. Solution paths for the HBE and HBN adaptations have diverse behaviours moving in many directions. However, the HSE and HSN adaptations have little diversity in terms of the direction of trajectories. Adaptations HEE, HEN, HME and HMN generally have paths moving in an upward trajectory.

The **Dropout** simulator configuration demonstrates diverse sets of behaviours. Adaptations using the **Basic** and **Dropout** simulators configurations are relatively similar in terms of having highly diverse sets of solutions. However, the **Dropout** simulator configurations appear to be slightly more grouped together. Adaptations using the **Ensemble** and **Ensemble Multi-output** simulator configurations result in controllers that are significantly similar to each other.

The SNS adaptations can be studied in terms of the diversity of the controller solutions produced and the general simulated and real-world distances travelled. Figure 4.13 illustrates where each adaptation falls on the diversity and distance travelled spectrum. The magnitude of the distance represents the simulated and real-world distanced travelled. Two trends are observed in terms of the general heading of solution paths. Controller solutions either conform to an upward trajectory or solutions tend to move in diverse directions.

All SNS adaptations that include simulator noise tend to produce controller solutions that travel shorter distances compared to adaptations that include simulator noise. Not including simulator noise is beneficial in that there is an increased likelihood of evolving controllers that cover greater distances. Noise slightly increases diversity of solutions when SNNs are configured in an ensemble configuration (HEN, HMN).

Adaptations using an ensemble approach are likely to produce solutions that have trajectories with a forward direction. The **Ensemble** simulator configuration without noise (HEE) produces a slightly less diverse set of solutions compared to the same simulator configuration with noise. The HEE adaptation also has paths all moving in the forward direction while the HEN adaptation has five solution paths not having in a forward trajectory. All adaptations using multi-output SNNs are biased towards producing forward moving solutions.



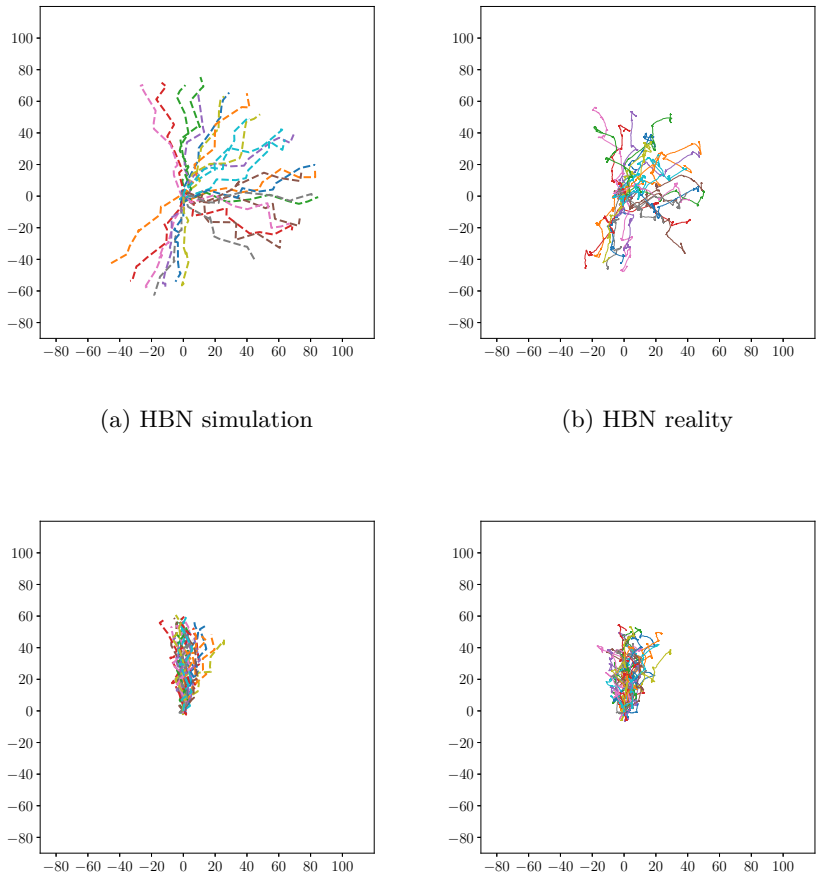
(a) HBE simulation

(b) HBE reality

(c) HSE simulation

(d) HSE reality

Figure 4.9: Solution paths for the Basic (HBE) and Basic Multi-output (HSE) simulator configurations without noise



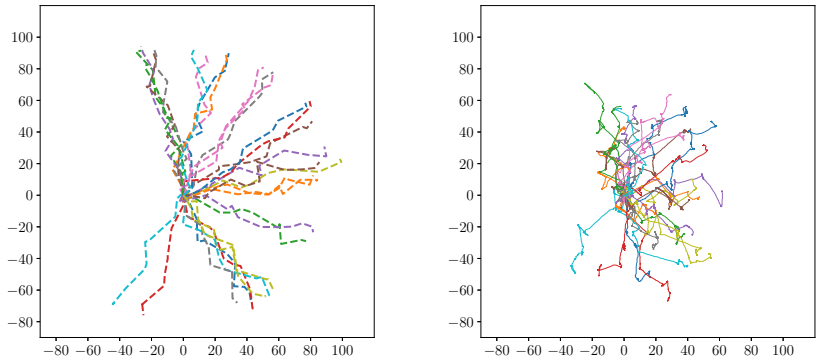
(a) HBN simulation

(b) HBN reality

(c) HSN simulation

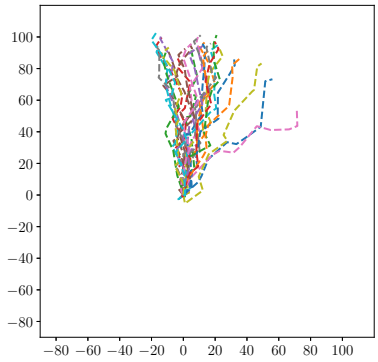
(d) HSN reality

Figure 4.10: Solution paths for the Basic (HBN) and Basic Multi-output (HSN) simulator configurations with noise

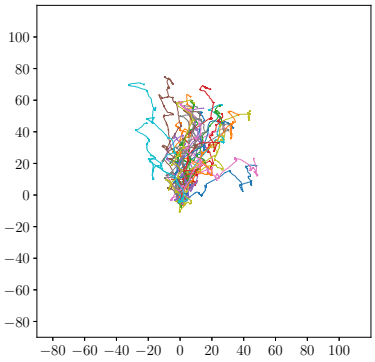


(a) HDE simulation

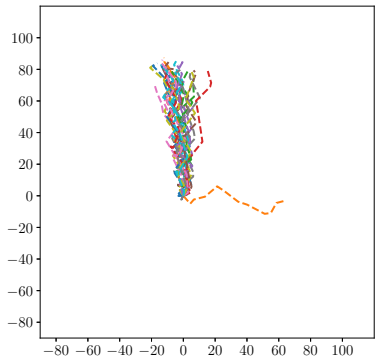
(b) HDE reality



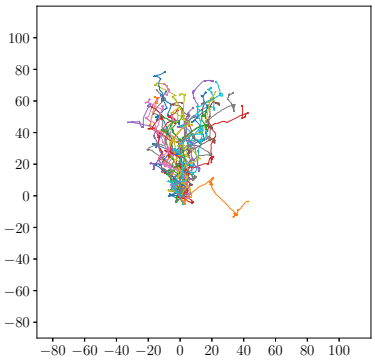
(c) HEE simulation



(d) HEE reality

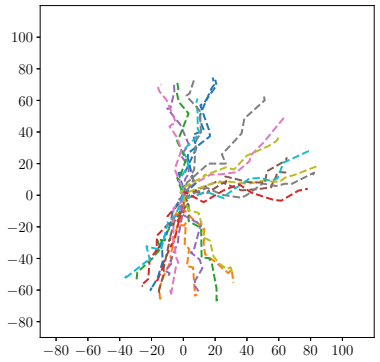


(e) HME simulation

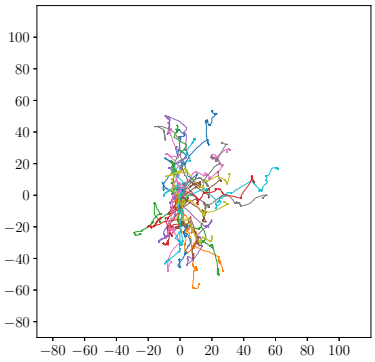


(f) HME reality

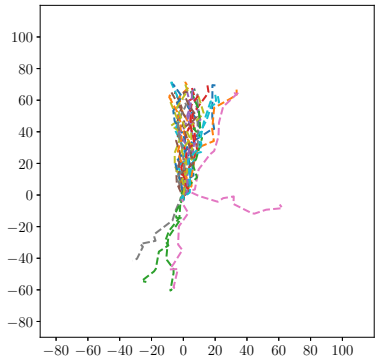
Figure 4.11: Solution paths for Dropout (HDE), Ensemble (HEE) and Ensemble Multi-output (HME) configurations without noise



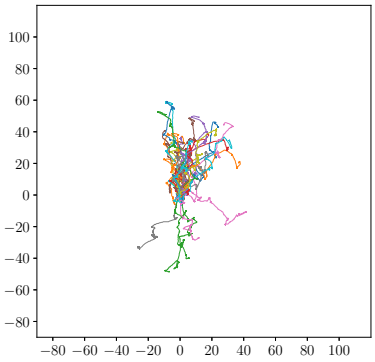
(a) HDN simulation



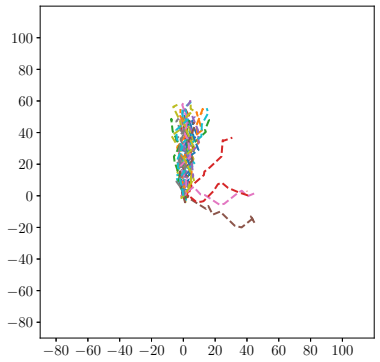
(b) HDN reality



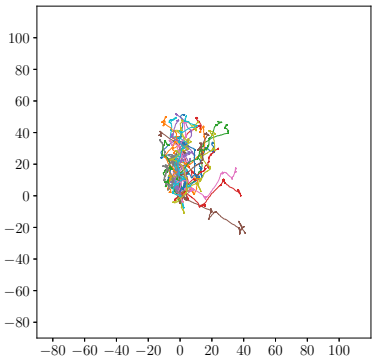
(c) HEN simulation



(d) HEN reality



(e) HMN simulation



(f) HMN reality

Figure 4.12: Solution paths for Dropout (HDN), Ensemble (HEN) and Ensemble Multi-output (HMN) configurations with noise

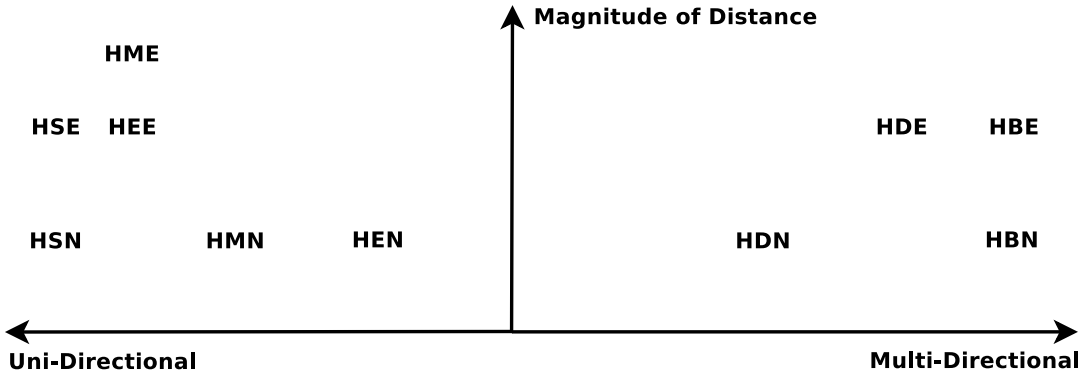


Figure 4.13: Solution diversity and magnitude trends between SNS adaptations

The **Basic Multi-output** simulator configurations with and without noise always produces solutions with trajectories that move in the forward trajectory. The **Ensemble Multi-output** simulator configuration without noise only consisted of one controller not moving in a forward trajectory.

4.4.6 Best Controllers

The best controller for each SNS adaptation is discussed in this section. The best performing controllers simulated and real-world trajectories are illustrated in Figures 4.14 and 4.15. Figures have the same dimensions in order to easily identify differences. The dashed lines represent the simulated trajectories of the robot for their respective controller evaluations. The solid lines represent the trajectories generated by the real-world robot for the specified controllers evaluated. The x and y axis is measured in centimetres. The starting position of the robot is located at the origin with the robot facing the positive y direction. The red arrow at the origin represents the initial heading of the robot. The final heading of the robot in simulation and reality is represented by the red arrows at the end of the simulated and real-world trajectories.

Behavioural metrics for the best controller in each adaptation is presented in Table 4.10. The performance and transferability columns are defined as in previous sections. The position error is calculated at the Euclidean distance between a controller's evaluated final positions in simulation and reality. The actual distance is measured as the Euclidean distance between the starting and final positions of the robot's trajectory in reality. Simi-

larly, simulated distance is the Euclidean distance between the starting and final positions of the robot in simulation. The heading error is the difference between the robot's final heading in simulation and reality in degrees.

All the best controllers for adaptations with simulator noise have performance values less than or equal to 62.5 centimetres. Top controllers evolved from adaptations without simulator noise have performance values greater than or equal to 71.9 centimetres. The inclusion of simulator noise visibly reduces the distance travelled by the best controller in each SNS adaptation.

For the best controller in each adaptation, simulated and real-world trajectories are a relatively close match. All the best controllers transfer well from simulation to reality. The simulator tends to overestimate the actual distances travelled for the best controllers in the HBN, HDE, HDN and HEE adaptations. However, behaviours observed between the simulated and real-world controller evaluations correspond relatively well even if there is a slight mismatch between the simulated and real-world distances travelled.

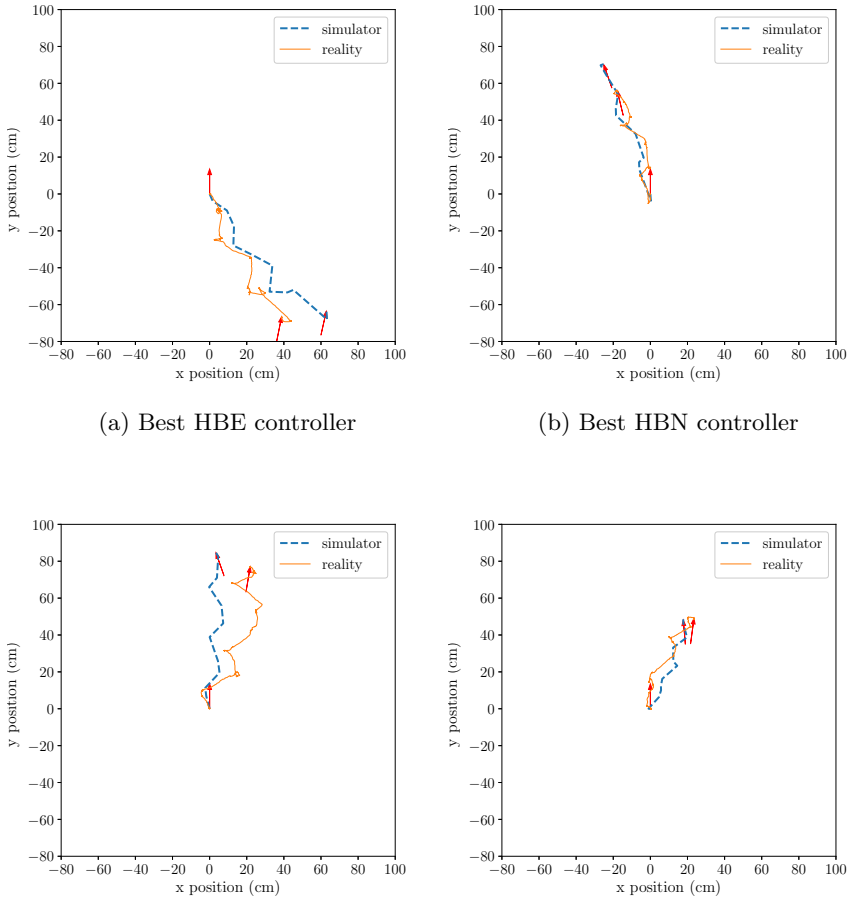
The best controllers for adaptations with multi-output SNNs (HME, HMN, HSE, HSN) are accurately simulated. Differences between the simulated and actual distances travelled is less than 6.7 centimetres for the best controllers evolved using adaptations with multi-output SNNs. Similarly, the best controllers developed using adaptations with single output SNNs (HBE, HBN, HDE, HDN, HEE, HEN) have simulated and actual distance differences greater than 11.1 centimetres.

Studying only the simulated and real-world final trajectory positions of evaluated controllers does not take into account the robot's simulated and real-world headings. The simulated and real-world final position headings are represented by the red arrows in Figures 4.14 and 4.15. The difference between the final real-world and simulated headings is relatively low for most of the best controllers. Only the best controllers for the HMN and HSE adaptations have heading errors greater than 13 degrees. The angle between the final simulated and real-world headings for the best controllers of the HME and HSE adaptations is 24.8 and 28.8 degrees, respectively.

Most of the best controller solutions move a significant distance in the positive y direction. The best solutions for the HBN, HEE, HEN, HME, HMN, HSE and HSN adaptations have trajectories moving mostly in the positive y direction. Seven of the

Adaptation	Performance (cm)	Transferability	Position Error (cm)	Actual Distance (cm)	Simulated Distance (cm)	Heading Error (degrees)
HBE	74.7	0.32	24.5	77.3	89.2	1.3
HBN	58.6	0.28	16.4	58.6	74.8	5.6
HDE	71.9	0.35	24.9	72.0	96.4	6.4
HDN	62.5	0.35	22.0	62.7	84.0	10.2
HEE	74.1	0.33	25.0	75.2	95.4	4.8
HEN	59.4	0.20	11.6	59.5	70.7	4.0
HME	*78.5*	*0.11*	*8.9*	*78.8*	*85.4*	*11.7*
HMN	51.7	0.27	14.7	53.8	51.8	24.8
HSE	77.5	0.25	20.3	79.8	85.0	28.8
HSN	53.8	0.11	5.7	54.1	51.6	12.4

Table 4.10: Behavioural metrics of the best controller in each adaptation



(a) Best HBE controller

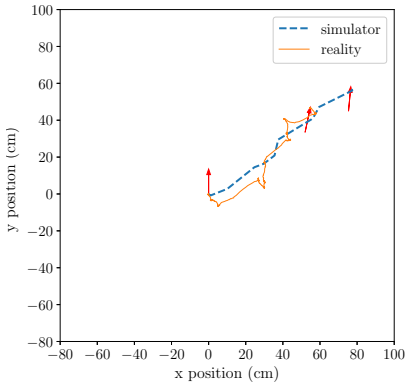
(b) Best HBN controller

(c) Best HSE controller

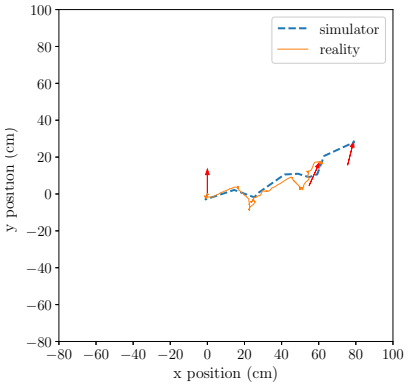
(d) Best HSN controller

Figure 4.14: Best performing solutions for the Basic and Basic Multi-output simulator configurations

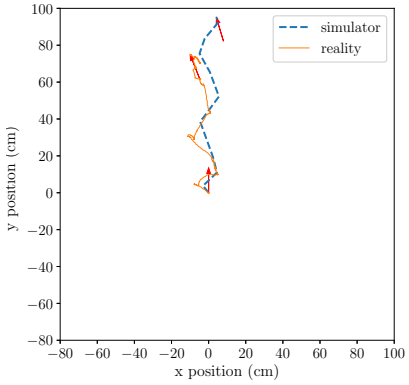
ten best controllers travel in the positive y direction. The only best controller to travel towards the negative y direction is evolved from the HBE adaptation. Trajectories moving a significant distance in the positive x direction are from the HDE and HDN adaptations.



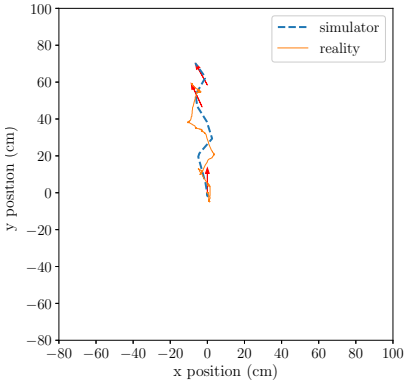
(a) Best HDE controller



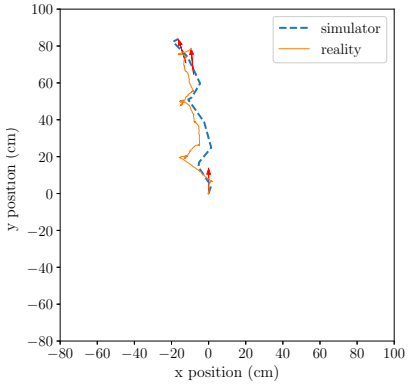
(b) Best HDN controller



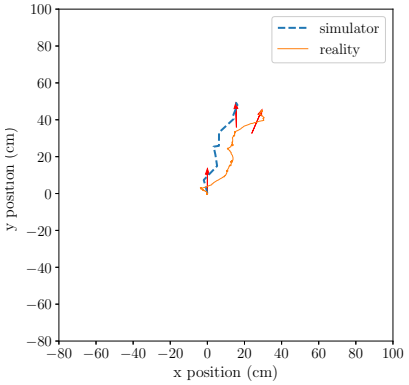
(c) Best HEE controller



(d) Best HEN controller



(e) Best HME controller



(f) Best HMN controller

Figure 4.15: Best performing solutions for the Dropout, Ensemble and Ensemble Multi-output configurations

4.5 Conclusion

The chapter successfully validates that the SNS approach can develop effective gait controllers for a Hexapod robot. Controllers evolved in simulation transfer well into reality. The controller design used is high dimensional, low-level and provides sufficient freedom for a robot to exhibit arbitrary behaviours. The simulator design models low-level commands which results in the simulator being compatible with any controller design.

Different simulator configurations are trained to model Hexapod robot behaviours. Many independent ER runs are conducted using different simulator configurations with and without simulator noise. The trajectories followed by solution controllers in simulation and reality are analysed. Solution controllers are grouped according to their adaptations and are analysed in terms of performance, transferability and general behaviours.

Prior research investigating the SNS approach has largely focused on simple robots or controllers. This work pioneers the way towards SNS approaches that try to compensate for the inaccuracies in simulating high dimensional robot behaviours. The prior work established a guideline that multi-output SNNs perform worse compared to using a separate SNN for each behavioural component. The results presented here strongly indicate that multi-output SNNs can at least for the specified problem outperform single-output SNNs.

The **Ensemble Multi-output** simulator configuration without noise performs significantly better in terms of the distance travelled in the direction of the simulated path compared to all other adaptations tested. The factors contributing towards improved performance outcomes is a combination of an ensemble approach, uncertainty penalisation and the use of multi-output SNNs without simulator noise. Not adding simulator noise to controller evaluations significantly improved the likely performance outcomes of solutions while not adversely affecting transferability. Simulator configurations with multi-output SNNs demonstrated particularly good transferability properties.

The diversity in observed trajectories for each adaptation differed significantly. The **Basic** and **Dropout** simulator configurations increase solution diversity while other simulator configurations produced mostly similar solutions. The observed differences in diversity might be due to the smoothness in the simulator search space. Multi-output SNNs and ensembles likely smooth out the controller search space which biases certain behaviours.

The solution search space likely changes significantly between certain simulator configurations.

Choosing the ideal adaptation can depend on certain goals. If highly diverse solution controllers are required then the **Basic** simulator configuration without noise should be used. The **Basic Multi-output** simulator configuration without noise is the best choice if computational efficiency is of primary importance. If discovering the best performing solution controller is the most important goal, the **Ensemble Multi-output** simulator configuration without noise is the best adaptation. For the problem investigated, results confirm the findings of prior research that not including simulator noise significantly improves performance outcomes for the specified gait optimisation problem.

The SNS approach does have significant disadvantages. The data collection scheme is time consuming and explores many unnecessary behaviours. Smarter data collection methods should be explored. The SNS approach creates static simulators. If the robot or environment changes significantly, the trained simulator might become inaccurate and unusable. The next chapter investigates the BNS approach which is designed to remedy some disadvantages inherent in the SNS approach.

Chapter 5

HEXAPOD BOOTSTRAPPED NEURO-SIMULATION

5.1 Introduction

The BNS approach is yet to be investigated on a limbed, walking robot such as the Hexapod robot. For the BNS approach, a lengthy data collection phase can be avoided. Little specialised knowledge would be required and the ER process would begin sooner than if a SNS or physics-based approach were used.

Details of the experimental procedures followed in this chapter are discussed in Section 5.2. The experimental work is split into two phases. The initial phase benchmarks a large number of adaptations completely in simulation without a real-world robot (Methodology B). Benchmarking a large number of adaptations is not feasible in reality. This simulated benchmarking phase is simply used to identify promising adaptations.

In the second phase (Methodology C), top performing adaptations observed during the first phase are selected and validated on the real-world Hexapod robot. Validating adaptations on real-world hardware is only viable for a small number of adaptations. Details of the two phases of experimental work are covered in Section 5.3.

A successful solution controller is demonstrated in Section 5.4. The experimental results specific to the first phase are presented in Section 5.5. Results specific to the second phase are presented in Section 5.6. The results in this chapter are compared to

the SNS experimental observations achieved in the previous chapter (Section 5.7). Lastly, conclusions are discussed in Section 5.8.

5.2 Experimental Procedure

The experimental work consists of two phases, namely the simulated BNS experiments (Methodology B) and the BNS validation experiments (Methodology C). Illustrations of these methodologies are given in Figures 5.1 and 5.2. A detailed discussion of these methodologies can be found in Chapter 3. Purely simulated BNS experiments benchmark the proposed adaptations on a quantitative level. Methodology B uses the Static SNNs developed in the previous chapter as a substitute to the real-world Hexapod robot. Controllers sampled and evaluated using the Static SNNs always include simulator noise in order to represent noise inherent in real-world controller evaluations. Top adaptations identified during the simulated BNS experiments are selected for validation. The validation experimental work studies the BNS approach on a real-world Hexapod robot in order to demonstrate and validate real-world viability.

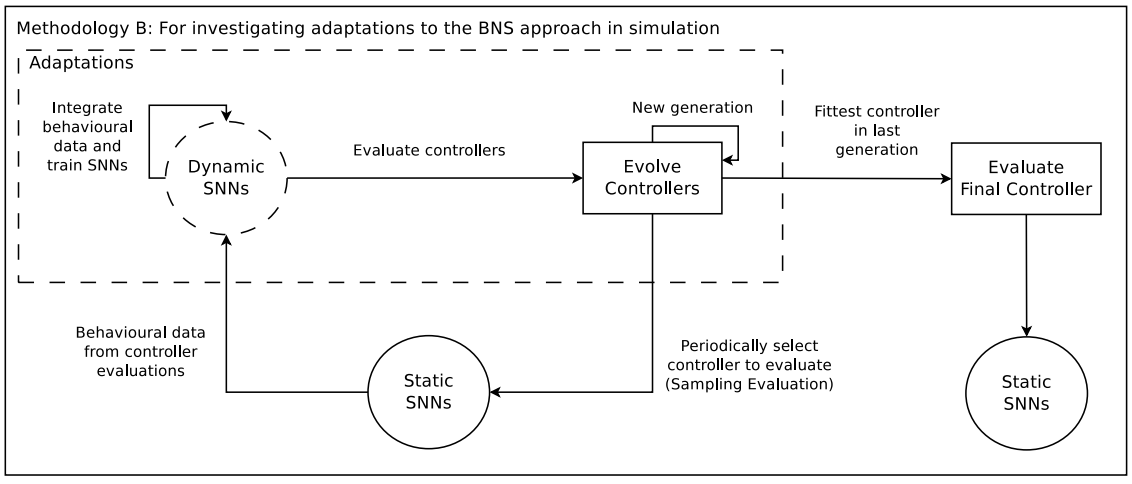


Figure 5.1: Methodology B: Adaptations to the BNS approach

The experimental procedure followed in the SNS experimental work is slightly modified for the BNS approach. Implementation details specific to data acquisition, controller design and simulator development require modifications when applied to the BNS approach. Data acquisition is more tightly integrated into the ER process. The BNS approach

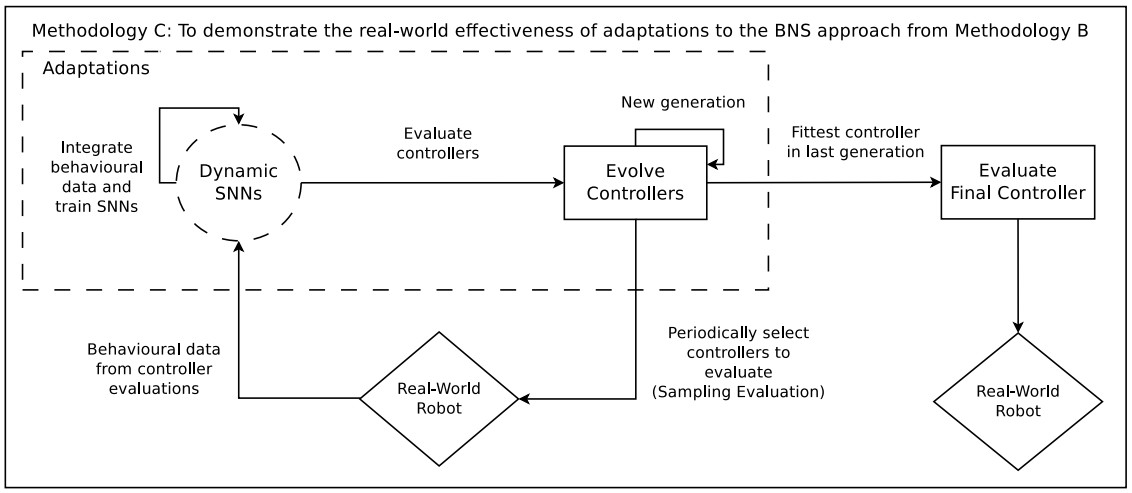


Figure 5.2: Methodology C: Real-world validation of adaptations

requires real-time data acquisition and simulator training during the ER process. Modifications to the data acquisition process are discussed in Section 5.2.1. The controller and simulator designs remain similar to that of the SNS approach, however, modifications specific to the BNS approach are required. Changes to the controller design are discussed in Section 5.2.2. Similarly, changes to the simulator design and training process are covered in Section 5.2.3.

5.2.1 Hardware and Data Capture

The same Hexapod robot described for the SNS experimental work is used (Section 3.2.1). The BNS approach requires real-time data collection, processing and training. Controller evaluations are continually performed in order to collect behavioural data during the ER process. The simulator is continually improved based on the behavioural data collected. Controllers are selected for real-world evaluations from the latest controller population. Behavioural data is not randomly generated but is acquired through the evaluation of evolving solutions throughout the execution of the BNS approach.

5.2.2 Controllers

The controller design (Section 4.2.1) and fitness function (Algorithm 2) is identical to that used in the SNS approach, however, uncertainty penalty normalisation is by necessity

quite different. For the SNS approach, uncertainty penalties are normalised based on the maximum standard deviations observed in the test dataset for the respective behavioural components. However, for the BNS approach, no test dataset exists upfront. It is difficult to normalise prediction standard deviations when little data is available. Parameter settings needed to normalise the standard deviation of SNN predictions is initially unknown.

This work proposes a method of normalising prediction standard deviations based on predictions observed every controller evolution generation. For a population of controllers, the standard deviations for each behavioural component simulated is recorded. These standard deviations are divided by the maximum standard deviation observed for each behavioural component. The fitness function uses the normalised standard deviations in order to calculate the uncertainty penalties of controllers.

Due to no behavioural dataset being available upfront for the BNS approach, the normalisation parameters are continually recalculated. The maximum standard deviation for each behavioural component is calculated from predictions encountered through the evaluation of the controller population. These maximum standard deviations in predictions are used to normalise all prediction standard deviations during fitness calculations. The normalisation parameters are recalculated every generation based on the standard deviation in predictions of the current controller population.

5.2.3 Simulator

A behavioural dataset is not completely available during most of the execution of the BNS approach. Not having a complete behavioural dataset creates certain complications with regards to Dynamic SNNs. Three aspects regarding Dynamic SNNs require special attention. Namely, the standardisation process applied to the behavioural patterns used for training, the Dynamic SNN training process and how simulator noise is applied.

The behavioural data standardisation process improves the accuracy of SNNs. The BNS approach cannot use the same training data standardisation process used in the SNS approach. Standardisation transforms all behavioural data features and outputs to have a zero mean and a unit standard deviation. For the SNS approach, the behavioural data standardisation parameters are calculated based on the fully available behavioural dataset. A full behavioural dataset is not available for most of the BNS approach. Standardisation

parameters need to be dynamically calculated based on the latest available behavioural dataset. The standardisation parameters are dynamic and are re-calculated using the mean and standard deviation of the latest available behavioural dataset. These dynamic standardisation parameters change significantly during the early stages of the BNS approach and stabilise as more behavioural data is acquired. This dynamic standardisation process is a novel contribution of this research.

Dynamic SNNs are trained for 1000 iterations of the Adam algorithm after every half dozen behavioural patterns are collected (twice per sampling evaluation). Early stopping is used where training is terminated if the validation dataset error does not improve within 10 iterations. Only weights associated with the lowest validation dataset MSE are used for the Dynamic SNNs. This can be thought of as a check-pointing procedure, where only the best SNN weights are used based on the validation MSE. For each behavioural pattern collected, there is an 80% probability of it being added to the training dataset and a 20% probability of being added to the validation dataset. Checking and avoiding overfitting of the Dynamic SNNs is a novel contribution of this thesis. Prior research has never tested a mechanism for avoiding overfitting during the BNS approach.

Certain adaptations inject noise into the simulated (Dynamic SNNs) controller evaluations during controller evolution. The BNS approach requires a method for dynamically calculating the noise distribution based on the latest available behavioural dataset. This is due to the absence of a complete behavioural dataset at the start of the BNS approach. This research is the first time that dynamic noise distribution parameter settings have been proposed and investigated for the BNS approach. The level of noise injected into simulator (Dynamic SNNs) evaluations is dynamically calculated. The parameter settings used to generate simulator noise are estimated using the validation dataset. This estimation process takes place after every training phase has ended (twice per sampling evaluation). The noise distribution used to inject noise into each behavioural component is Gaussian with a mean of zero. The standard deviation of the noise is calculated as the standard deviation of the errors observed between the simulator and the latest available validation dataset.

5.3 BNS Experiments

The last two out of three phases of experimental work conducted on the Hexapod robot are covered in this chapter. The first phase, investigating adaptations to the SNS approach, was covered in the previous chapter. The second phase investigates the BNS approach within a purely simulated environment without a real-world robot (Section 5.3.1). A purely simulated environment enables extensive analysis and comparisons between BNS adaptations. The last phase of the experimental work demonstrates and validates the viability of the BNS approach on a real-world Hexapod robot. Top performing BNS adaptations are selected based on the Simulated BNS Experimental observations and experiments are repeated for validation on the real-world Hexapod robot (Section 5.3.2).

5.3.1 The Simulated BNS Experiments

The parameter settings chosen for the BNS approach remain unchanged compared to the SNS approach (Table 4.1). However, an additional parameter setting is required for the sampling strategy. A sampling strategy is the method used for selecting controllers from the controller population and evaluating on a target robot. Two different sampling strategies are investigated, namely, the **High Fitness** and **Most Uncertain** approaches. The first uses a **High Fitness** sampling strategy where a tournament selection of 70 randomly chosen controllers are selected from the controller population and the controller with the highest fitness is selected. The second method (**Most Uncertain**) simply selects the controller with the highest accumulation of uncertainty penalties to be evaluated. The **Most Uncertain** sampling strategy can only be applied to adaptations that produce uncertainty information.

The BNS adaptations consist of changes to the simulator configuration, simulator noise, resetting procedures and sampling strategy. A total number of 64 unique combinations of adaptation settings are investigated (Tables 5.1 to 5.4). The configurations are separated into different tables according to the resetting procedure. An encoded naming scheme is used to represent each adaptation. The encoded names are given in the first columns of Tables 5.1 to 5.4.

The first letter of the encoding scheme stands for the robot morphology, namely, the

Hexapod robot (**H**: Hexapod). The second letter indicates the simulator configuration (**B**: Basic, **E**: Ensemble, **D**: Dropout, **M**: Ensemble Multi-output, **S**: Basic Multi-output) and the third letter indicates the resetting procedure (**N**: None, **C**: Controller, **S**: Simulator, **B**: Both). The fourth letter indicates if simulator noise is present (**N**: for including simulator noise; **E**: for exclude simulator noise). Finally, the sampling strategy (**T**: Tournament, **U**: Most uncertain) is represented by the last letter of the encoding scheme. Thirty independent trial runs of the BNS approach are conducted per adaptation. Each adaptation produces 30 independent solution controllers to make a total of 1920 solutions over all 64 adaptations.

Adaptations	Simulator Configuration	Resetting Procedure	Simulator Noise	Uncertainty Penalties	Samplng Strategy
HBNNT	Basic	None	Yes	No	High Fitness
HBNET	Basic	None	No	No	High Fitness
HENNT	Ensemble	None	Yes	Yes	High Fitness
HENNU	Ensemble	None	Yes	Yes	Most Uncertain
HENEU	Ensemble	None	No	Yes	Most Uncertain
HENET	Ensemble	None	No	Yes	High Fitness
HDNNT	Dropout	None	Yes	Yes	High Fitness
HDNET	Dropout	None	No	Yes	High Fitness
HDNNU	Dropout	None	Yes	Yes	Most Uncertain
HDNEU	Dropout	None	No	Yes	Most Uncertain
HMNEU	Ensemble Multi-output	None	No	Yes	Most Uncertain
HMNNT	Ensemble Multi-output	None	Yes	Yes	High Fitness
HMNNU	Ensemble Multi-output	None	Yes	Yes	Most Uncertain
HMNET	Ensemble Multi-output	None	No	Yes	High Fitness
HSNNT	Basic Multi-output	None	Yes	No	High Fitness
HSNET	Basic Multi-output	None	No	No	High Fitness

Table 5.1: BNS adaptations using no resetting procedure

The procedure for each BNS simulated experimental run is as follows:

1. A specific adaptation is chosen from the configurations listed in Tables 5.1 to 5.4.
2. The BNS approach described in Methodology B is performed.
3. The ER process continues until 100 controllers have been evaluated using the substitute real-world simulator.

Adaptations	Simulator Configuration	Resetting Procedure	Simulator Noise	Uncertainty Penalties	Sampling Strategy
HBCNT	Basic	Controller	Yes	No	High Fitness
HBCET	Basic	Controller	No	No	High Fitness
HECNT	Ensemble	Controller	Yes	Yes	High Fitness
HECNU	Ensemble	Controller	Yes	Yes	Most Uncertain
HECEU	Ensemble	Controller	No	Yes	Most Uncertain
HECET	Ensemble	Controller	No	Yes	High Fitness
HDCNT	Dropout	Controller	Yes	Yes	High Fitness
HDCET	Dropout	Controller	No	Yes	High Fitness
HDCNU	Dropout	Controller	Yes	Yes	Most Uncertain
HDCEU	Dropout	Controller	No	Yes	Most Uncertain
HMCEU	Ensemble Multi-output	Controller	No	Yes	Most Uncertain
HMCNT	Ensemble Multi-output	Controller	Yes	Yes	High Fitness
HMCNU	Ensemble Multi-output	Controller	Yes	Yes	Most Uncertain
HMCET	Ensemble Multi-output	Controller	No	Yes	High Fitness
HSCNT	Basic Multi-output	Controller	Yes	No	High Fitness
HSCET	Basic Multi-output	Controller	No	No	High Fitness

Table 5.2: BNS adaptations using the controller resetting procedure

Adaptations	Simulator Configuration	Resetting Procedure	Simulator Noise	Uncertainty Penalties	Sampling Strategy
HBSNT	Basic	Simulator	Yes	No	High Fitness
HBSET	Basic	Simulator	No	No	High Fitness
HESNT	Ensemble	Simulator	Yes	Yes	High Fitness
HESNU	Ensemble	Simulator	Yes	Yes	Most Uncertain
HESEU	Ensemble	Simulator	No	Yes	Most Uncertain
HESET	Ensemble	Simulator	No	Yes	High Fitness
HDSNT	Dropout	Simulator	Yes	Yes	High Fitness
HDSET	Dropout	Simulator	No	Yes	High Fitness
HDSNU	Dropout	Simulator	Yes	Yes	Most Uncertain
HDSEU	Dropout	Simulator	No	Yes	Most Uncertain
HMSEU	Ensemble Multi-output	Simulator	No	Yes	Most Uncertain
HMSNT	Ensemble Multi-output	Simulator	Yes	Yes	High Fitness
HMSNU	Ensemble Multi-output	Simulator	Yes	Yes	Most Uncertain
HMSET	Ensemble Multi-output	Simulator	No	Yes	High Fitness
HSSNT	Basic Multi-output	Simulator	Yes	No	High Fitness
HSSET	Basic Multi-output	Simulator	No	No	High Fitness

Table 5.3: BNS adaptations using the simulator resetting procedure

Adaptations	Simulator Configuration	Resetting Procedure	Simulator Noise	Uncertainty Penalties	Sampling Strategy
HBBNT	Basic	Both	Yes	No	High Fitness
HBBET	Basic	Both	No	No	High Fitness
HEBNT	Ensemble	Both	Yes	Yes	High Fitness
HEBNU	Ensemble	Both	Yes	Yes	Most Uncertain
HEBEU	Ensemble	Both	No	Yes	Most Uncertain
HEBET	Ensemble	Both	No	Yes	High Fitness
HDBNT	Dropout	Both	Yes	Yes	High Fitness
HDBET	Dropout	Both	No	Yes	High Fitness
HDBNU	Dropout	Both	Yes	Yes	Most Uncertain
HDBEU	Dropout	Both	No	Yes	Most Uncertain
HMBEU	Ensemble Multi-output	Both	No	Yes	Most Uncertain
HMBNT	Ensemble Multi-output	Both	Yes	Yes	High Fitness
HMBNU	Ensemble Multi-output	Both	Yes	Yes	Most Uncertain
HMBET	Ensemble Multi-output	Both	No	Yes	High Fitness
HSBNT	Basic Multi-output	Both	Yes	No	High Fitness
HSBET	Basic Multi-output	Both	No	No	High Fitness

Table 5.4: BNS adaptations using the controller and simulator resetting procedures

4. The fittest controller in the final controller population is selected as the solution.

5. For the solution controller, the paths generated by the Static and Dynamic simulators are collected for analysis.

The time taken to evaluate controllers on a real-world robot and the complexity of the simulator used affect the design of the simulated BNS experiments. Evaluation times do not need to be simulated but the number of controller evolution generations iterated per sampling evaluation needs to be simulated based on real-world estimates. Controller evolution is carried out throughout the BNS approach and many controller population generations are iterated per sampling evaluation. The number of controller evolution generations processed per sampling evaluation is significantly different between simulator configurations. More complex simulator configurations slow down controller evaluations and consequentially training times. The number of generations achieved per sampling evaluation is given in Table 5.5. The simulated BNS experiments are configured to process a certain number of generations per selective evaluation. Adaptations using the **Basic** simulator configurations iterate through 156 generations for each sampling evaluation.

Adaptations consisting of **Ensemble** or **Dropout** simulator configurations have the lowest number of generations per sampling evaluation at 55 and 74 generations, respectively. The **Basic** and **Dropout** simulator configurations have equally complex architectures, however, evaluating controllers using the later configuration is significantly more computationally expensive. Adaptations consisting of an **Ensemble Multi-output** simulator configuration iterates 150 generations per sampling evaluation. Adaptations using a **Basic Multi-output** simulator configuration achieve the highest number of generations per sampling evaluation at 280 generations. The total number of controller evolution generations iterated for the tested adaptations can differ significantly from each other depending on the simulator configuration used. Certain adaptations have the advantage of a faster evolution rate compared to others.

Simulator Configuration	Number Generations per Sampling Evaluation
Basic	156
Dropout	74
Ensemble	55
Ensemble Multi-output	150
Basic Multi-output	280

Table 5.5: Number of controller evolution generations iterated per sampling controller evaluation for the BNS approach on the Hexapod robot

Adaptations are studied in terms of their performance, transferability and convergence properties. The top 10 best performing adaptations are identified and selected for further investigations.

5.3.2 The BNS Validation Experiments

The BNS validation experiments follow the approach specified in Methodology C. The experimental procedure for the BNS validation experiments are similar to that of the Simulated BNS Experimental work (Methodology B), except that a real-world robot is used instead of a simulation.

Six promising BNS adaptations are identified from the results of the Simulated BNS

Experimental work and selected for real-world testing. Each selected adaptation is validated through 5 independent experimental runs of the BNS approach (Methodology C). The paths generated by the final simulated and real-world evaluations are captured, analysed and presented in Section 5.6.

The procedure for each BNS validation experimental run is as follows:

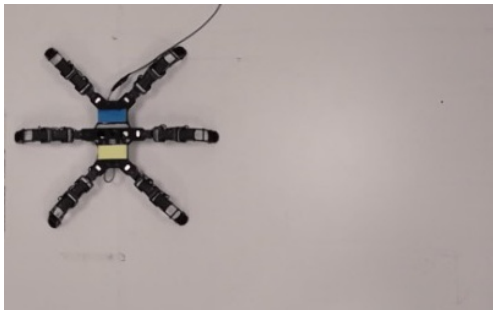
1. A specific adaptation is chosen from the configurations listed in Tables 5.1 to 5.4.
2. The BNS approach described in Methodology C is performed.
3. The ER process continues until 100 controllers have been evaluated using the real-world robot.
4. The fittest controller in the final controller population is selected as the solution.
5. For the solution controller, the paths generated in simulation and reality are collected for analysis.

5.4 Successful BNS Hexapod Controller

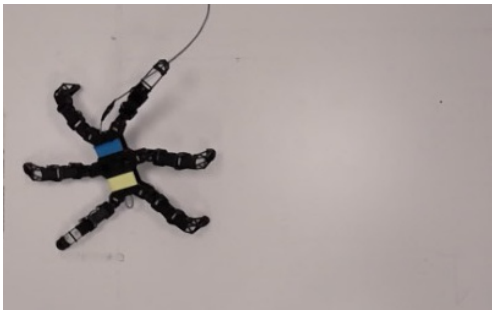
The BNS approach is successfully used to develop effective distance maximising gait controllers for the Hexapod robot. A BNS simulator can be trained to accurately simulate Hexapod robot behaviours during the ER process. Less behavioural data is collected when compared to the SNS approach. Some of the BNS adaptations investigated perform and transfer similarly well compared to the best SNS adaptation tested for the Hexapod robot. The BNS approach is a robust method for developing gait maximising controllers for the Hexapod robot. This section demonstrates a single example solution produced by the BNS approach.

A time-lapse demonstration[1] of a solution developed using the BNS approach is presented in Figure 5.3. The simulated and real-world trajectories are also given in Figure 5.4. The simulated path generated by the Static Simulator is presented in order to demonstrate that the simulator developed by the BNS approach is more accurate in comparison.

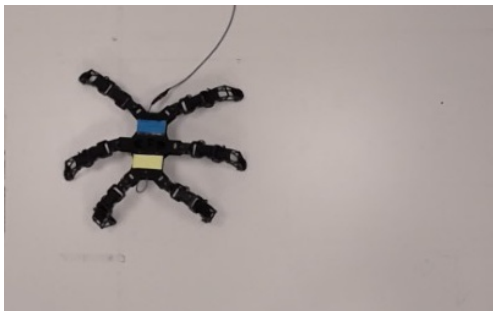
[1] https://youtu.be/8ZWUWEjL1Cw



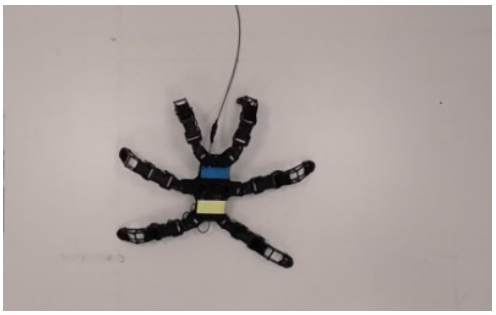
(a) Start



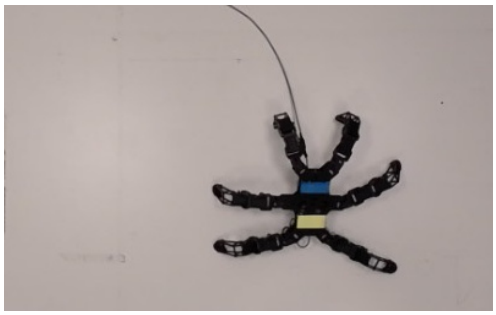
(b) 6 seconds



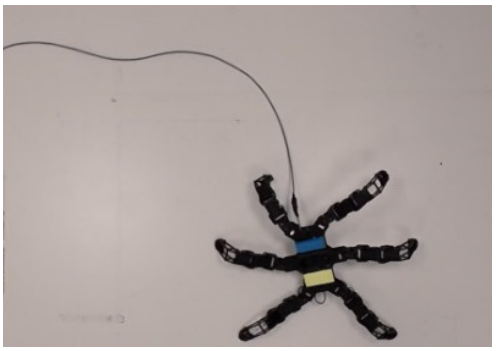
(c) 12 seconds



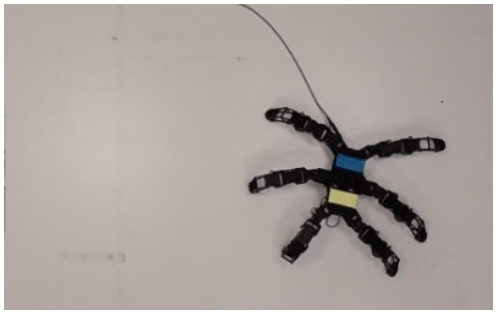
(d) 18 seconds



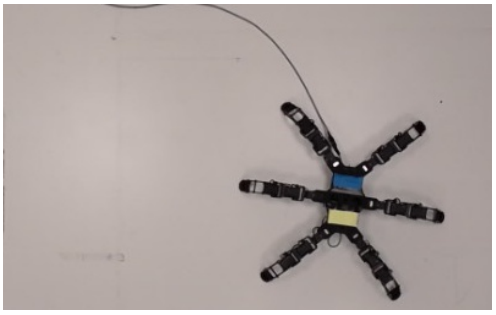
(e) 24 seconds



(f) 30 seconds



(g) 36 seconds



(h) 42 seconds

Figure 5.3: Solution controller demonstration

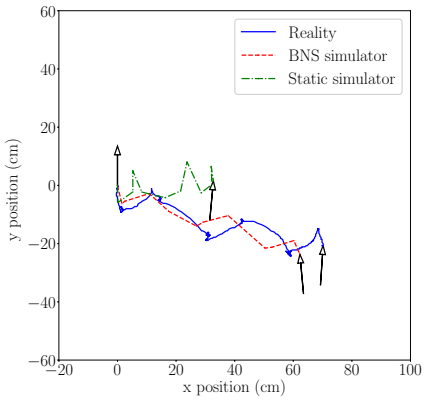


Figure 5.4: Real-world and simulated trajectories

The time-lapse is captured such that frames are 6 seconds apart. The starting position (Figure 5.3a) begins with all legs making contact with the ground. The robot reaches downwards in the first 6 seconds (Figure 5.3b), followed by stepping towards the right (Figure 5.3c). The robot steps moves towards the bottom-right in order to reach the position presented in Figure 5.3d after which the robot side shifts further right (Figure 5.3e). The robot reaches downwards (Figure 5.3f) and repositions its legs in order to reach upward (Figure 5.3g). Lastly, the robot adjusts to its initial starting stance in Figure 5.3h.

5.5 The Simulated BNS Experiment Results

The BNS adaptations in prior work used the **Basic** simulator configuration with simulator noise, no resetting of any kind and High Fitness sampling. Results demonstrate that many novel variations on the BNS approach perform significantly better than the baseline BNS approach. High level performance comparisons between adaptation settings are covered in Section 5.5.1. Differences between the transferability distributions of different adaptation settings are presented in Section 5.5.2. The convergence properties over the lifetime of the BNS approach for the different resetting procedures are studied in Section 5.5.3.

Grouping adaptations according to the simulator configuration, resetting procedures, simulator noise or sampling strategy may not be necessarily give good indications of the best adaptations settings to use. For example, the **Ensemble** simulator configuration

may not perform well in aggregate over other adaptation settings but could show excellent performance for specific adaptation settings. In order to account for such a scenario, the best 10 adaptations are studied in Section 5.5.4. Lastly, a summary of the findings is given in Section 5.5.5.

5.5.1 Overall Comparisons

This section presents the performances of adaptation settings discussed in Section 3.7. Adaptations investigated include the Simulator Configurations (Section 5.5.1.1), Resetting Procedures (Section 5.5.1.2), Simulator Noise (Section 5.5.1.3) and Sampling Strategies (Section 5.5.1.4). Every possible adaptation setting is grouped over all other adaptation settings and the performance distributions studied. The performance properties of adaptation settings are identified and discussed. An overall summary of the comparison work is discussed in Section 5.5.1.5.

5.5.1.1 Simulator Configurations

The performance distributions for the tested simulator configurations are illustrated as standard box-plots in Figure 5.5. Summary statistics are given in Table 5.6. All statistical comparisons use a significance level of 5%. The Kruskal-Wallis H test is used to determine whether the performance distributions of the tested simulator configurations originate from the same distribution. The performance distributions for the tested the simulator configurations are significantly different from each other, with a p-value of 1.35×10^{-52}. A post hoc comparison is performed using pairwise Mann-Whitney U tests. The p-values obtained are given in Table 5.7.

The **Basic** simulator configuration is considered the baseline configuration against which all other configurations can be measured. The **Ensemble Multi-output** simulator configuration performs significantly better than all other simulator configurations. The second best performer is the **Basic Multi-output** simulator configuration. A Mann-Whitney U test indicates a significant performance difference between the **Ensemble Multi-output** and **Basic Multi-output** simulator configurations, with a p-value of 0.0132. The **Ensemble Multi-output** simulator configuration's performance IQR is

between 15.8 and 36.1 centimetres. The performance IQR for the **Basic Multi-output** simulator configuration is between 13.1 and 32.3 centimetres.

The overall performance distribution of the **Basic** simulator configuration is statistically equal to the **Ensemble** and **Basic Multi-output** simulator configurations. The performance IQR for the **Basic** simulator configuration is between 12.0 and 29.6 centimetres while the **Ensemble** configuration is between 10.5 and 30.4 centimetres. The use of either ensembles or multi-output SNNs alone does not significantly improve overall performance outcomes when compared to the baseline. The **Dropout** simulator configuration performs significantly worse than all other configurations with an IQR between 6.9 and 17.4 centimetres.

The performance distributions are skewed towards the right where average performances are higher than median performances. The performance distribution for the **Dropout** simulator configuration is less skewed than other configurations which is probably due to poor performance and low standard deviation.

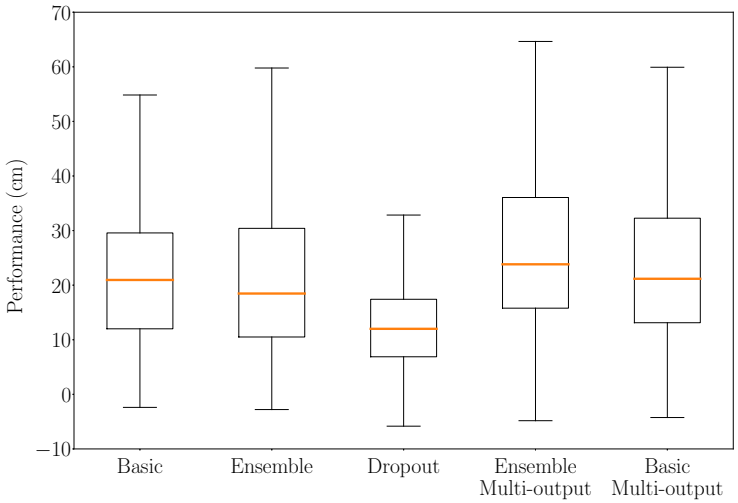


Figure 5.5: Performance comparisons between simulator configurations

5.5.1.2 Resetting Procedures

Performance distributions for the tested resetting procedures are illustrated in Figure 5.6 and summary statistics are given in Table 5.8. Similar to the simulator configurations, the

	Mean	Median	Q_1	Q_3	Std. Dev.
Basic	23.4	21.0	12.0	29.6	16.1
Ensemble	22.4	18.5	10.5	30.4	16.7
Dropout	13.4	12.0	6.9	17.4	10.1
Ensemble Multi-output	*27.4*	*23.8*	15.8	36.1	16.9
Basic Multi-output	24.8	21.2	13.1	32.3	17.4

Table 5.6: Summary statistics for the simulator configuration performance distributions

	Basic	Ensemble	Dropout	Ensemble Multi-output
Ensemble	1.94×10^{-1}	-		
Dropout	1.52×10^{-19}	2.12×10^{-20}	-	
Ensemble Multi-output	4.07×10^{-4}	1.05×10^{-8}	1.13×10^{-50}	-
Basic Multi-output	3.59×10^{-1}	2.54×10^{-2}	4.26×10^{-23}	1.32×10^{-2}

Table 5.7: The p-values of post hoc analysis comparing performance distributions between simulator configurations

resetting procedure performance distributions are skewed towards the right. Performance distributions are skewed towards the right if the mean performance is greater than the median. Most adaptations within each distribution perform poorly while some adaptations perform disproportionately well.

The Kruskal-Wallis H test is used to determine whether the performance distributions of the tested resetting procedures originate from the same distribution. The performance distributions for the tested resetting procedures are significantly different from one another, with a p-value of 1.76×10^{-113}. A post hoc pairwise comparison is performed using Mann-Whitney U tests. The p-values obtained are given in Table 5.9. All resetting procedures have significantly different performance distributions when compared to each other.

The no resetting procedure is considered the baseline against which improvement are measured. Results demonstrate that the no resetting procedure has the worst overall performance distribution compared to the newly proposed resetting procedures. Periodically resetting the simulator performs significantly better than all other tested resetting proce-

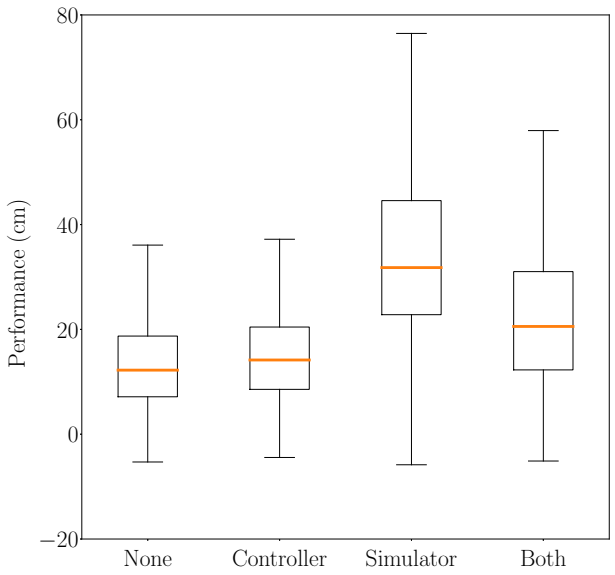


Figure 5.6: Performance comparison between resetting procedures

dures. Results indicate that periodically resetting the simulator during the ER process significantly increases the likelihood of discovering good controller solutions. The lower quartile of the simulator resetting procedure is higher than the upper quartiles of the no resetting and controller resetting procedures.

SNN training is likely to stop early if the validation dataset MSE shows no improvement after 10 iterations of the Adam training algorithm. Premature stopping would result in the SNNs not learning the newly acquired behavioural patterns. The simulator may then become more specialised at simulating solutions obtained during the early stages of the BNS approach. Periodically resetting the simulator and completely retraining the SNNs likely helps avoids bias in the prediction models.

The second best procedure resets both the controllers and simulator. Resetting both the controller and simulator does not perform as well as only resetting the simulator. Periodically resetting the simulator has a greater ability to improved the likely performance outcomes of solutions than resetting controllers.

Periodically resetting the controller population helps the ER process to escape sub-optimal convergence in fitness. Periodically resetting controllers improvements the likely

	Mean	Median	Q_1	Q_3	Standard Deviation
None	14.0	12.2	7.1	18.7	11.0
Controller	15.2	14.2	8.6	20.4	9.2
Simulator	*34.7*	*31.8*	22.8	44.6	18.1
Both	23.4	20.6	12.3	31.0	15.7

Table 5.8: Summary statistics for the resetting procedure performance distributions

	None	Controller	Simulator
Controller	1.94×10^{-3}	-	
Simulator	5.18×10^{-84}	4.29×10^{-78}	-
Both	6.06×10^{-28}	1.22×10^{-18}	1.38×10^{-27}

Table 5.9: The p-values from post hoc analysis comparing performance distributions between resetting procedures

performance outcomes compared to the no resetting procedure. The IQR of the controller resetting procedure is between 8.6 and 20.4 centimetres while the no resetting procedure is slightly lower being between 7.1 and 18.7 centimetres. After controller resets the ER process can better traverse the updated fitness landscape. However, completely resetting the controller population means that all known good solutions are lost and need to be rediscovered which can negatively affect performance.

If simulator is reset, it is likely that the controller search space might change significantly. Any solution convergence in the controller population might be disturbed due to simulator resets. The controller population is less impacted by simulator resets during the later stages of the BNS approach. Periodically resetting the simulator has the highest observed IQR between 22.8 and 44.6 centimetres while resetting both controllers and the simulator is only between 12.3 and 31.0 centimetres.

5.5.1.3 Simulator Noise

Performance distributions for the inclusion or exclusion of simulator noise are illustrated in Figure 5.7 and summary statistics given in Table 5.10. A comparison between per-

formance distributions for the inclusion or exclusion of simulator noise is tested using a Mann-Whitney U test. A p-value of 4.42×10^{-12} is obtained. Not injecting noise into the simulator performs significantly better compared to when simulator noise is included. Adding simulator noise likely results in many random changes to the fitness landscape during the ER process. Many random changes in the fitness landscape may be impairing the ability of the ER process to exploit known good solutions. The performance distributions are skewed towards the right where a subset of adaptations perform disproportionately better than others. The IQR for not injecting noise into the simulator is between 12.0 and 32.3 centimetres. The IQR for adaptations that include simulator noise is between 9.6 and 25.7 centimetres.

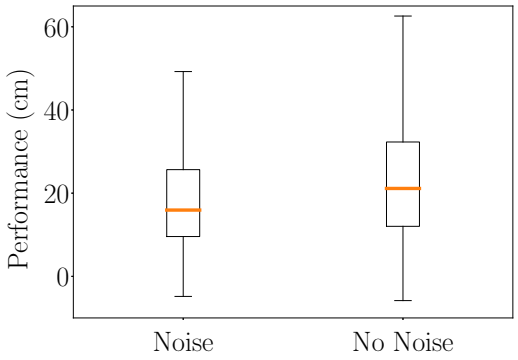


Figure 5.7: Performance comparison for simulator noise

Has Simulator Noise	Mean	Median	Q_1	Q_3	Standard Deviation
Noiseless	*24.4*	*21.1*	12.0	32.3	17.5
Noise	19.2	15.9	9.6	25.7	14.4

Table 5.10: Summary statistics for the simulator noise performance distributions

5.5.1.4 Sampling Strategies

Performance distributions for the tested sampling strategies are illustrated in Figure 5.8 and summary statistics are given in Table 5.11. Evaluating the most uncertain controller for every sampling evaluation does not perform significantly better compared to a High

Fitness sampling strategy. The High Fitness sampling strategy has a slightly better mean, median and IQR compared to selectively evaluating the most uncertain controllers. A comparison between the performance distributions of the tested sampling strategies achieves a p-value of 0.24 for the Mann-Whitney U test. The tested sampling strategy performance distributions are not significantly different from each other.

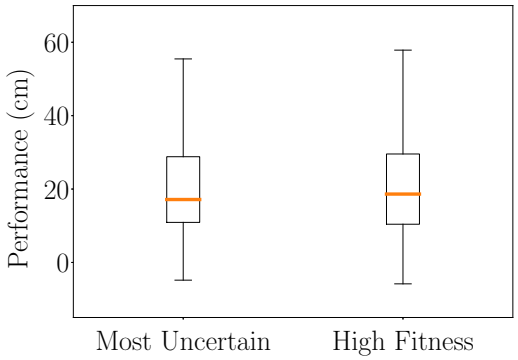


Figure 5.8: Performance comparisons of sampling strategies

	Mean	**Median**	**Q_1**	**Q_3**	**Standard Deviation**
Most Uncertain	21.4	17.1	10.9	28.8	16.1
High Fitness	22.1	18.6	10.4	29.6	16.3

Table 5.11: Summary statistics of sampling strategy performance distributions

5.5.1.5 Summary

The most influential adaptation settings for improving the likely performance of adaptations is the resetting procedure used. The simulator resetting procedure is the most beneficial adaptation setting tested for improving performance outcomes of solution controllers developed for the Hexapod robot. The simulator configuration is the second most important adaptation setting for improving the likely performance outcomes of solutions. The **Ensemble Multi-output** simulator configuration is identified as the best configuration for improving overall performance outcomes. Not including simulator noise during controller evolution is the third best technique for improving the overall performance

outcomes. The tested sampling strategies do not greatly affect the likely performance outcomes of solutions.

5.5.2 Transferability

This section presents the transferability of adaptation settings discussed in Section 3.7. Adaptation settings investigated include the Simulator Configurations (Section 5.5.2.1), Resetting Procedures (Section 5.5.2.2), Simulator Noise (Section 5.5.2.3) and Sampling Strategies (Section 5.5.2.4). Every possible adaptation setting is grouped over all other adaptation settings and the transferability distributions studied. An overall summary of the comparison work is discussed in Section 5.5.2.5.

5.5.2.1 Simulator Configurations

The transferability distributions for the tested simulator configurations are illustrated in Figure 5.9 and summary statistics are given in Table 5.12. Lower transferability values indicate a closer correspondence between simulation and reality. The Kruskal-Wallis H test is used to determine whether the transferability distributions for the tested simulator configurations originate from the same distribution. The p-value for the Kruskal-Wallis H test is 6.94×10^{-66} which is highly significant. The transferability distributions of the tested simulator configurations are significantly different from each other.

	Mean	**Median**	$\mathbf{Q_1}$	$\mathbf{Q_3}$	**Std. Dev.**
Basic	2.34	1.32	0.71	2.82	2.66
Dropout	4.74	3.04	1.77	5.37	6.29
Ensemble	2.37	1.25	0.54	2.68	4.47
Ensemble Multi-output	1.70	0.96	0.49	1.89	2.96
Basic Multi-output	2.40	1.56	0.59	2.82	4.05

Table 5.12: Transferability statistics for the simulator configurations

A post hoc comparison between simulator configurations is performed using pairwise Mann-Whitney U tests. The p-values for the pairwise comparisons are given in Table 5.13. The **Basic**, **Ensemble** and **Basic Multi-output** simulator configurations do not have

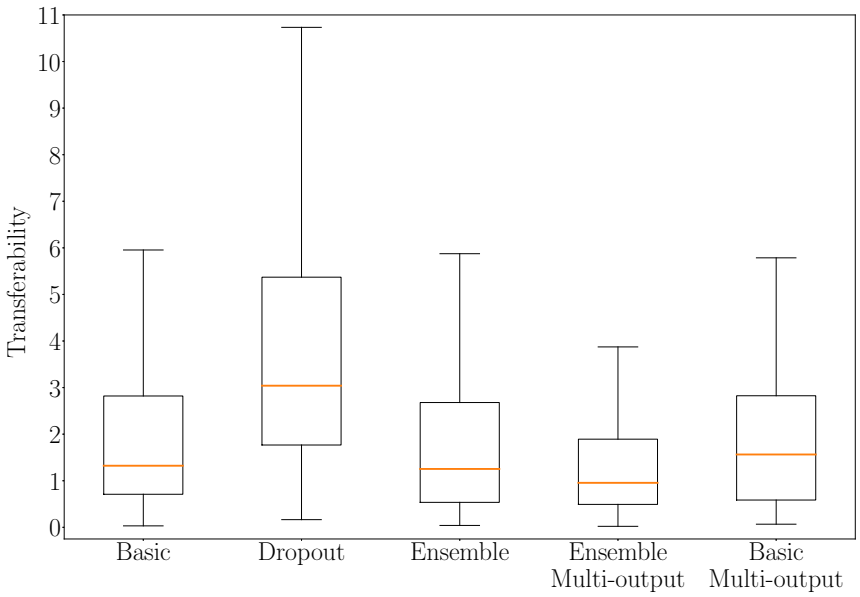


Figure 5.9: Transferability distributions for each simulator configurations

significantly different transferability distributions compared to each other. The **Basic** simulator configuration has an IQR between 0.71 and 2.82 while the **Ensemble** simulator configuration is between 0.54 and 2.68. The **Ensemble** simulator configuration transferability distribution IQR is slightly better than the **Basic** simulator configuration, however, a comparison between the two yields a p-value of 0.147. There is insufficient evidence to conclude that solutions related to the **Ensemble** simulator configuration have significantly better transferability than the **Basic** simulator configuration.

The **Ensemble Multi-output** simulator configuration has the best transferability

	Basic	**Dropout**	**Ensemble**	**E. Multi-output**
Dropout	1.93×10^{-21}	-		
Ensemble	1.47×10^{-1}	2.44×10^{-38}	-	
E. Multi-output	1.41×10^{-5}	6.03×10^{-62}	7.61×10^{-4}	-
Basic Multi-output	8.68×10^{-1}	2.56×10^{-21}	1.78×10^{-1}	3.33×10^{-5}

Table 5.13: Transferability distribution p-values for comparisons between simulator configurations

distribution compared to all other configurations. The IQR for the **Ensemble Multi-output** simulator configuration is between 0.49 and 1.89. The second best transferability outcome is found with the **Ensemble** simulator configuration.

The **Dropout** simulator configuration demonstrated a significantly worse transferability distribution compared to all other configurations. The median transferability for the **Dropout** simulator configuration is 3.04 which is greater than the upper quartile of all other simulator configurations. A partial reason for the **Dropout** simulator configuration performing so poorly is due to the significantly lower real-world distances travelled by solutions. The real-world distance travelled by the robot is an important component of the transferability metric.

An ensemble approach alone does not significantly improve the transferability of solutions compared to a non-ensemble approach. Similarly, the use of multi-outputs SNNs alone does not significantly improve the likely transferability of solutions. However, applying multi-output SNNs within an ensemble configuration (**Ensemble Multi-output**) results in a large improvement in the likely transferability of solutions.

The transferability distributions for the tested simulator configurations are heavily skewed towards the right. The mean transferability for each simulator configuration is higher than its median. The skewness in the transferability distributions is due to subsets of adaptations with disproportionately bad transferability profiles.

5.5.2.2 Resetting Procedures

The transferability distributions of the different resetting procedures are illustrated in Figure 5.10 and summary statistics are given in Table 5.14. The Kruskal-Wallis H test is used to determine if the transferability distributions for the different resetting procedures originate from the same distribution. The p-value of the Kruskal-Wallis H test is 1.06×10^{-146}. The transferability distributions for the tested resetting procedures are significantly different from each other.

Post hoc comparison tests are performed between the resetting procedure transferability distributions. The p-values of the Mann-Whitney U pairwise comparison tests are given in Table 5.15. All p-values obtained from the comparisons are less than the 5% significance level. This indicates that there is a significant difference in the transferability

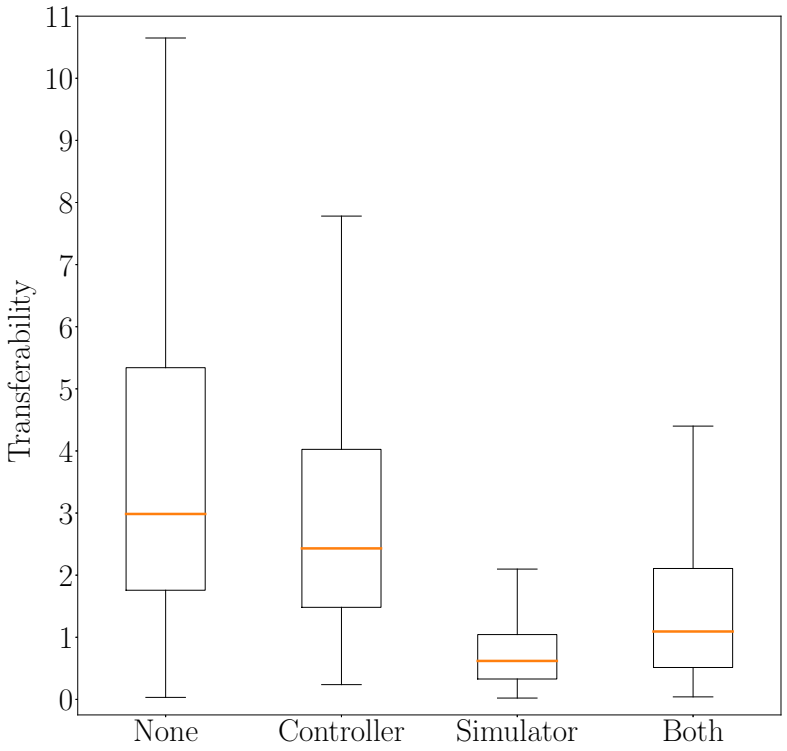


Figure 5.10: Transferability distributions for each resetting procedure

distributions between all resetting procedures.

No periodic resetting of controllers or the simulator results in a significantly worse transferability distribution compared to the newly proposed resetting procedures. The IQR for the no resetting procedure is between 1.76 and 5.34. Periodically resetting controllers has better transferability compared to the no resetting procedure. The IQR for the controller resetting transferability distribution is between 1.48 and 4.03.

The mean transferability for each resetting procedure is higher than its median transferability. However, the simulator resetting procedure has the closest gap between the mean and median transferability statistics.

Results indicate that the best proposal for improving a controller's transferability is the simulator resetting procedure. Periodically resetting the simulator and retraining likely

	Mean	Median	Q_1	Q_3	Standard Deviation
None	4.75	2.99	1.76	5.34	6.60
Controller	3.63	2.43	1.48	4.03	4.78
Simulator	0.96	0.62	0.33	1.04	1.37
Both	1.84	1.09	0.51	2.11	2.87

Table 5.14: Transferability statistics for the resetting procedures

	None	Controller	Simulator
Controller	6.77×10^{-5}	-	
Simulator	1.49×10^{-100}	2.71×10^{-93}	-
Both	1.73×10^{-51}	1.54×10^{-38}	2.79×10^{-18}

Table 5.15: Transferability distribution p-values for comparisons between resetting procedures

helps eliminate modelling bias towards robot behaviours encountered during the early stages of the BNS process. The simulator resetting procedure has the best transferability distribution with an IQR between 0.33 and 1.04. Controller evolution is less likely to exploit weaknesses in the simulators due to frequent resets. Simulator weaknesses before and after resets are likely to be different.

Periodically resetting the controllers and simulator has a significantly better transferability distribution compared to not resetting at all or simply resetting the controller population. The IQR for resetting both is between 0.51 and 2.11. The ER process appears to be unable to recover fully from frequent controller resets.

5.5.2.3 Simulator Noise

The transferability distributions for adaptations including or excluding simulator noise are illustrated in Figure 5.11 and summary statistics are given in Table 5.16. The median transferability achieved when injecting noise into the simulator is 1.76 with an IQR between 0.81 and 3.32. The noiseless simulator approach has a median transferability of 1.35 and an IQR between 0.61 and 2.98. The noiseless simulator approach has a significantly better

	Mean	Median	Q_1	Q_3	Standard Deviation
Noiseless	2.56	1.35	0.61	2.98	4.10
Noise	3.03	1.76	0.81	3.32	5.07

Table 5.16: Transferability statistics for the simulator noise

transferability distribution. The p-value for the Mann-Whitney U test comparing the inclusion or exclusion of simulator noise is 3.71×10^{-4}. The transferability distribution for including simulator noise has a standard deviation of 5.07 while the transferability distribution of the noiseless simulator configuration has a standard deviation of 4.10.

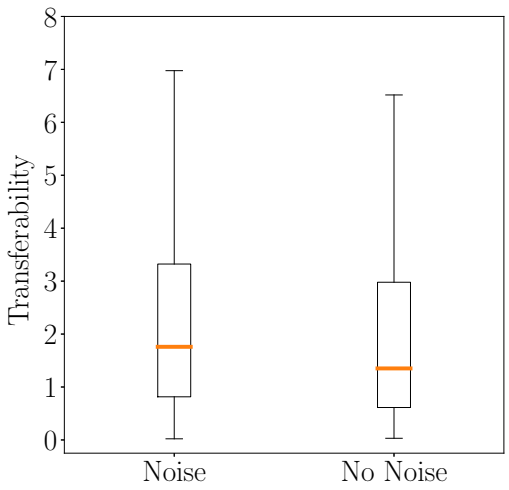


Figure 5.11: Transferability distributions between simulator noise approaches

5.5.2.4 Sampling Strategies

Transferability distributions for the sampling strategies tested are illustrated in Figure 5.12 and summary statistics are given in Table 5.17. The observed mean, median, IQR and standard deviations of the tested sampling strategies are relatively close to one another. The **High Fitness** sampling strategy has a slightly better transferability than selecting the most uncertain controllers. The p-value for the Mann-Whitney U test comparing the transferability distributions of the sampling strategies is 0.568. The sampling strategies

tested do not significantly affect the likely transferability outcomes of solutions.

The median transferability for the **High Fitness** sampling strategy is 1.55 with an IQR between 0.67 and 3.11. When only sampling the most uncertain controller (**Most Uncertain** sampling strategy), the median transferability is 1.63 with an IQR between 0.72 and 3.26. The mean transferability for both sampling strategies is higher than the median. The transferability distributions are highly skewed towards poor transferability outcomes.

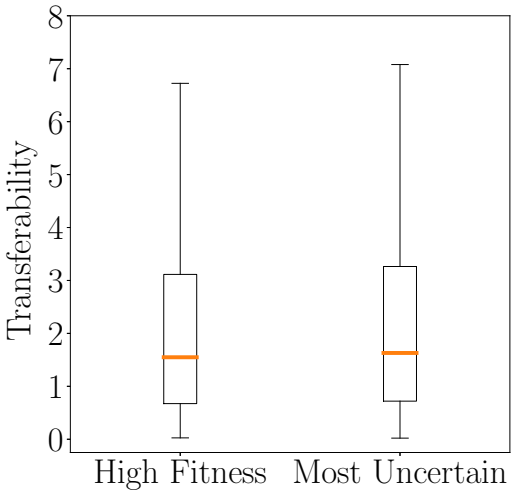


Figure 5.12: Transferability distributions between sampling strategies

	Mean	Median	Q_1	Q_3	Standard Deviation
High Fitness	2.78	1.55	0.67	3.11	4.56
Most Uncertain	2.82	1.63	0.72	3.26	4.72

Table 5.17: Transferability statistics for sampling strategies

5.5.2.5 Summary

The high level transferability properties of tested adaptations settings are now summarised. Periodically resetting the simulator during the BNS approach improves the likely transferability of solutions more than any other adaptation setting tested. The **Ensemble**

Multi-output simulator configuration is the second most influential adaptations setting tested with regards to improving the transferability of solutions. Solution controllers produced from noiseless adaptations are more likely to have better transferability outcomes compared to adaptations with noise. The tested sampling strategies do not affect the likely transferability of solutions.

5.5.3 Convergence Properties

In order to study the performance properties of the tested resetting procedures over time, the median performance over time for the different resetting procedures is given in Figure 5.13. The performance of the fittest controller after every sampling controller evaluation is recorded for all experimental runs. Controllers are evaluated by the Static SNNs and the developing Dynamic SNNs which produces "real-world" and simulated trajectories, respectively. The median performance of the fittest controller after each sampling evaluation is calculated for each resetting procedure.

Performances achieved during the early stages of the BNS approach are poor but gradually improve over time. The median performance of the different resetting procedures appear similar up to the twentieth evaluation, after which median performances significantly diverge from each other. The no resetting procedure is considered the baseline.

The simulator resetting procedure, reset the simulator every 10 sampling controller evaluations. The controller resetting procedure, resets the controller population every 10 sampling controller evaluations. The last proposed resetting procedure is when both the controller population and simulator are reset every 10 sampling controller evaluations.

The no resetting procedure has a consistent but relatively slow upward trend in performance over time. The improvement in performance over time is relatively small compared to procedures that include simulator resetting. After the 60^{th} evaluation, the no resetting procedure appears to show little improvement.

The simulator resetting procedure performs significantly better than all other procedures for the majority of the timeline. Past the 20^{th} evaluation, the simulator resetting procedure almost always outperforms the other procedures. The second best resetting procedure periodically reset both the controllers and simulator.

If the controller population is periodically reset, performance drastically drops every

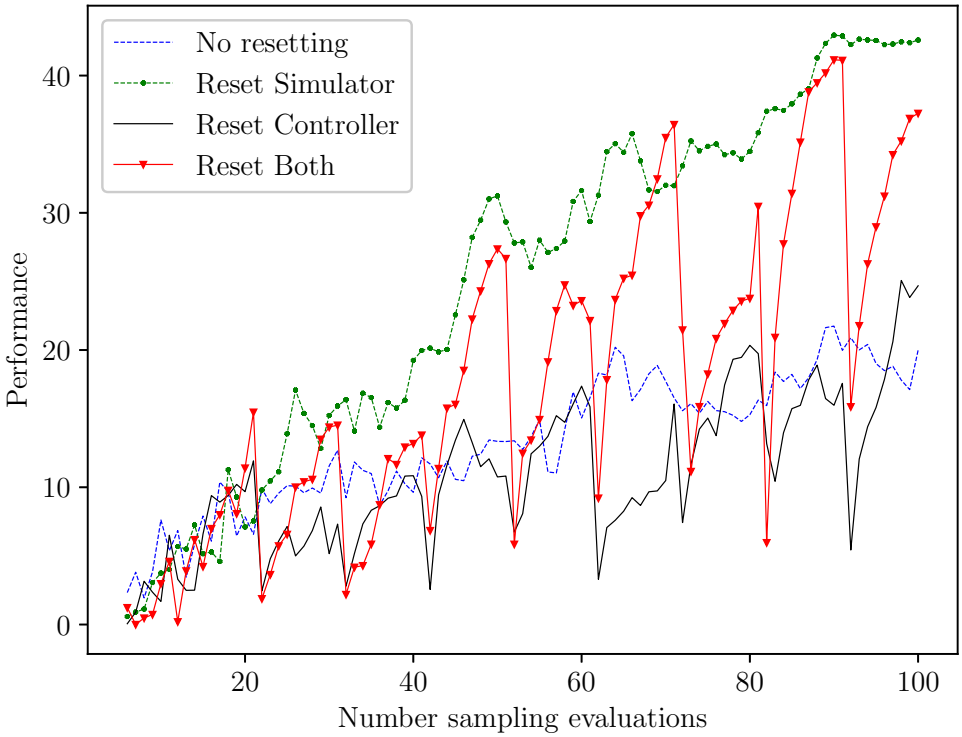


Figure 5.13: Performance over time for the best performer of each resetting procedure

10 sampling controller evaluations, followed by a recovery period. The controller resetting procedure results in the optimisation process spending a significant amount of time recovering from controller resets.

For the resetting procedure that resets both the simulator and controllers, swings in performance are significantly greater than all other procedures. As with the controller resetting procedure, it is likely that controllers are selectively evaluated before the ER process has converged. Behavioural data collected directly after a controller population reset is significantly less relevant to converged solutions compared to behavioural data collected before a controller population reset. Due to controller population resetting, a larger portion of the behavioural data is less similar to converged solutions, negatively affecting simulator specialisation. Another factor is that the controller population may often reset before convergence is achieved. Newly found convergence points could also be significantly different to the previous convergence point and the simulator would be

required to learn new behaviours.

5.5.4 Top Performers

The 64 tested adaptations are ordered in descending order of performance and the top 10 best performers are discussed in this section. Performance and transferability statistics of each tested adaptation is given in Tables B.9 and B.10, respectively. Performances are ordered based on the results of the Mann-Whitney U pairwise comparison tests. The best performing adaptation is identified as the HESEU adaptation. Tested adaptations are ordered based on the p-value obtained from the Mann-Whitney U test performed against the best adaptation.

The performance distributions of the top 10 best performing BNS adaptations tested are summarised in Table 5.18 and illustrated in Figure 5.14. The mean performances for the top 10 BNS adaptations are between 36.4 and 47.9 centimetres while the median performances are between 32.9 and 42.6 centimetres. The HESEU and HESET adaptations both use an **Ensemble** simulator configuration without simulator noise and use the simulator resetting procedure. The only difference between these two adaptations is that the HESEU adaptation uses the **Most Uncertain** sampling strategy while the HESET adaptation uses the **High Fitness** sampling strategy.

The 3^{rd} up to 7^{th} top adaptations all use the **Ensemble Multi-output** simulator configuration. The 8^{rd} best adaptation, HBSET, uses the **Basic** simulator configuration without simulator noise and the simulator resetting procedure. The last two top adaptations make use of the **Basic Multi-output** simulator configuration without simulator noise. The HSBET adaptation periodically resets both the controllers and simulator. The HSSET adaptation uses the **Basic Multi-output** simulator configuration without simulator noise and periodic simulator resetting.

The 1^{st} and 2^{nd} best adaptations are essentially the same adaptation except for the sampling strategy used. Similarly, the 3^{rd} and 4^{th} top adaptations only differ in terms of the sampling strategy chosen, and so does the 5^{th} and 6^{th} top adaptations. These groupings indicate that similar adaptations perform similarly well. Contrary to the overall comparison results, the **Ensemble** simulator configurations perform better than the **Ensemble Multi-output** simulator configurations.

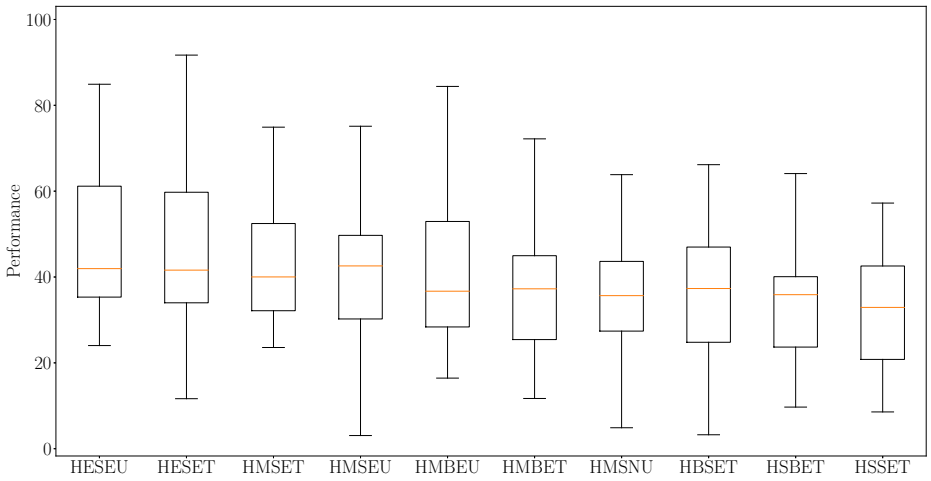


Figure 5.14: Performance distributions of the top 10 adaptations for the BNS approach

The first seven top adaptations use an ensemble approach. Half of the top adaptations use the **Ensemble Multi-output** simulator configuration. Only one adaptation uses the **Basic** simulator configuration. Two top adaptations have a **Basic Multi-output** simulator configuration.

All top 10 adaptations reset the simulator periodically. Only three of the top adaptation reset both the simulator and controller population. Resetting the simulator appears to be an important method for improving the performance of the BNS approach.

Nine of the top 10 best performing adaptations do not include simulator noise. Adding noise to the simulator appears to negatively affect the likely performance of solutions. The tested sampling strategies are almost evenly represented amongst the top adaptations and appear to have little significance.

The top adaptations are compared to one another using pairwise Mann-Whitney U tests. The p-values are given in Table 5.19. If the best performing adaptation (HESEU) is compared to all other adaptations, only HESET, HMSET, HMSEU and HMBEU are found to have a statistically equivalently performance distribution to the HESEU adaptation. The second best performer (HESET) has a significantly better performance distribution compared to the HSBET and HSSET adaptations. If the top two adaptations are excluded,

	Mean	Median	Q_1	Q_3	Std. Dev.
HESEU	*47.9*	42.0	*35.3*	*61.2*	17.1
HESET	45.9	41.6	34.0	59.7	18.0
HMSET	43.1	40.0	32.1	52.5	13.5
HMSEU	42.7	42.6	30.2	49.7	19.6
HMBEU	40.7	36.7	28.4	52.9	17.5
HMBET	37.0	37.3	25.4	45.0	15.1
HMSNU	36.8	35.7	27.4	43.6	17.3
HBSET	37.4	37.3	24.8	47.0	16.5
HSBET	36.4	35.9	23.7	40.1	19.0
HSSET	37.4	32.9	20.8	42.6	23.2

Table 5.18: Performance summary of the top 10 adaptations for the BNS approach

the remaining adaptations have statistically equivalent performance distributions to each other.

The transferability distributions for the best BNS adaptations are illustrated in Figure 5.15 and summarised in Table 5.20. The p-values for the Mann-Whitney U pairwise comparisons are given in Table 5.21. Amongst the top performers, the HESEU and HESET adaptations achieved the lowest median and IQR for their transferability distributions. The HESEU and HESET adaptations have significantly better transferability distributions compared to the HMBET, HBSET, HSBET and HSSET adaptations.

Ideal adaptations should have a low transferability metrics while also possessing high performance metrics. The HESEU adaptation has the best performance distribution while also demonstrating one of the best transferability distributions. The standard deviation of the transferability distributions is an additional factor to consider. A low standard deviation for the transferability distribution indicates that an adaptation is relatively consistent in terms of the reality gap between simulation and reality.

The HESEU adaptation achieved the lowest standard deviation amongst the best performing adaptations. A Levene's statistical test for assessing equality of variance is used to compare the variances between the HESEU adaptation and the other best performing

	HESEU	**HESET**	**HMSET**	**HMSEU**	**HMBEU**	**HMBET**	**HMSNU**	**HBSET**	**HSBET**
HESET	8.07×10^{-1}	-							
HMSET	2.90×10^{-1}	5.59×10^{-1}	-						
HMSEU	2.64×10^{-1}	5.40×10^{-1}	7.84×10^{-1}	-					
HMBEU	8.77×10^{-2}	2.40×10^{-1}	4.04×10^{-1}	6.84×10^{-1}	-				
HMBET	4.51×10^{-2}	6.15×10^{-2}	1.49×10^{-1}	2.52×10^{-1}	6.20×10^{-1}	-			
HMSNU	3.03×10^{-2}	5.94×10^{-2}	1.02×10^{-1}	2.23×10^{-1}	4.46×10^{-1}	8.88×10^{-1}	-		
HBSET	2.81×10^{-2}	9.05×10^{-2}	1.19×10^{-1}	2.64×10^{-1}	5.69×10^{-1}	7.84×10^{-1}	8.53×10^{-1}	-	
HSBET	2.32×10^{-2}	2.92×10^{-2}	7.98×10^{-2}	1.45×10^{-1}	3.40×10^{-1}	5.30×10^{-1}	6.73×10^{-1}	4.92×10^{-1}	-
HSSET	1.84×10^{-2}	3.39×10^{-2}	5.94×10^{-2}	9.63×10^{-2}	2.64×10^{-1}	5.69×10^{-1}	6.84×10^{-1}	4.92×10^{-1}	9.59×10^{-1}

Table 5.19: Performance comparisons between the top 10 adaptations for the BNS approach

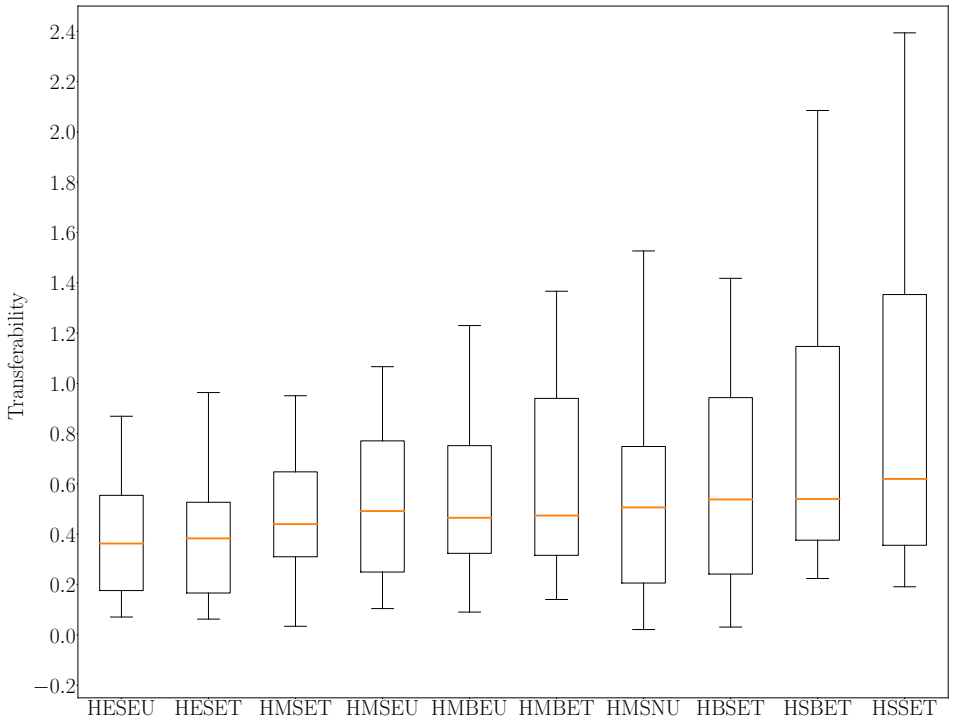


Figure 5.15: Transferability distributions for the top 10 BNS adaptations

adaptations. The p-values for the Levene's statistical test are given in Table 5.22. The variance between the HESEU adaptation and the HESET, HMSET, HMSEU, HMBEU, HMBET and HMSNU adaptations have statistically equal variances. The HBSET, HSBET and HSSET adaptations have significantly higher variances compared to the HESEU adaptation. The top adaptations configured in non-ensemble configurations have significantly higher variances compared to the HESEU adaptation.

5.5.5 Summary

This section summarises results obtained from the Simulated BNS Experimental work. The overall performance and transferability capabilities of the tested adaptation settings are studied and compared. When measuring the median performance and transferability of all simulator configurations, the **Ensemble Multi-output** and **Basic Multi-output**

	Mean	Median	Q_1	Q_3	Std. Dev.
HESEU	0.39	0.36	0.18	0.55	0.24
HESET	0.40	0.38	0.17	0.53	0.27
HMSET	0.50	0.44	0.31	0.65	0.28
HMSEU	0.96	0.49	0.25	0.77	2.35
HMBEU	0.61	0.47	0.32	0.75	0.53
HMBET	0.65	0.47	0.32	0.94	0.48
HMSNU	0.63	0.51	0.21	0.75	0.70
HBSET	0.70	0.54	0.24	0.94	0.65
HSBET	0.87	0.54	0.38	1.15	0.79
HSSET	0.97	0.62	0.36	1.35	0.91

Table 5.20: The transferability statistics for the top 10 adaptations for the BNS approach

configurations performed best overall while the **Ensemble Multi-output** and **Ensemble** configurations had the best overall transferability. Multi-output SNNs obtained the best performance properties while configuring SNNs into ensembles has the best transferability profiles.

Periodically resetting the simulator greatly improves both the likely performance and transferability of solution controllers. Not adding noise to controller evaluations during controller evolution improved both the performance and transferability of controllers compared to simulator configurations with noise. The sampling strategies do not significantly affect the likely performance or transferability profiles of adaptations.

The top performing adaptations from the Simulated BNS Experimental results are selected and studied. Many of the best adaptations consisted of either **Ensemble** or **Ensemble Multi-output** simulator configurations without noise. Most of the top adaptations use the simulator resetting procedure. Almost all the top performing adaptations make use of simulator noise.

	HESEU	**HESET**	**HMSET**	**HMSEU**	**HMBEU**	**HMBET**	**HMSNU**	**HBSET**	**HSBET**
HESET	9.12×10^{-1}	-							
HMSET	1.22×10^{-1}	1.41×10^{-1}	-						
HMSEU	9.05×10^{-2}	9.93×10^{-2}	7.51×10^{-1}	-					
HMBEU	9.93×10^{-2}	1.02×10^{-1}	8.88×10^{-1}	9.23×10^{-1}	-				
HMBET	3.64×10^{-2}	2.71×10^{-2}	5.30×10^{-1}	5.79×10^{-1}	6.10×10^{-1}	-			
HMSNU	2.58×10^{-1}	2.84×10^{-1}	9.12×10^{-1}	6.52×10^{-1}	6.63×10^{-1}	4.38×10^{-1}	-		
HBSET	3.51×10^{-2}	4.36×10^{-2}	4.83×10^{-1}	7.17×10^{-1}	6.95×10^{-1}	9.82×10^{-1}	4.46×10^{-1}	-	
HSBET	1.95×10^{-3}	1.95×10^{-3}	1.05×10^{-1}	1.19×10^{-1}	1.76×10^{-1}	2.64×10^{-1}	1.02×10^{-1}	2.97×10^{-1}	-
HSSET	9.52×10^{-4}	2.16×10^{-3}	4.84×10^{-2}	7.98×10^{-2}	8.50×10^{-2}	1.91×10^{-1}	5.75×10^{-2}	1.81×10^{-1}	7.84×10^{-1}

Table 5.21: Transferability comparisons between the top 10 adaptations for the BNS approach

	HESEU
HESET	9.35×10^{-1}
HMSET	7.29×10^{-1}
HMSEU	2.30×10^{-1}
HMBEU	9.34×10^{-2}
HMBET	5.78×10^{-2}
HMSNU	5.97×10^{-2}
HBSET	2.42×10^{-2}
HSBET	3.75×10^{-2}
HSSET	6.28×10^{-3}

Table 5.22: The p-values for transferability variance comparisons between the HESEU adaptation and the other top adaptations for the BNS approach

5.6 The BNS Validation Experiment Results

Results presented on the BNS approach so far have been completely simulated and have not yet been validated on a real-world Hexapod robot. Promising BNS adaptations discovered through the simulated experimental work are selected for real-world viability tests. Three of the top performing adaptations are selected based on the Simulated BNS Experimental results. The chosen adaptations are HESEU, HMSEU and HBSET. The HESEU adaptation is the best performing adaptation in the Simulated BNS Experimental results. The second best performing adaptation in the Simulated BNS Experimental results is the HESET adaptation but this adaptation is essentially the HESEU adaptation using a different sampling strategy. The third and fourth best performing adaptations (HMSET, HMSEU) in the simulated results are essentially equivalent adaptations but with different sampling strategies. No significant difference between the tested sampling strategies was found in the simulated results. The HMSEU adaptation is chosen for validation purposes. The next best adaptation from the simulated results that consisted of a different simulator configuration from the HESEU and HMSEU adaptations is the HBSET adaptation.

A limited number of real-world experiments are possible and the validation adaptations are carefully chosen based on better likely performance/transferability outcomes and in

order to validate a diverse set of adaptation settings. The chosen adaptations all use the simulator resetting procedure without noise. All chosen adaptations purposely use the simulator resetting procedure due to its excellent performance and transferability properties observed in the Simulated BNS Experimental results. Each adaptation utilises a different simulator configuration in order to validate many different simulator configurations. The HESEU and HMSEU adaptations use the **Ensemble** and **Ensemble Multi-output** simulator configurations, respectively. The HBSET adaptation uses a traditional **Basic** simulator configuration.

According to the ER literature, the addition of simulator noise should improve the likely transferability of solutions and result in better likely performance outcomes for solutions. However, simulated results indicated that, at least for the given robot task, not adding noise improved both the performance and transferability outcomes compared to adaptations that include noise. In order to validate that this is not the result of problems in the methodology chosen for the simulated experimental work (simulated inaccuracies), adaptations with and without simulator noise are validated.

Based on the chosen adaptations (HESEU, HMSEU and HBSET), the corresponding adaptations with simulator noise are also selected for real-world testing. The noise injected adaptations are HESNU, HMSNU and HBSNT. An additional advantage being that the HMSNU adaptation also happens to be one of the top 10 best performing adaptations discovered in the Simulated BNS Experiments. This is also the reason the HMSEU adaptation was chosen over the HMSET adaptation.

Each of the six chosen adaptations are investigated through five experimental trial runs of the BNS approach on the real-world Hexapod robot. The low number of samples per tested adaptation means that statistical comparisons between adaptations are unreliable. The validation experimental work is conducted for the primary purpose of demonstrating the viability of the BNS approach on a real-world Hexapod robot. Solution controllers are evaluated on the real-world Hexapod robot and the paths generated are recorded. Each experimental run can take between 3-4 hours to complete. Thirty experimental trial runs of the BNS approach are performed in total (5 trial runs per adaptation).

Performance statistics are collected and discussed in Section 5.6.1. The transferability properties of the tested adaptations are covered in Section 5.6.2. The paths generated

in simulation and reality are presented in Section 5.6.3 and lastly a summary is given in Section 5.6.4.

5.6.1 Performance

Performances achieved in trial runs are provided in Table B.5. Performance statistics for the validation experiments are provided in Table 5.23. The performance distributions of the tested adaptations are illustrated as standard box plots in Figure 5.16. The median performance achieve over all 30 trial runs is 50.8 centimetres with an IQR between 41.0 and 61.2 centimetres. The worst trial run has a solution with a performance metric of 20.6 centimetres. The second worst solution has a performance metric of 31.7 centimetres. Eighty percent of the 30 trial runs have a performance metric of 40 centimetres or greater. The HBSET, HBSNT and HESNU adaptations each contain a single outlier solution.

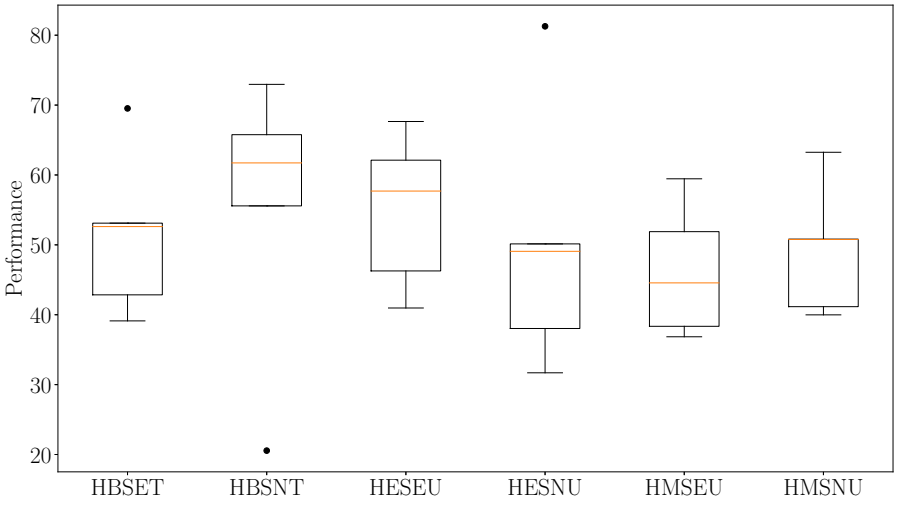


Figure 5.16: Validation experiment performance distributions

The HBSNT adaptation obtained the best performance IQR which is between 55.6 and 65.8 centimetres. The worst performing trial run also happens to be an outlier of the HBSNT adaptation. The second best IQR is observed for the HESEU adaptation solutions and is between 46.3 and 62.1 centimetres. The other adaptations have lower quartiles between 38.0 and 42.8 centimetres and upper quartiles are between 50.1 and

	Mean	Median	Q_1	Q_3	Std. Dev.
HBSET	51.4	52.6	42.8	53.1	11.8
HBSNT	55.3	61.7	55.6	65.8	20.4
HESEU	54.9	57.7	46.3	62.1	11.1
HESNU	50.0	49.1	38.0	50.1	19.1
HMSEU	46.2	44.6	38.3	51.9	9.5
HMSNU	49.2	50.8	41.2	50.8	9.4

Table 5.23: BNS performance statistics on validation results

53.1 centimetres. The HBSNT and HESNU adaptations have relatively high standard deviations largely due to outliers and the small sample sizes.

When comparing patterns seen in the validated and simulated results, it is important to take into account the small sample sizes of the validated results. Expected performance observations are confirmed, such as the validated HESEU performance being higher than the validated performance for the HBSET, HESNU, HMSEU and HMSNU adaptations. The validated HBSNT adaptation performs better than the HESEU adaptation which is not consistent with the simulated results. However, the small sample sizes of the validated adaptations is important to stress. If only two or three solution controllers had performed slightly better or worst between the validated HBSNT and HESEU adaptation results, the HBSNT adaptation could perform worse than the HESEU adaptation. The simulated standard deviation in performance outcomes is between 15 and 20 centimetres for most adaptations. The possibility of the HBSNT adaptation performing well by chance is relatively high.

The top 10 best performing adaptations from the Simulated BNS Experimental results have median performances between 32.9 and 42.6 centimetres. For the Validation BNS Experimental results, tested adaptations achieved median performances between 44.6 and 61.7 centimetres. The real-world validation experiment results perform better than expected compared to the completely simulated experimental investigations. The Simulated BNS Experimental work gives a relative indication of the likely performance differences between tested adaptations but does not perfectly represent reality. Differences between

the validated and simulated results may be due to simulated inaccuracies or high noise levels.

The validation results are grouped according to the inclusion or exclusion of simulator noise. The performance distributions for the noise injected and noiseless simulator configurations are illustrated in Figure 5.17 and summary statistics given in Table 5.24. The mean and median performance metrics are relatively close to each other. The performance distributions for the noise injected and noiseless simulator configurations are relatively symmetrical, however, the IQR and standard deviations appear different. The standard deviation for the noise injected and noiseless simulator configurations are 16.0 and 10.7 centimetres, respectively. However, sample sizes are too small to achieve a statistically significant comparisons when comparing variances. This large difference between the performance variance appears to be due to outliers.

According to the Simulated BNS Experimental work, noiseless adaptations are supposed to result in better likely performance outcomes. Except for the HESEU and HESNU adaptation validation results, the other validated adaptations indicate better performance outcomes when simulator noise is used. However, the sample sizes are too small to make statistical judgements based on the validation results.

	Mean	**Median**	$\mathbf{Q_1}$	$\mathbf{Q_3}$	**Standard Deviation**
Noiseless	50.9	51.9	41.9	58.6	10.7
Noise	51.5	50.8	40.6	62.5	16.0

Table 5.24: BNS validation performance statistics grouped by simulator noise

For the 30 validation solutions, 7 solution controllers are considered poor performers where performance measurements are less than 40 centimetres. Performances greater than 60 centimetres are considered excellent. Eight validation solution controllers demonstrated excellent performance outcomes. The remaining 15 validation solutions achieved acceptable performance values between 40 and 60 centimetres.

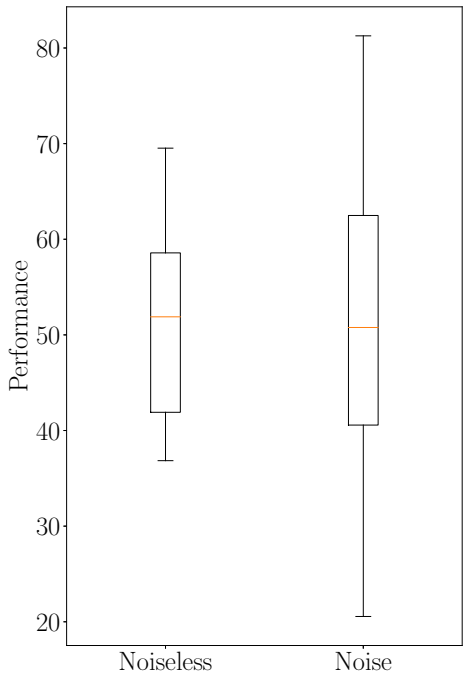


Figure 5.17: BNS validation performance distributions grouped by simulator noise

5.6.2 Transferability

Transferability values achieved in trial runs are provided in Table B.6. The transferability distributions for the validated BNS adaptations are illustrated in Figure 5.18 and summary statistics are given in Table 5.25. The transferability values for the tested adaptations is relatively low. Lower transferability values indicate a better correspondence between the real-world and simulated trajectories. The median transferability over all validated adaptations is 0.43 with an IQR between 0.23 and 0.56. The HBSET, HBSNT and HMSEU adaptations each consisted of a single outlier solution.

The HMSEU and HBSNT adaptations have the best median transferability. The IQR for the HMSEU adaptation is between 0.30 and 0.45 while the HBSNT adaptation is between 0.12 and 0.42. The HESEU adaptaton has a good lower quartile transferability of 0.13 but the median and upper quartile is relatively high at 0.47 and 0.66, respectively.

The HESNU and HMSNU adaptations have the worst transferability distribution.

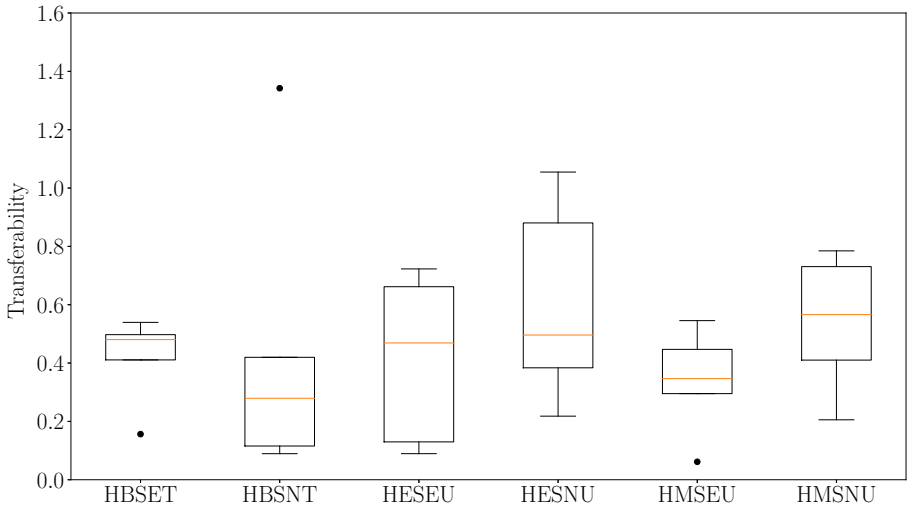


Figure 5.18: Validation experiment transferability distributions

The median transferability for the HESNU and HMSNU adaptations are 0.50 and 0.57, respectively. The IQR for the HESNU adaptation is between 0.38 and 0.88 while the HMSNU adaptation is between 0.41 and 0.73. Adding simulator noise to the solution search space probably leads to more varied behaviours and convergences over the lifetime of the BNS approach. The developing BNS simulator likely cannot keep up with the modelling of new behaviours and transferability is poorer.

The transferability distributions of the noise injected and noiseless simulator configurations are illustrated in Figure 5.19 and summary statistics given are in Table 5.26. The mean and median transferability for adaptations injecting simulator noise is 0.53 and 0.42, respectively. The mean and median transferability for noiseless adaptations is 0.39 and 0.45, respectively. The transferability distributions for the noise injected and noiseless simulator configurations are heavily skewed. Noiseless simulator configuration are skewed towards better transferability while the noise injected simulator configurations are skewed towards poor transferability. The standard deviation for the noise included and noiseless simulator configurations are 0.36 and 0.21, respectively.

According to the Simulated BNS Experimental work, noiseless adaptations are supposed to result in better likely transferability outcomes. Except for the HBSET and

Approach	Mean	Median	Q_1	Q_3	Std. Dev.
HBSET	0.42	0.48	0.41	0.50	0.15
HBSNT	0.45	0.28	0.12	0.42	0.52
HESEU	0.41	0.47	0.13	0.66	0.29
HESNU	0.61	0.50	0.38	0.88	0.35
HMSEU	0.34	0.35	0.30	0.45	0.18
HMSNU	0.54	0.57	0.41	0.73	0.24

Table 5.25: BNS transferability statistics on validation results

HBSNT adaptation validation results, the other validated adaptations confirmed that noiseless configurations improve the likely transferability outcomes of solutions. However, the sample sizes are too small to make statistical judgements based on the validation results.

	Mean	Median	Q_1	Q_3	Standard Deviation
Noiseless	0.39	0.45	0.23	0.52	0.21
Noise	0.53	0.42	0.25	0.76	0.36

Table 5.26: BNS validation transferability statistics grouped by simulator noise

5.6.3 Validation Solutions

The 30 controller solutions evolved during the validation experiments are described in greater detail in this section. The real-world and simulated paths generated by the evaluated solution controllers are presented in Figures 5.20 to 5.25. The robot starts at the origin facing the positive y-direction as represented by the arrow. The final simulated and real-world headings of the robot at the final positions in each trajectory are indicated with arrows.

The simulated paths generated by the BNS and SNS simulators are included. The two simulated paths are provided in order to compare the predictive accuracies between the BNS and SNS simulators for the given solutions. The simulated paths generated by the BNS simulator are usually a better indication of real-world performance compared

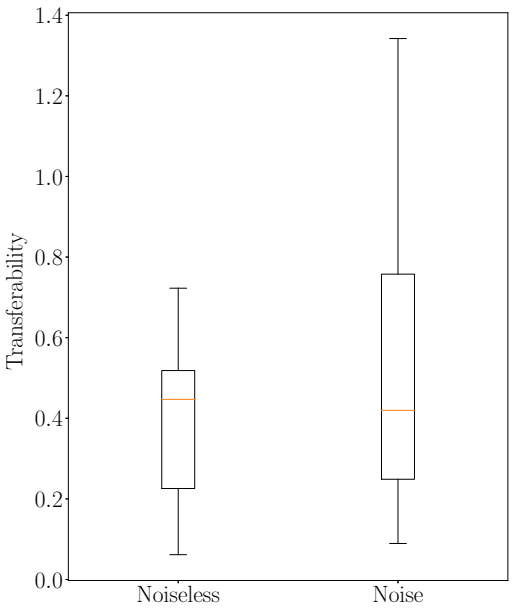
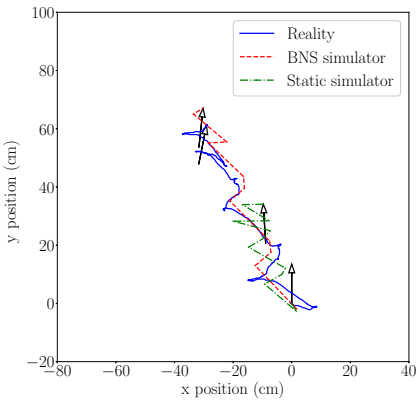


Figure 5.19: BNS validation transferability statistics grouped by simulator noise

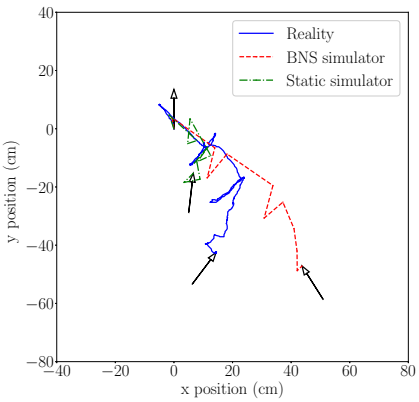
to the SNS simulator. The BNS simulator specialises in the modelling of those specific behaviours seen in the final controller solutions. Most solution controllers generate paths that travel a significantly forward/backward trajectory. Large movement towards the left or right are observed less often.

Solution controllers specific to the HESEU adaptation are illustrated in Figure 5.20. Evolved controller solutions successfully transfer well into reality. The first and fourth trial runs demonstrate solutions with particularly good transferability. The BNS simulated paths for second and fifth trial runs deviate slightly from reality due to inaccuracies in early heading predictions. The third trial run has the greatest disparity between the BNS simulated and real-world final positions. The SNS simulator performed poorly at predicting the behaviours of all the HESEU solutions.

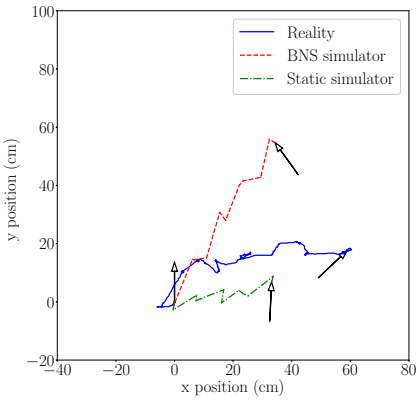
Solutions specific to the HESNU adaptation are presented in Figure 5.21. The first three trial runs demonstrate relatively good transferability. The first and second controller solutions travel a significantly further distances compared to the other HESNU adaptation solutions. The distance travelled in the third run is relatively small. The fourth and



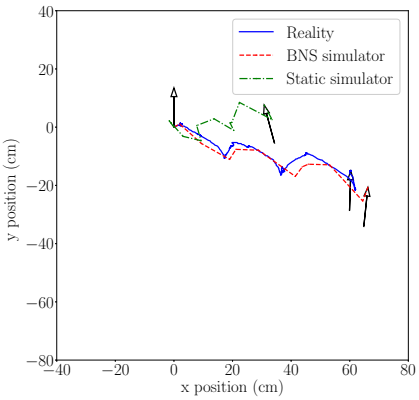
(a) First run



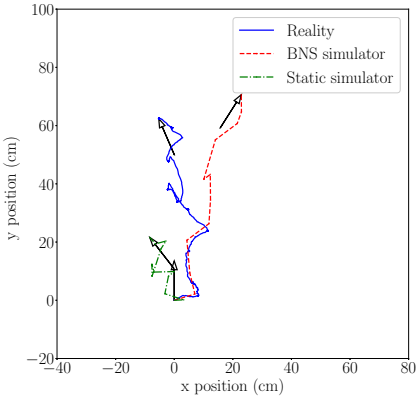
(b) Second Run



(c) Third run

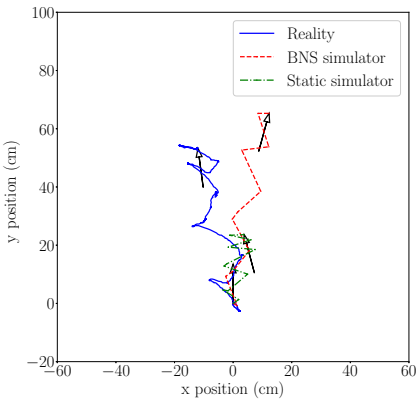


(d) Fourth run

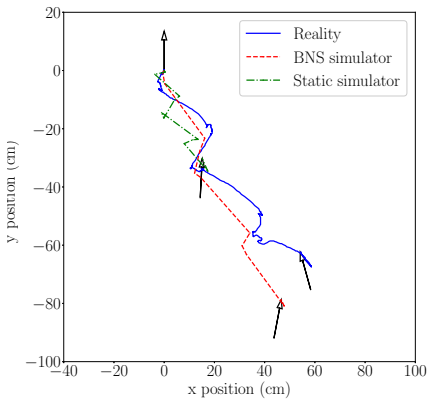


(e) Fifth run

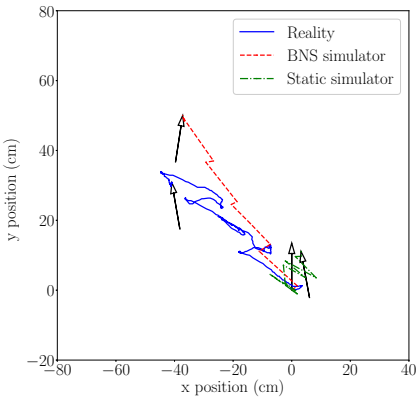
Figure 5.20: HESEU Real-world experiments



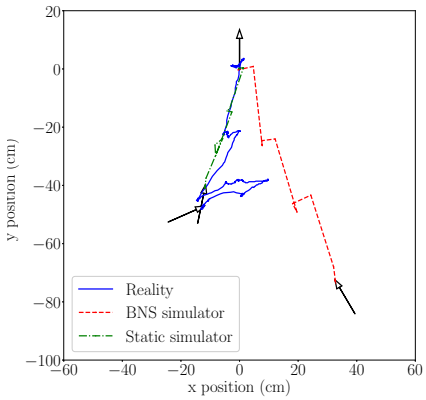
(a) First run



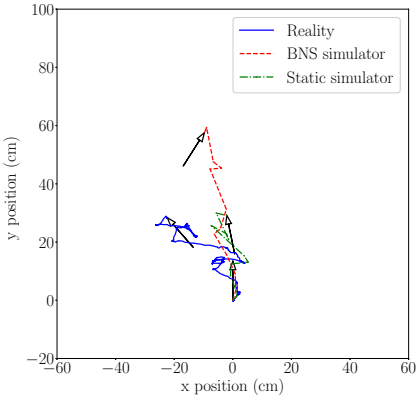
(b) Second Run



(c) Third run

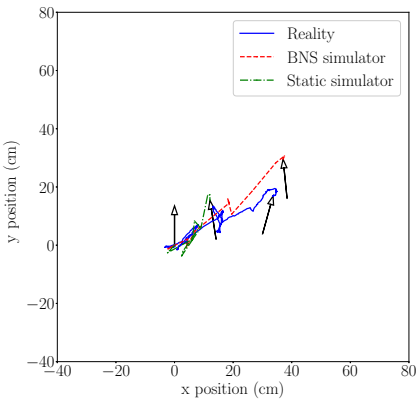


(d) Fourth run

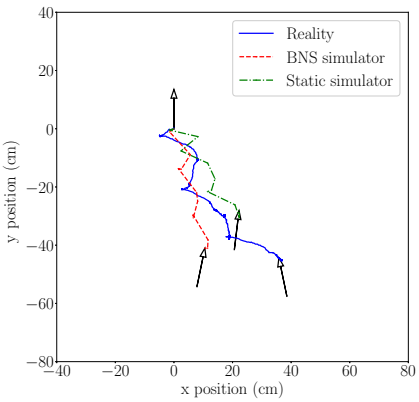


(e) Fifth run

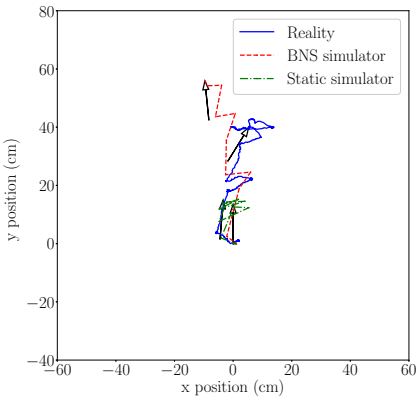
Figure 5.21: HESNU Real-world experiments



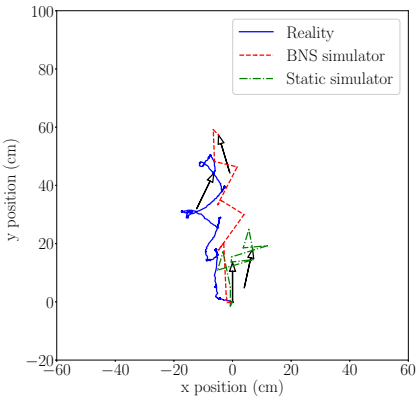
(a) First run



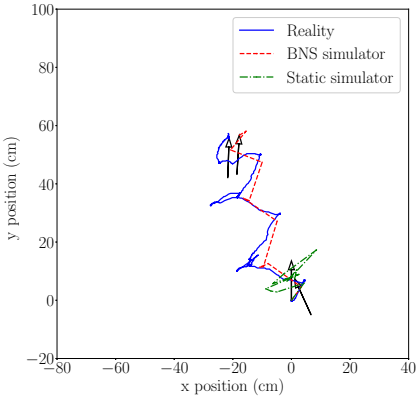
(b) Second Run



(c) Third run

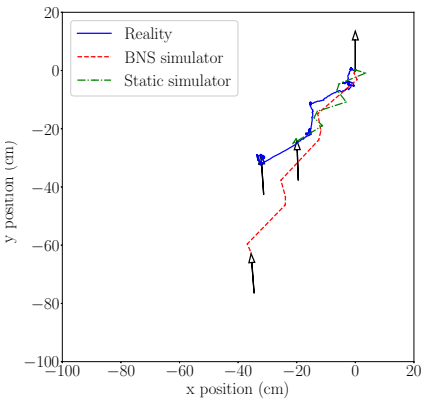


(d) Fourth run

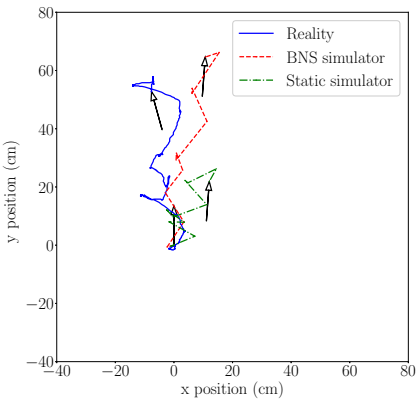


(e) Fifth run

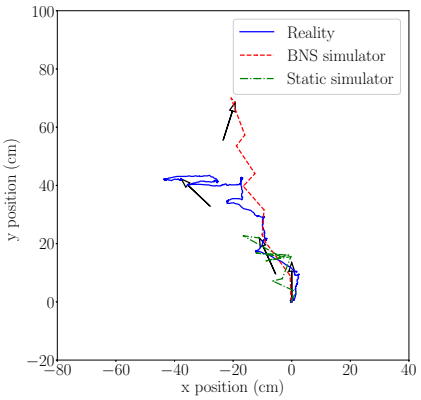
Figure 5.22: HMSEU Real-world experiments



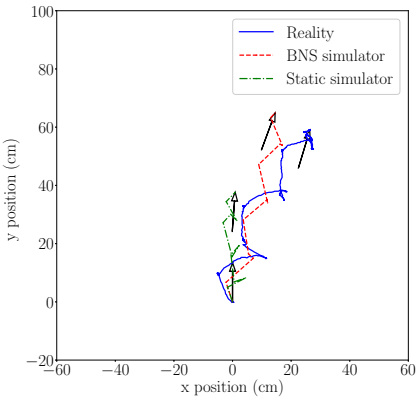
(a) First run



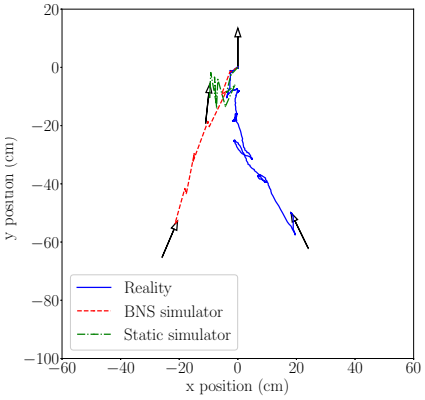
(b) Second Run



(c) Third run

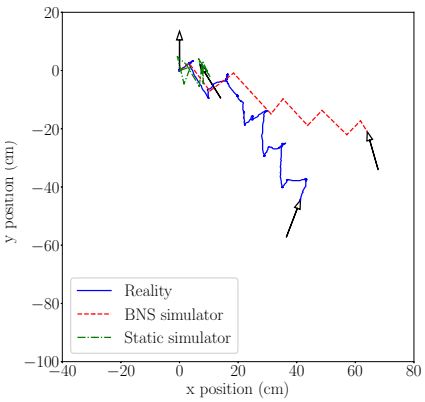


(d) Fourth run

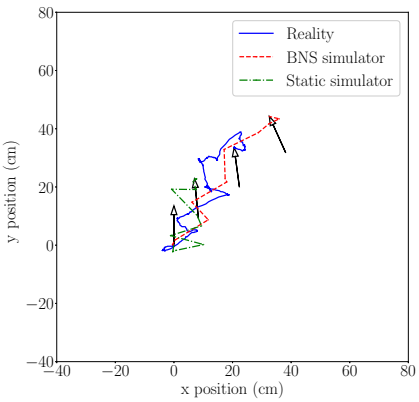


(e) Fifth run

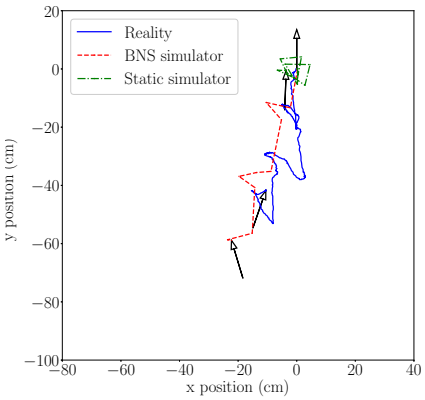
Figure 5.23: HMSNU Real-world experiments



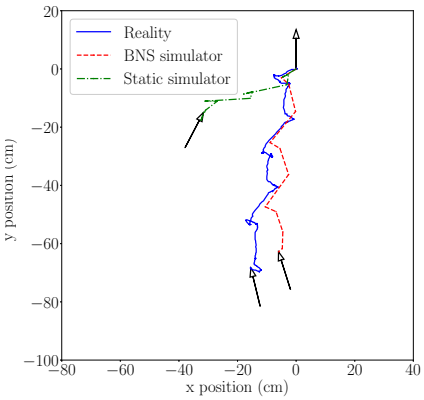
(a) First run



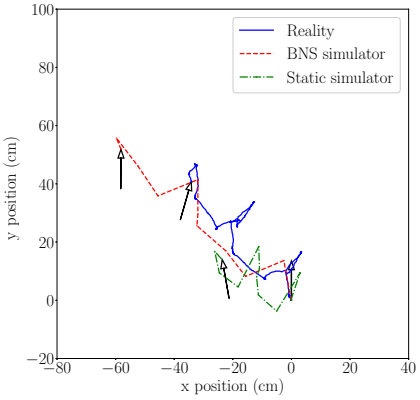
(b) Second Run



(c) Third run

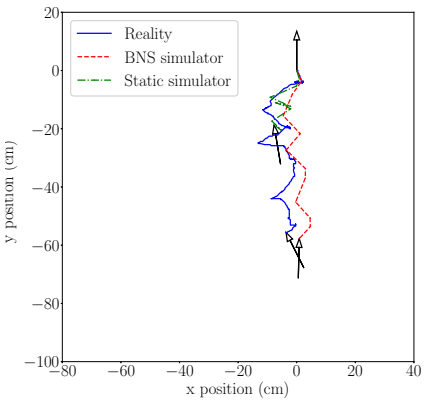


(d) Fourth run

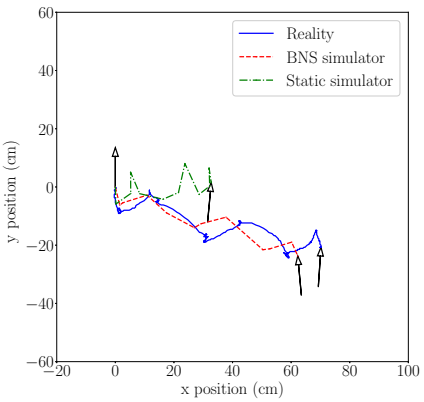


(e) Fifth run

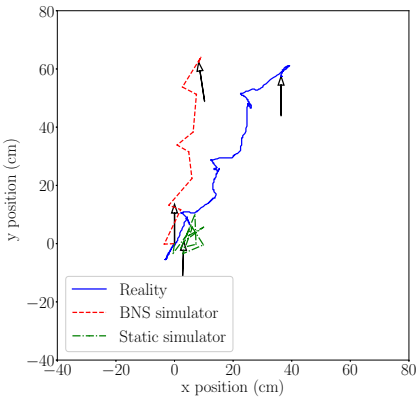
Figure 5.24: HBSET Real-world experiments



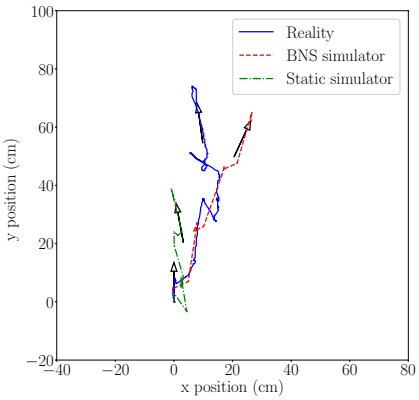
(a) First run



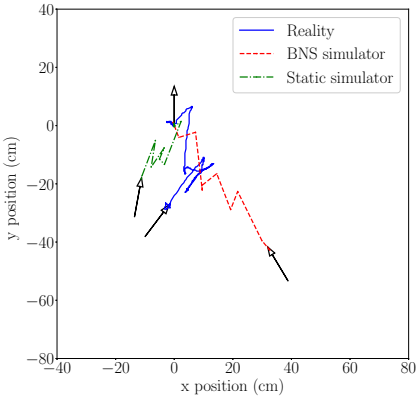
(b) Second Run



(c) Third run



(d) Fourth run



(e) Fifth run

Figure 5.25: HBSNT Real-world experiments

fifth trial runs transfer poorly into reality. For the SNS simulator, the first and second trial runs are poorly simulated in terms of the real-world distances travelled, however, the simulated direction of the paths are reasonably accurate. For the third run, the SNS simulated distance is significantly underestimated. The fourth and fifth solutions have SNS simulated paths that appear to more accurately represent reality than the BNS simulated paths.

Figure 5.22 present controller solution paths for the HMSEU adaptation. All solutions demonstrate good transferability into reality. The first trial run transfers well into reality but the distance travelled is relatively low. For the second trial run, the BNS simulator underestimates the real-world distance travelled but the simulated heading and shape are reasonably close to reality. The third trial run is simulated well for the first two-thirds of the BNS simulated path, after which the last few commands transfer poorly onto reality. The fourth and fifth trial runs demonstrate excellent performance, transferability and even small variations in behaviours are accurately predicted by the BNS simulator. The SNS approach significantly underestimates the real-world distances travelled but the simulated headings are mostly accurate.

Solution paths for the HMSNU adaptation are illustrated in Figure 5.23. The second, fourth and fifth trial runs demonstrate excellent performance in terms of distance travelled. For the BNS simulator, the simulated paths for the second and fourth trial runs closely correspond to reality. The fifth run is accurately simulated in term of distance travelled and overall behaviour, however, the simulated direction of the trajectory deviates substantially from reality. The deviation is due to inaccuracies in simulator predictions during the early part of the controller evaluation. The first and third trial runs travel significantly shorter distances in reality compared to the BNS simulated predictions. The BNS simulated path for the third trial run is relatively accurate for the first few commands but the real-world path deviates for the last few commands. The SNS approach greatly underestimates the real-world distances travelled for all trial runs but does often simulate the trajectory headings relatively well.

For the HBSET adaptation, paths for solution controllers (Figure 5.24) demonstrate good transferability for all trial runs. The first and fourth trial runs are accurately simulated (BNS simulator) in terms of the real-world distances travelled. The simulated

direction of the trajectory for the first trial run deviates slightly from reality but the zigzag behaviours are simulated relatively well. The third trial run demonstrates good transferability but the BNS simulator slightly overestimates the real-world distance travelled. The fifth trial run is simulated well in terms of the overall behaviour and heading but the simulated distance is overestimated significantly. The SNS simulator failed to accurately simulate the real-world behaviours for any of the solution controllers.

Solution paths for the HBSNT adaptation are presented in Figure 5.25. The BNS simulator demonstrates great transferability for the first four trial runs. The BNS simulator accurately estimates many low level behaviours for the first, second and third trial runs. For the second trial run, the BNS simulator slightly underestimates the real-world distance travelled. The fourth trial solution transfers well into reality but a few commands do appear to transfer poorly. The fifth trial run transfers poorly into reality due to a significant change in the real-world trajectory heading. The SNS simulated paths demonstrate poor transferability compared to the BNS simulator for all solutions, except the fifth trial run. Only the fifth trial run is simulated more accurately using the SNS simulator. The SNS simulator predicted most of the trajectory headings relatively well, however, simulated distances are always underestimated.

5.6.4 Summary

Six promising adaptations were validated on a real-world Hexapod robot. Adaptations that performed well in simulated benchmarks are validated on a real-world robot in order to verify real-world viability of the BNS approach. For each adaptation, five BNS trial runs are performed on the real-world Hexapod robot and solution trajectories are recorded.

The best performing and most transferable BNS adaptation validated used the **Basic** simulator configuration with noise, simulator resetting and the **Tournament** sampling strategy (HBSNT). A close second best performing adaptation, used a noiseless **Ensemble** simulator configuration with simulator resetting and the **Most Uncertain** sampling strategy (HESEU).

Most of the validated solutions demonstrated excellent transference of simulated behaviours into reality. Only 7 out of the 30 validated solutions demonstrated poor performance. This conclusively demonstrates that the tested BNS adaptations are viable in

reality.

The Dynamic Simulators developed using the BNS approach are specialised towards simulating particular behaviours for a single solution, whereas Static Simulators are generalised for use on multiple problems and/or behaviours. Solutions developed using the BNS approach tend to be poorly simulated using the simulator developed by the SNS approach. BNS developed simulators could more accurately predict the real-world behaviours of evolved solutions compared to SNS developed simulators. The Static Simulators are unable to adequately predict the behaviours of solution controllers developed using the BNS approach. This indicates that solution controllers evolved using the BNS approach are unlikely have been discovered using the SNS approach.

5.7 SNS and BNS Comparisons

The behavioural dataset used to train the Static Simulators for the SNS approach consists of 4942 behavioural patterns. A single experimental run of the BNS approach (100 sampling controller evaluations) consists of 1100 behavioural patterns. The BNS approach developed adequate controller solutions with significantly fewer behavioural patterns compared to the SNS approach.

The SNS approach requires a lengthy data collection process. It takes approximately 16 hours to collect the amount of behavioural data used to train the Static SNNs for the Hexapod robot. Once the behavioural data collection phase is completed, training any of the tested simulator configurations takes less than 15 minutes. Data collection and training process only need to be performed once, after which the produced Static SNNs can be reused. Once the Static Simulator is trained, producing a solution for the SNS approach takes between 1.5 and 15 minutes, depending on the adaptation used.

A trial run of the BNS approach is approximately 3.5 hours when including rest periods. An adequate solution is often discovered in less than 100 controller sampling evaluations. For the BNS approach, simulator training and controller evolution is performed concurrently with the behavioural data collection process. In order to produce the first solution, the BNS approach can take 3.5 hours while the SNS approach takes approximately 16.5 hours.

The best performing BNS adaptation in the validation experimental work achieved a median performance of 61.7 centimetres with an IQR between 55.6 and 65.8 centimetres. The best performing adaptation for the SNS approach achieved a median performance of 60.2 centimetres with an IQR between 54.4 and 66.8 centimetres. The best SNS adaptation performs similarly well compared to the best performing validated BNS adaptation. A statistical comparison between the SNS and BNS approaches is not possible due to the small sample sizes of the validated BNS adaptation results. However, experimental outcomes appear to indicate that the BNS approach can at least match the performances outcomes achieved by the SNS approach.

The best performing SNS adaptation has a similar transferability compared to all the tested BNS adaptations. The BNS validated solutions have a median transferability of 0.43 while the HME adaptation achieved a median transferability of 0.42. In terms of the diversity of solutions produced, the SNS approach is greatly affected by the chosen adaptation. Independent trial runs of the BNS approach can produce a diverse set of solutions no matter which adaptation is chosen.

5.8 Conclusion

This research is the first time that the BNS approach has been validated on a Hexapod robot. This chapter successfully demonstrates that the BNS approach can develop effective Hexapod gaits that can travel significant distances without real-time position or sensor feedback. The controller design is high dimensional without relying on prior knowledge of Hexapod locomotion modes. Developed simulators can accurately model the behaviours of a Hexapod robot based on prior observations and little specialised human knowledge. The BNS approach can develop effective solutions without the lengthy data collection phase seen with the SNS approach.

Many novel BNS adaptations are proposed and tested. Configuring SNNs into ensembles improved the likely performance and transferability outcomes of solution controllers. The **Ensemble** simulator configuration performed particularly well in simulated and validation results. Not using simulator noise might improve the likely performance outcomes of solutions. However, validation results did not confirm that noiseless adaptations always

lead to improved performance results. Periodically resetting the simulator significantly improves the likely performance and transferability outcomes of solutions. Simulator resetting appears to reduce the likelihood that the ER process will exploit weaknesses in simulated behaviours. The sampling strategies tested do not significantly influence performance outcomes.

Chapter 6

SNAKE STATIC NEURO-SIMULATION

6.1 Introduction

Prior research has demonstrated that the SNS approach is viable for a Snake-like robot morphology [Woodford *et al.*, 2015]. However, for a Snake robot, no prior work has investigated the SNS approach for a controller design that relies on little prior knowledge. This chapter presents a first time investigation demonstrating that the SNS approach is viable for a Snake robot without a simplified controller design. The experimental procedures used to investigate the proposed SNS adaptations are discussed in Section 6.2.

The Snake robot simulator investigated in prior work was simplified to the point of not being compatible with the controller design chosen in this work. A new generalised Snake robot simulator needs to be built. However, the ideal SNN architectures that specify the number of hidden layers and layer sizes has not been identified. Behavioural data is collected from a real-world Snake robot and used to benchmark a large number of potential SNN architectures in order to identify ideal SNN architecture (Section 6.3). The SNN architecture benchmarking results are presented in Section 6.4.

The SNS experimental work investigating proposed adaptations is described in Section 6.5. Experimental results are presented and discussed in Section 6.6. The SNS experimental outcomes between the Hexapod and Snake robots are compared to each other and

discussed in Section 6.7. Finally, conclusions to the chapter are drawn in Section 6.8.

6.2 Experimental Procedure

Methodology A is the experimental procedure followed in this chapter (Figure 6.1). Methodology A is applied similarly to the Hexapod experiment work and is discussed in greater detail in Section 3.6.

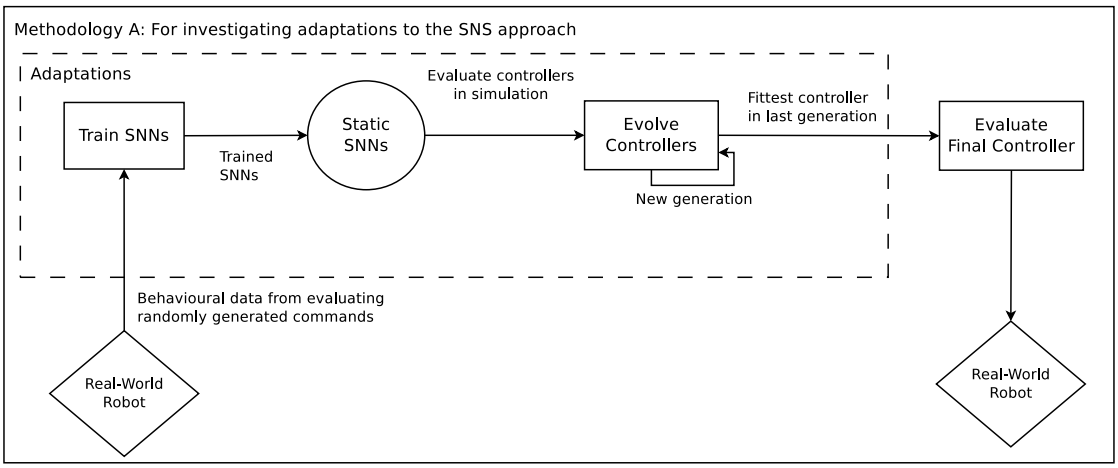


Figure 6.1: Methodology A: Adaptations to the SNS approach

The Snake robot controller design is discussed in Section 6.2.1 while the Snake robot simulator is covered in Section 6.2.2.

6.2.1 Controllers

The Snake robot controller design is similar to that described in Section 3.3. The controller design is slightly modified to accommodates the Snake robot platform. Controllers are optimised for developing crawling gaits where the Euclidean distance between the start and final positions of the robot are maximised. The positions measured are calculated as the midpoint of the tracked markers located on the head of the Snake robot. Prior work positioned tracked points on either end of the Snake robot's body. An advantage of the Snake robot tracking approach explored in this research is that only a single tracked position is simulated. The prior tracking method required simulating two tracked positions for the Snake robot [Woodford *et al.*, 2015]. No real-time feedback is provided to controllers

regarding its current position or state during evaluations.

Due to limitations of the tracking system, a Snake robot cannot be tracked if the robot turns upside down during controller evaluations. Evaluations where the robot turns upside-down for more than two commands are categorised as failures. In order to evolve controllers exhibiting relatively upright behaviours, the robot's head orientation is simulated and failures are penalised in the fitness function.

A controller consists of a sequential list of thirteen commands. Each command consist of 12 joint angle changes, one for each joint on the robot. Each joint angle change is an integer value between -100 and 100 units which translates to angle changes between -30 to 30 degrees. Joints have a resolution of approximately 0.3 degrees. Lateral joints have a movement range between -60 and 60 degrees relative to the starting position. Vertical joint angles have a range between -30 and 30 degrees relative to the starting positions. The vertical joint position range is intentionally smaller than the lateral range in order to reduce the likelihood of the robot exhibiting behaviours that cause it to roll onto its back. The difference in vertical and lateral freedom of movement also helps reduce joint failures caused by high joint torques.

For a controller, the sequence of 13 commands is repeated multiple times in order to produce cyclical crawling behaviours. The Snake robot is initially positioned with straight joint angles. The first command positions the joint angles into an initial stance. Thirteen commands are then executed sequentially three times to form cycles. The robot returns to an initial stance at the end of each cycle. Forty commands are executed in total for any given Snake robot controller evaluation solution. The Snake robot controllers require significantly more commands compared to the Hexapod robot in order to traverse similar distances.

If simulator noise is present, controllers are evaluated five times in simulation with noise added to each behavioural component prediction. The average fitness from the five evaluations becomes the controller fitness. The noise distributions are Gaussian with a zero mean and the standard deviations are equal to the test dataset error standard deviations. Test dataset error standard deviations are 3.03 centimetres for Δx, 2.00 centimetres for Δy, 7.17 degrees for Δa and 15.49 degrees for Δo.

6.2.2 Simulator

An orientation sensor is placed on the head of the Snake robot. The head orientation state of a robot has been simulated as a yaw, pitch and roll [De Nardi and Holland, 2008]. However, this requires simulating multiple behavioural components which may be unnecessary. This research opted to simulated the head orientation as a single behavioural component. The orientation is measured as the angle between the normal of the working surface and the normal on the head of the robot. This overall change in the robot's orientation is denoted as Δo (Figure 3.4).

Unlike the Hexapod robot, Snake controller commands cannot be independently evaluated. The behavioural outcome of the first command depends on the starting head orientation. Subsequent commands depend on the outcome of all previous head orientation predictions. This can slow down the evaluation process and be a source of accumulated inaccuracies.

The starting head orientation of the first command, θ_1, is always zero because the robot's starting position is straight and upright. The first command is taken as input to the SNNs which produces simulated behavioural component displacements as output. The simulated change in the head orientation is used to calculate the new head orientation, θ_2. The second command can only be evaluated once the new head orientation has been predicted. This sequential evaluation process continues until all commands have been evaluated.

6.3 Simulator Benchmark Experiments

The Simulator Benchmarking Experiments investigate a large number of proposed SNN architectures. SNN architectures tested can consist of between one and five hidden layers. Hidden layers can have either 25, 100, 400 or 800 Artificial Neurons. The total number of possible SNN architectures tested is 1364 combinations per behavioural component simulated. Every possible SNN architecture is independently initialised and trained thirty times in order to perform statistical comparisons. Only the best performing architectures are presented but summarised test data can be found in Appendix B. The best performing SNN architectures are those that achieve the lowest average error on the unseen test

dataset. For each behavioural component simulated, the best SNN architectures have statistically the lowest test dataset errors compared to all other architectures. SNN architectures with the lowest number of hidden layers or with the smaller hidden layer sizes are selected when multiple SNN architectures perform statistically equally the best.

Training is performed using the Adam optimiser and the Deep Learning libraries called Keras [Chollet, 2015] and Tensorflow [Abadi *et al.*, 2015]. SNN weights are saved after every training iteration whenever the validation error is improved. SNNs are trained for 4000 iterations. Only weights associated to the lowest validation error are used in the final models in order to avoid problems related to over-fitting. During training, hidden layer nodes have a dropout rate of 50%.

6.4 Static SNN Results

This section discusses the identified ideal SNN architectures and the training of the SNNs based on the collected behavioural data, which is required before controllers can be evolved during the SNS approach. Section 6.4.1 presents an analysis of the collected behavioural data. The ideal SNN architectures are identified and presented in Section 6.4.2. Once the ideal SNNs have been trained, the accuracy of the proposed simulator configurations are discussed in Section 6.4.3.

6.4.1 Behavioural Data

Behavioural observations collected during the data collection phase are analysed in this section. A total number of 4990 behavioural patterns are collected from randomly generated commands that are evaluated on the real-world Snake robot. Ten percent of this data forms the test dataset, another 10% forms the validation set used to prevent over-fitting and the remaining 80% forms the training dataset.

Position displacements for the training dataset are illustrated in Figure 6.2. Each point represents the relative displacement of the tracked position on the head of the robot after evaluating a single command. The x and y displacements are the relative changes in the Snake robot's position due to the execution of the given command. The displacement trends are significantly different compare to the Hexapod robot covered in Section 4.4.2.

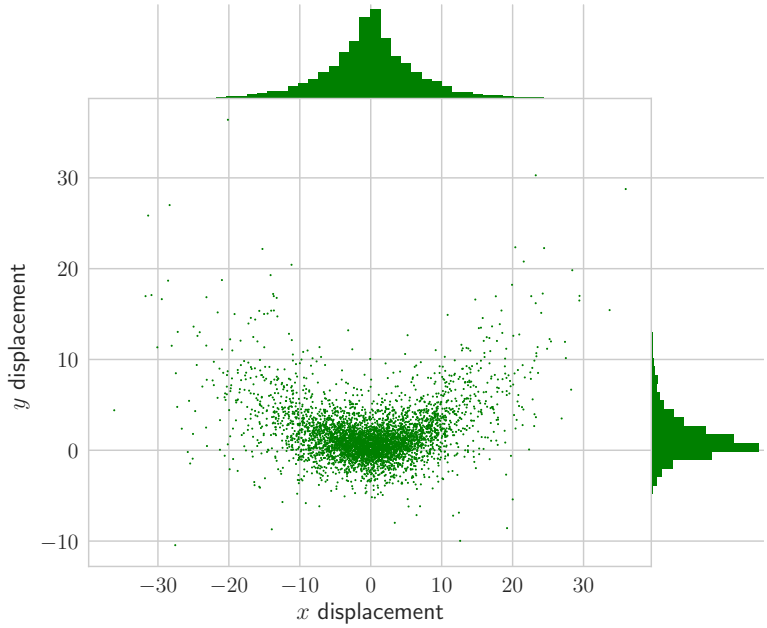


Figure 6.2: Scatter plot for the training data Δx and Δy behavioural components

Displacements appear to have a parabolic shape. There is a significant bias towards moving in the positive y direction. Few observation are found moving significant distances in the positive or negative y directions without comparable x displacement. The parabola shape is likely due to the body of the Snake robot weighing more than the tracked head. The head tends to get pulled towards the body during movements.

Density plots for the robots x and y displacements are given in Figure 6.3. The IQR for the x displacements are between -3.65 and 3.43 centimetres. The IQR for the y displacements are between -0.01 and 2.49 centimetres which is greatly skewed towards positive displacements. The Snake head is unlikely to move in the negative y direction due to the heavy Snake body. A large proportion of commands generate x and y displacements close to zero. The robot has greater mobility in moving the head in a side-ways trajectory, rather than directly forward/backward. The density decreases for highly positive or negative displacement values. The training data is imbalanced and trained models are likely more accurate for predictions close to the origin.

Density plots for the robots heading (a displacements) and head orientation (o displacements) training datasets are given in Figure 6.4. The IQR for the heading displacements

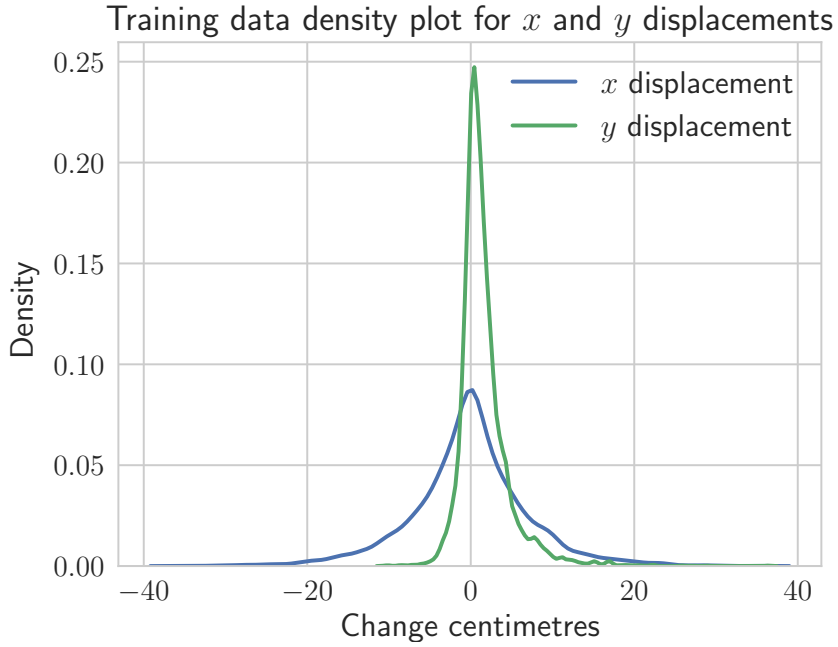


Figure 6.3: Density plot for the Δx and Δy behavioural components

are between -13.0 and 12.0 degrees. The IQR for the head orientation displacements are between -13.6 and 12.25 degrees. The heading displacement density plot is relatively symmetrical and normal. However, the head orientation displacement density plot appears to be non-symmetrical. This indicates that the Snake robot head orientations are slightly biased towards falling sooner on one side of the robot's body compared to the other. The weight distribution for the robot's body is not perfectly balanced.

6.4.2 Ideal SNN Architectures

Test error results for all the tested SNN architectures are referenced in Appendix B. The ideal SNN architectures are presented in Table 6.1. The ideal number of SNN hidden layers was found to be three for all behavioural components. When identifying the ideal SNN architecture for each behavioural component, comparisons between candidate SNN architectures are performed using Mann-Whitney U tests. A significance level of 5% is chosen for all comparisons. For each behavioural component, the SNN architecture with the best test dataset accuracy is compared to all other configurations. SNN architectures that are not significantly worse compared to the most accurate SNN architecture are iden-

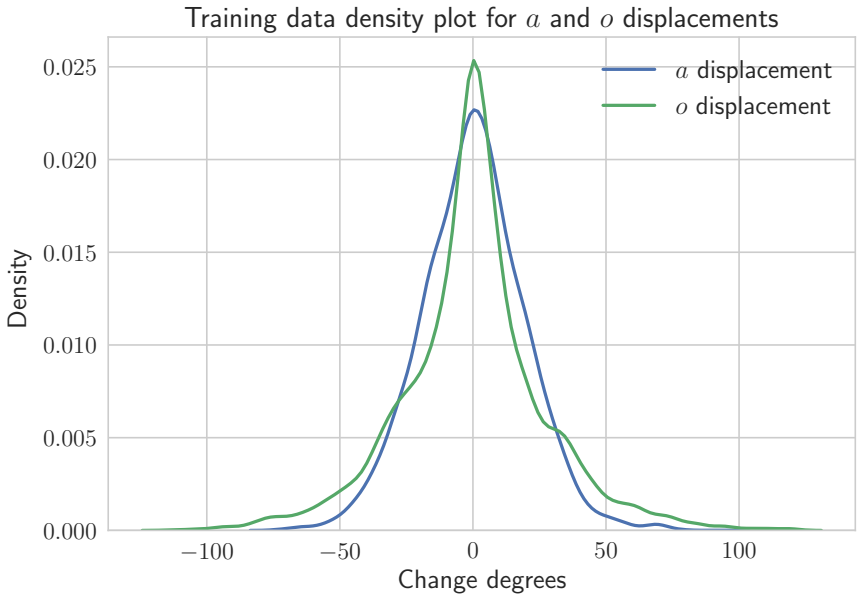


Figure 6.4: Density plot for the Δa and Δo behavioural components

tified. If multiple SNN architectures perform statistically similar to the best architecture then the SNN architecture with the lowest number of hidden layers or Artificial Neurons is declared the ideal SNN architecture.

FFNN component	**First hidden layer**	**Second hidden layer**	**Third hidden layer**
Δx	800	800	100
Δy	800	800	25
Δa	800	800	100
Δo	800	800	25

Table 6.1: SNN architectures

SNN architectures with the best accuracy for modelling the x and o displacements are significantly more accurate than all other tested architectures. The most accurate SNN architecture for predicting y displacements is statistically equal to two other SNN architectures tested. One of these architectures has four hidden layers while the other

two only have three hidden layers. Four hidden layers is considered more computationally expensive, so the 3 layer configuration is selected. The remaining y displacement candidate architectures all have 800 Artificial Neurons for the first two hidden layers. The last hidden layer for one architecture has 100 Artificial Neurons while the chosen ideal architecture only has 25.

The best SNN architecture for modelling the a displacements performs similarly well to one other SNN architecture. Both candidate SNN architectures have 800 Artificial Neurons for the first two hidden layers. The third hidden layer has 400 Artificial Neurons for the one architecture while the chosen ideal SNN architecture has 100 Artificial Neurons.

6.4.3 Training Errors

The ideal SNN architectures are used to construct the simulator configurations specified in Section 3.7.1. For each ideal simulator configuration, MSE and the IQR for the squared errors of the training and test behavioural datasets are presented in Tables 6.2 and 6.3, respectively. The first and third quartiles of the squared errors are given by the columns labelled Q_1, and Q_3, respectively. The lowest Δx training MSE and IQR is achieved by the **Dropout** simulator configuration, followed closely by the **Basic** and **Ensemble** configurations. The **Ensemble Multi-output** and **Basic Multi-output** simulator configurations have MSE values more than four times higher than the other configurations.

For the y displacements, the **Dropout**, **Basic** and **Ensemble** simulator configurations have the best training dataset accuracy. However, the **Ensemble** simulator configuration has the lowest training IQR of squared errors. The **Ensemble Multi-output** and **Basic Multi-output** simulator configurations have MSE values more than 2.5 times higher than the other configurations.

The **Dropout** simulator configuration has the lowest Δa training dataset MSE and IQR of squared errors, followed closely by the **Basic** simulator configuration. The **Ensemble Multi-output** and **Basic Multi-output** simulator configurations have Δa MSE values over four times higher than other simulator configuration.

The **Ensemble Multi-output** and **Basic Multi-output** simulator configurations have Δo training dataset MSE values more than 4.5 times higher than all other simulator configurations. The **Dropout** simulator configuration has the lowest Δo training dataset

Training Dataset	Δx			Δy			Δa			Δo		
	MSE	Q_1	Q_3	**MSE**	Q_1	Q_3	**MSE**	Q_1	Q_3	**MSE**	Q_1	Q_3
Basic	1.64	0.05	1.13	0.77	0.04	0.58	7.09	0.16	3.12	40.16	0.96	24.66
Dropout	*1.34*	0.05	0.75	*0.74*	0.03	0.49	*6.20*	0.14	2.14	*37.87*	0.96	16.59
Ensemble	1.60	0.03	1.18	0.80	0.02	0.36	9.20	0.23	7.37	41.01	0.81	28.68
Ensemble Multi-output	7.44	0.36	6.45	2.03	0.11	1.63	36.88	2.40	34.44	197.26	8.47	160.11
Basic Multi-output	8.63	0.43	7.78	2.26	0.12	1.83	43.09	2.86	41.17	227.61	10.12	188.66

Table 6.2: Training dataset MSE and IQR for each simulator configuration

Test Dataset	Δx			Δy			Δa			Δo		
	MSE	Q_1	Q_3	**MSE**	Q_1	Q_3	**MSE**	Q_1	Q_3	**MSE**	Q_1	Q_3
Basic	10.08	0.30	7.00	4.24	0.16	2.83	59.63	2.53	41.05	382.50	10.43	205.25
Dropout	*9.74*	0.32	6.70	4.12	0.15	2.75	*58.82*	2.08	38.24	382.16	10.94	210.43
Ensemble	10.00	0.38	7.19	*4.03*	0.14	2.56	59.26	2.67	39.57	*357.34*	9.30	207.66
Ensemble Multi-output	10.92	0.47	7.79	4.06	0.16	2.56	62.75	3.23	50.61	421.53	15.75	245.05
Basic Multi-output	12.22	0.54	9.43	4.34	0.18	2.56	69.40	3.43	58.48	440.55	15.78	281.36

Table 6.3: Test dataset MSE and IQR for each simulator configuration

MSE and IQR of squared errors, followed closely by the **Basic** and **Ensemble** simulator configurations.

The **Dropout** simulator configuration has the lowest Δx MSE and IQR of squared errors for the test dataset, followed closely by the **Basic**, **Ensemble** and **Ensemble Multi-output** simulator configurations. The **Basic Multi-output** simulator configuration has the worst Δx MSE and IQR of squared errors. However, differences between configurations are less pronounced compared to the training dataset.

For the Δy behavioural component, the test dataset MSE and IQR of squared errors for all simulator configurations are relatively similar to each other. The test dataset MSEs for the different simulator configurations are all within 8% of each other. The **Ensemble** simulator configuration has the lowest MSE followed closely by the **Ensemble Multi-output** simulator configuration.

The **Dropout**, **Ensemble** and **Basic** simulator configurations have similarly the lowest Δa test dataset MSE and IQR for squared errors. For the test dataset Δa behavioural component, the test dataset MSE and IQR of squared errors is highest for the **Basic Multi-output** and **Ensemble Multi-output** simulator configurations.

The **Ensemble** simulator configuration has the lowest test dataset Δo behavioural component MSE and IQR of squared errors, followed closely by the **Basic** and **Dropout** simulator configurations. The **Basic Multi-output** and **Ensemble Multi-output** simulator configurations have the worst test dataset Δo behavioural component MSE and IQR of squared errors.

The test dataset MSE values and IQR of squared errors between simulator configurations are relatively close to one another compared to the training dataset. The test and training dataset squared errors are closest to each other for the **Ensemble Multi-output** and **Basic Multi-output** simulator configurations.

Simulator configurations with the lowest test dataset squared errors are now considered (Table 6.3). The **Dropout** simulator configuration is most accurate for the Δx and Δa behavioural components while the **Ensemble** and **Basic** simulator configurations are relatively accurate too. The **Ensemble** simulator configuration has the best accuracy for the Δy behavioural component, followed closely by the **Ensemble Multi-output** simulator configuration. For the Δo behavioural component, the **Ensemble** simulator configura-

tion is most accurate. If only one simulator configuration is chosen for all behavioural components, it is likely that an **Ensemble** simulator configuration might possess the best overall accuracy in modelling overall Snake robot behaviours. Simulator configurations consisting of multi-output SNNs achieved higher test dataset squared error values compared to single-output SNNs. This is consistent with the finding in prior research that single-output SNNs tend to be more accurate.

6.5 SNS Experiments

The fitness function used during controller evolution is given in Algorithm 3. The function takes as input the simulated path positions and associated controller commands. Fitnesses are calculated based on two component considerations. The Euclidean distance between the start and final positions of the simulated path is the first component of the fitness calculation. The second component consists of penalties applied based on uncertainty information and penalties applied to simulated head orientations violating a specified boundary.

The distance of the trajectory is extended from 3 to 9 cycles before the Euclidean distance travelled is calculated. Solution controllers have a tendency to develop unintended turning behaviours if the trajectory is not extended. Trajectories where the whole Snake robot body moves a significant distance is considered the intended behaviour. By extrapolating the estimated trajectory, the evolutionary process rewards controllers that exhibit significantly less turning behaviours.

A time-lapse of a prior solution controller evolved from a fitness function that does not extend the trajectory is presented in Figure 6.5 in order to illustrate the turning behaviour of this particular controller[1]. The robot appears to simply turn in place approximately 90 degrees every 300 seconds. This solution illustrates an example of the ER process exploiting a weakness in the fitness function. Ideal solutions would displace the entire Snake robot body from the starting position. Extending the trajectory significantly decreases the degree to which controllers exhibit such turning behaviours.

Penalties are needed in order to prevent the robot from rolling onto its back which is not

[1] https://youtu.be/WMWQ2SM5Qj4

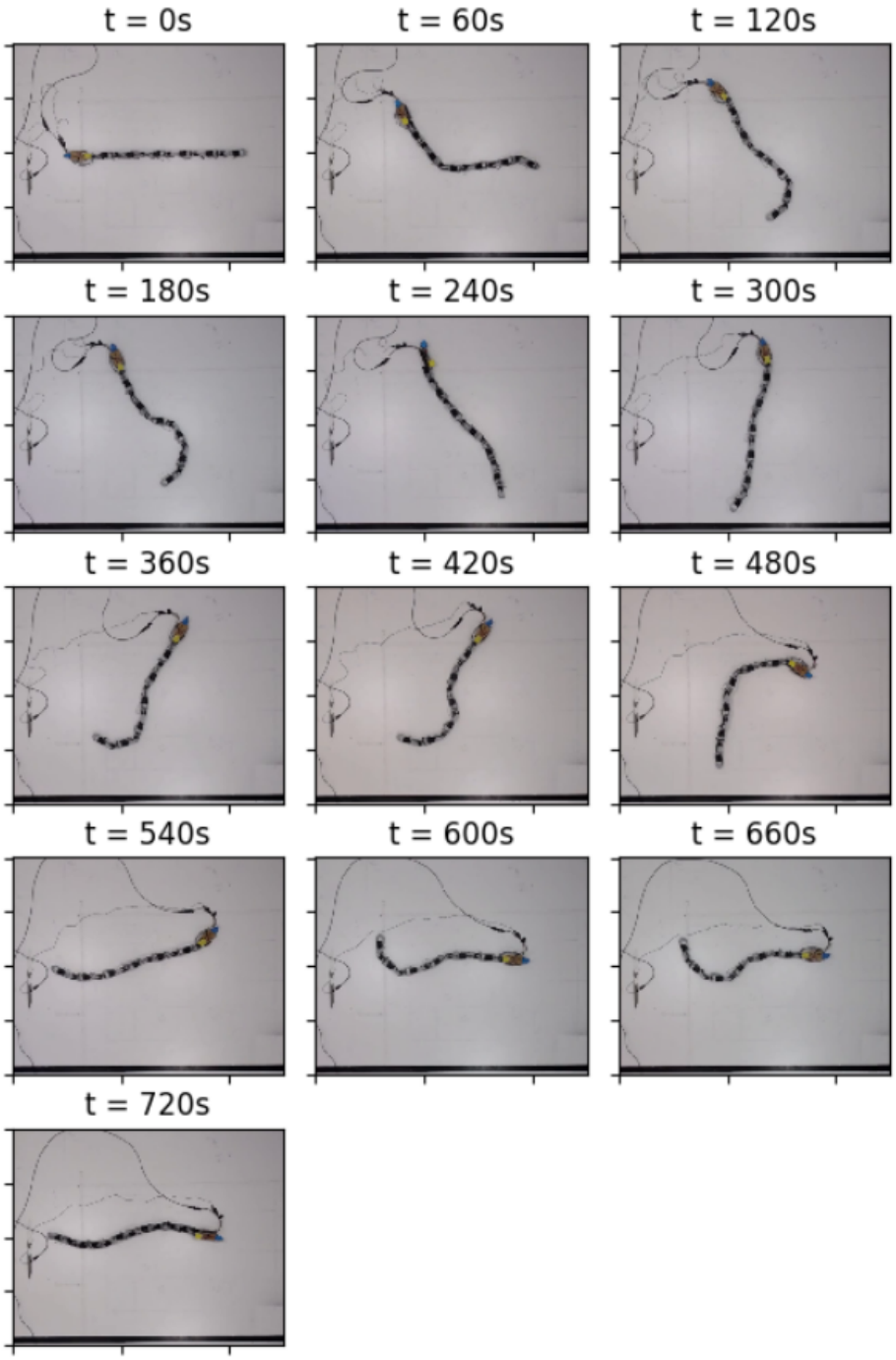


Figure 6.5: Controller solution that produces turning behaviours

Algorithm 3 Snake Controller fitness evaluation

Require: *positions* ← Simulated path list and *commands* ← List of controller commands

```
distance ← Distance between start and end of positions extrapolated 3 times
penalties ← 1
c ← A large constant
o ← A orientation cycle penalty constant
v ← A constant
for each command in commands do
    if command has reached a boundary limit then
        penalties ← penalties + c
    end if
    if each component being predicted then
        std ← Normalised standard deviation of predicted component
        penalties ← penalties + (v × std)
    end if
end for
angle ← Head orientation difference between cycles
penalties ← penalties + angle × o
fitness ← distance/penalties
return fitness
```

supported by the chosen tracking system in this research. For each evaluated command, a penalty is applied whenever the simulated orientation of the robot head relative to the starting upright position is outside the range -90 to 90 degrees. The head orientations at the end of each cycle are evolved to be as close to each other as possible in order to increase consistency in behaviours for each cycle. Consistency in behaviour for each cycle is important for the BNS approach due to sampling evaluations only evaluating a single cycle in terms of data collection.

Simulator configurations not producing uncertainty information will have prediction standard deviations of zero and will not contribute any uncertainty penalties. For each command and behavioural component simulated, the normalised standard deviation of the

predictions is determined. The normalised standard deviation is multiplied by a penalty factor and added to an accumulation of penalties. The normalisation is based on the maximum standard deviations observed for predictions on the test dataset. Normalisation is necessary in order equally consider the uncertainty weight of each behavioural component.

The parameter settings (Table 6.4) used for controller evolution is based on prior work [Woodford *et al.*, 2017]. The controller population consists of 400 controller individuals. The initial population of controllers is generated from a uniform distribution. During controller evolution, parents are chosen for crossover operations using Tournament selection with a tournaments size of 50% of the controller population. For mutations, the probability that a random change is applied to any given joint angle is 10%. A random change is drawn from a uniform distribution between -15 and 15 degrees. Controller evolution proceeds for 1000 generations and the fittest controller in the final generation is declared the final solution.

Controller Population Size	400
Initialization	Random from a uniform distribution
Selection	Tournament (Tournament size 50%)
Cross-over Method	Two-parent crossover
Mutation Rate	10%
Mutation Method	Random Component Perturbation
Controller Generation Limit	1000

Table 6.4: Parameters for controller evolution

Table 6.5 lists the 10 adaptations tested. The adaptations investigated are the same as those specified in the SNS experimental work in the Hexapod robot. Adaptations are given encoded names listed in the first column. The first letter, **S** indicated that the Snake robot is used. The second letter represents the simulator configurations. Letters **B**, **D**, **E**, **M** and **S** represent the **Basic**, **Dropout**, **Ensemble**, **Ensemble Multi-output**

and **Basic Multi-output** simulator configurations, respectively. For the third letter, **E** indicates that simulator noise is excluded while **N** indicates noise is included.

Experiment	Simulator Configuration	Simulator Noise
SBE	Basic	No
SBN	Basic	Yes
SDE	Dropout	No
SDN	Dropout	Yes
SEE	Ensemble	No
SEN	Ensemble	Yes
SME	Ensemble Multi-output	No
SMN	Ensemble Multi-output	Yes
SSE	Basic Multi-output	Yes
SSN	Basic Multi-output	No

Table 6.5: SNS Experimental Adaptations

The procedure followed in each trial run of the SNS approach is as follows:

1. An adaptation from Table 6.5 is chosen for an experimental trial run.
2. The process described in Methodology A is performed (Figure 6.1).
3. The simulated and real-world trajectory paths of the solution controller is collected as observational data.

For each adaptation listed in Table 6.5, 30 independent trial runs of the SNS approach are performed. Final solution controllers are evaluated on the real-world Snake robot and performance, transferability and behavioural properties are analysed.

6.6 The SNS Experiment Results

A successful Snake robot controller solution is demonstrate in Section 6.6.1. The performance and transferability properties of the tested SNS adaptations are presented in

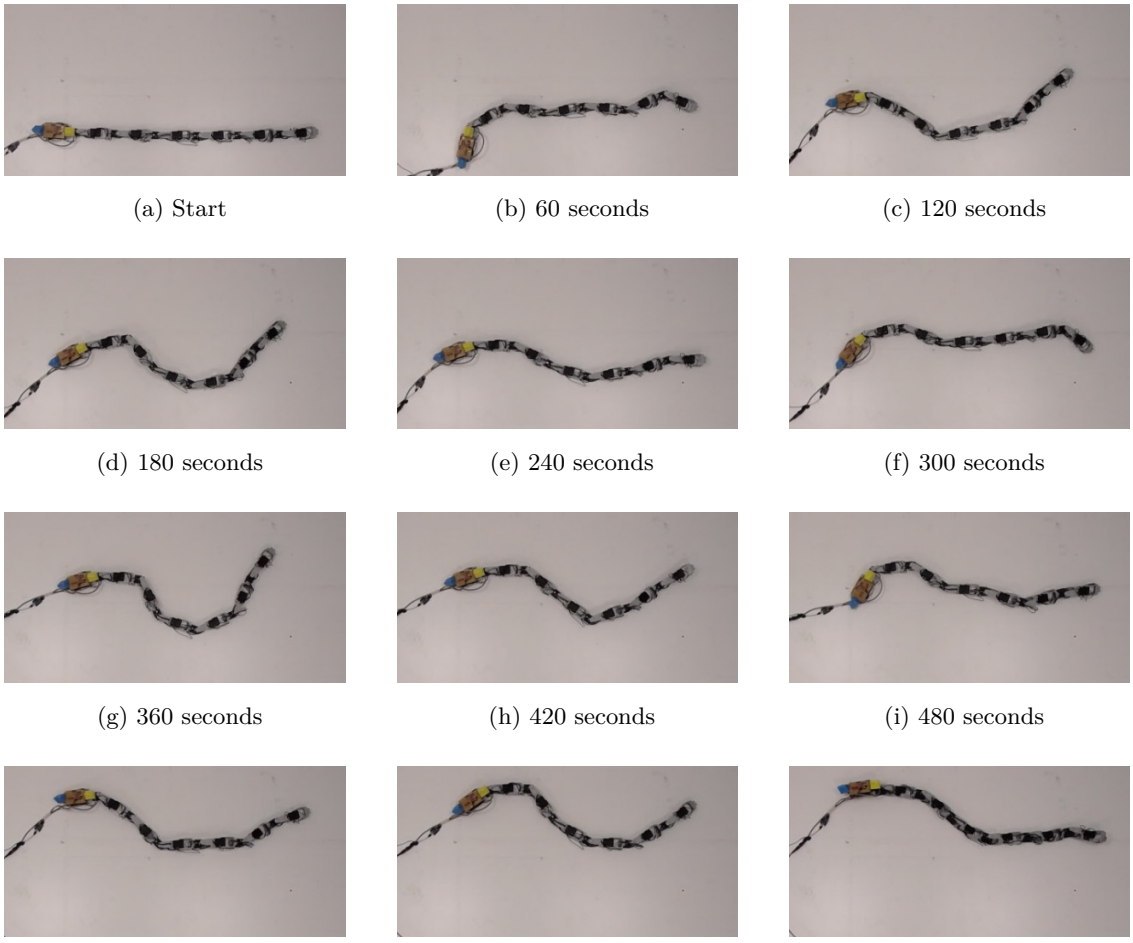
Sections 6.6.2 and 6.6.3, respectively. Identifying good SNS adaptations requires studying the performance, transferability and failure properties of solutions (Section 6.6.4). The solution trajectories are studied in Section 6.6.5 and the best performing solutions are presented in Section 6.6.6.

6.6.1 Demonstration

The SNS approach applied to the Snake robot can successfully be used to develop effective distance maximising gait controllers. The trained SNNs can simulate Snake robot behaviours over many commands. The head orientation can be simulated in order to avoid certain behaviours from developing during the ER process. The success rate for producing adequate solution controllers is relatively high when selecting appropriate adaptation settings. The SNS approach does not produce solutions that perform as consistently well compared to the Hexapod robot but that is to be expected considering the increased complexity of the problem applied to the Snake robot. This section demonstrates a single example solution produced by the SNS approach.

A time-lapse demonstration[2] of a solution developed using the SNS approach is presented in Figure 6.6. The real-world and BNS simulated trajectories are also illustrate in Figure 6.7. The head of the robot is towards the left and the tail is on the right. The time-lapse is captured such that frames are 60 seconds apart. The first command positions the robot into an initial stance, after which 13 commands are executed in 3 cycles in order to produce crawling behaviours. The starting position (Figure 6.6a) begins with the robot in its starting position. The tail end of the body shifts upward and the head rotates downward (Figure 6.6b). The middle of the body pushes the head and tail upwards (Figure 6.6c) after which the middle part of the body shifts upwards with the help of either end of the robot pulling it up (Figures 6.6d to 6.6f). Again the tail and head shift upwards in Figures 6.6g and 6.6h in order to pull the middle portion of the body upwards (Figure 6.6i). The head of the robot moves upwards (Figure 6.6j) and partially drags the middle part of the robot slightly upwards while the tail end straightens out (Figure 6.6l).

[2] https://youtu.be/4JxlYunz7k0



(a) Start (b) 60 seconds (c) 120 seconds

(d) 180 seconds (e) 240 seconds (f) 300 seconds

(g) 360 seconds (h) 420 seconds (i) 480 seconds

(j) 540 seconds (k) 600 seconds (l) 660 seconds

Figure 6.6: Solution controller demonstration (SME adaptation)

6.6.2 Performance

Performances achieved in trial runs are provided in Table B.3. The performance distributions of the tested SNS adaptations are illustrated in Figure 6.8 and summarised in Table 6.6. The Kruskal-Wallis H test is used to determine whether the performances of the tested adaptations originate from the same distribution. The tested adaptations perform significantly different to each other with a p-value of 1.34×10^{-2}.

A post hoc analysis is performed using pairwise Mann-Whitney U tests in order to determine which adaptations are significantly different from each other (Table 6.7). No statistically significant difference is found between the SBE adaptation and all other adap-

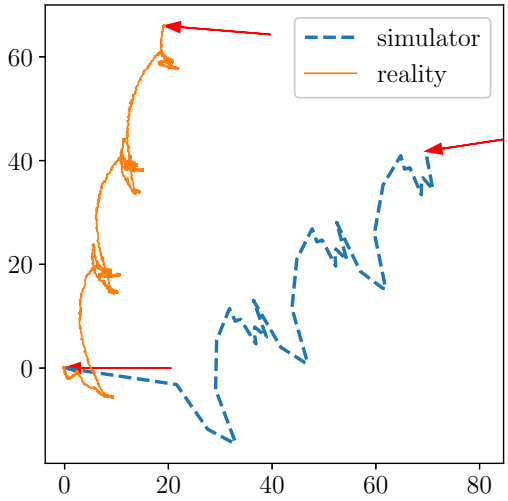


Figure 6.7: Simulated and Real-world trajectories of solution controller

tations. Solution controllers for the SBN and SDE adaptations perform statistically better than the SMN, SSE and SSN adaptations. The performance distributions for the SDN and SEE adaptations perform significantly better than the SME, SMN, SSE and SSN adaptations. Controllers evolved using the SEN adaptation perform significantly better than solutions to the SSE and SSN adaptations. Lastly, the performance distribution for the SME adaptation is significantly better than the SSE adaptation.

The SEE and SDN adaptations achieved the two highest mean and median performance outcomes. Generally, adaptations consisting of multi-output SNNs perform significantly worst than adaptations consisting of single-output SNNs. For single-output SNNs, adaptations using a **Basic** simulator configuration have lower performance medians compared to **Dropout** or **Ensemble** simulator configurations. For multi-output SNNs, adaptations using an **Ensemble Multi-output** simulator configuration have higher median performances compared to adaptations using a **Basic Multi-output** simulator configuration.

The failure rates of adaptations are given in Table 6.8. Failures occur when the Snake robot turns onto its back and can no longer be tracked. A solution can traverse a significant distance when evaluated but can still fail due to turning behaviours. The SDN and SME adaptations have the lowest failure rate at 13.3%. The second lowest failure rate is 16.7%

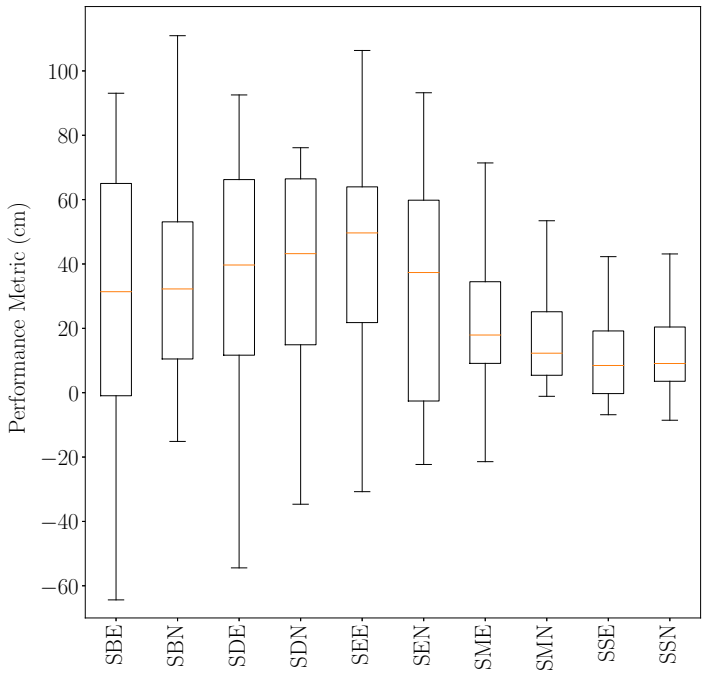


Figure 6.8: Performance distributions for SNS adaptations

for the SEE and SEN adaptations. The third lowest failure rate (20.0%) is observed for the SSN adaptation. The SBE and SMN adaptations have a failure rate of 23.3% and the SDE adaptation has a failure rate of 30.0%. The highest failure rates are observe for the SBN (36.7%) and SSE (50.0%) adaptations.

If the SNS adaptation trial runs with and without simulator noise are compared to each other, noiseless trials have a median performance of 22.7 centimetres and an IQR between 6.1 and 52.0 centimetres. Trial runs with simulator noise have a median performance of 23.3 centimetres and an IQR between 5.2 and 45.8 centimetres. Simulator noise does not appear to greatly affect the likely performance of solutions controllers for the Snake robot.

The time taken to complete a single trial run of the SNS approach is highly dependent on the simulator configuration used. Controller evaluations are slower on more complex simulator configurations. The estimated runtime for executing a single trial run of the SNS approach for adaptations using the tested simulator configurations are given in Table 6.9. Durations are estimated using a relatively high-end computer (AMD Ryzen Threadripper

	Mean	Median	Q_1	Q_3	Standard Deviation
SBE	26.2	31.4	-0.9	65.0	45.1
SBN	32.3	32.2	10.5	53.1	34.0
SDE	36.9	39.7	11.7	66.2	35.7
SDN	37.0	43.2	14.9	66.5	31.3
SEE	*42.6*	*49.7*	21.8	64.0	35.1
SEN	33.4	37.3	-2.6	59.8	32.8
SME	21.1	17.9	9.1	34.5	19.1
SMN	17.5	12.3	5.4	25.2	15.8
SSE	13.0	8.5	-0.3	19.2	19.8
SSN	13.6	9.1	3.6	20.4	16.7

Table 6.6: Performance statistics for the SNS adaptations

1920X 4.0 GHz) without using a Graphics Processing Unit (GPU). The shortest runtime durations are seen for trial runs consisting of **Basic Multi-output** and **Basic** simulator configurations. SNS trial runs using the **Basic** and **Basic Multi-output** simulator configurations take approximately 2.5 and 2 hours to complete, respectively. Adaptations consisting of an **Ensemble Multi-output** simulator configuration take approximately 5 hours to complete. Adaptations consisting of **Ensemble** or **Dropout** simulator configurations take the longest to complete at 7.5 or 6 hours, respectively.

Note that these durations are in hours while all Hexapod SNS adaptations take less than 15 minutes to complete. The Snake Robot SNNs consist of 3 times as many hidden layers compared to the Hexapod robot SNNs. A head orientation component is simulated with the Snake robot while the Hexapod robot orientations are not simulated. The controller population size for the Snake robot ER process is 4 time larger than the Hexapod robot. Hexapod robot controllers consist of 11 commands while each Snake robot controller consists of 40 commands. SNN predictions for the Hexapod robot are independent of each other which allows for bulk predictions. However, SNN predictions for the Snake robot are dependent on the head orientation reached by previous commands which creates significant overhead in predictions.

	SBE	**SBN**	**SDE**	**SDN**	**SEE**	**SEN**	**SME**	**SMN**	**SSE**
SBN	7.62×10^{-1}	-							
SDE	3.79×10^{-1}	5.20×10^{-1}	-						
SDN	4.20×10^{-1}	4.64×10^{-1}	9.12×10^{-1}	-					
SEE	2.12×10^{-1}	1.45×10^{-1}	4.92×10^{-1}	4.04×10^{-1}	-				
SEN	9.47×10^{-1}	9.94×10^{-1}	6.84×10^{-1}	4.64×10^{-1}	2.84×10^{-1}	-			
SME	4.46×10^{-1}	1.05×10^{-1}	5.37×10^{-2}	1.99×10^{-2}	1.24×10^{-3}	1.41×10^{-1}	-		
SMN	3.48×10^{-1}	2.71×10^{-2}	1.33×10^{-2}	6.38×10^{-3}	2.01×10^{-4}	8.24×10^{-2}	3.40×10^{-1}	-	
SSE	1.09×10^{-1}	3.34×10^{-3}	4.64×10^{-3}	6.91×10^{-4}	4.35×10^{-5}	3.03×10^{-2}	2.24×10^{-2}	1.49×10^{-1}	-
SSN	1.71×10^{-1}	5.32×10^{-3}	3.50×10^{-3}	1.60×10^{-3}	5.27×10^{-5}	3.78×10^{-2}	5.37×10^{-2}	2.46×10^{-1}	7.28×10^{-1}

Table 6.7: Comparisons between the performance distributions of adaptations for the SNS approach

	Failures out 30	Failure Rate
SBE	7	23.3%
SBN	11	36.7%
SDE	9	30.0%
SDN	4	13.3%
SEE	5	16.7%
SEN	5	16.7%
SME	4	13.3%
SMN	7	23.3%
SSE	15	50.0%
SSN	6	20.0%

Table 6.8: Failure rates for the tested SNS adaptations

	Time (hours)
Basic	2.5
Dropout	6
Ensemble	7.5
Ensemble Multi-output	5
Basic Multi-output	2

Table 6.9: SNS trial run durations

6.6.3 Transferability

Transferability values achieved in trial runs are provided in Table B.4. The transferability distributions observed for the tested SNS adaptations are illustrated in Figure 6.9 and summarised in Table 6.10. A lower transferability value indicates better correspondence between simulated and real-world trajectories. The Kruskal-Wallis H test is used to determine if the transferability distributions of the tested adaptations originate from the same distribution. A p-value of 4.00×10^{-11} is obtained which indicates that the tested adaptations have significantly different transferability distributions.

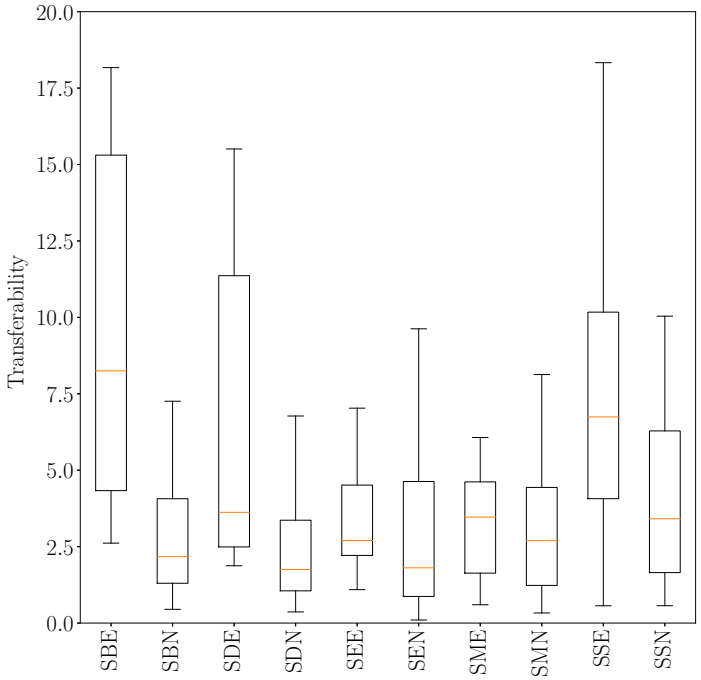


Figure 6.9: Transferability distributions for SNS adaptations

A post hoc analysis is performed using Mann-Whitney U pairwise comparisons (Table 6.11). The SBE, SDE, SME, SSE and SSN adaptations have particularly poor median transferability values. The SBE adaptation has a statistically worst transferability distribution compared to all other adaptations, except the SSE adaptation.

The SDN adaptation achieved the best median transferability. The transferability

	Mean	Median	Q_1	Q_3	Standard Deviation
SBE	21.06	8.25	4.33	15.31	33.15
SBN	4.05	2.17	1.30	4.07	5.15
SDE	10.34	3.62	2.49	11.36	15.36
SDN	7.71	1.75	1.05	3.37	27.23
SEE	3.90	2.70	2.21	4.51	3.07
SEN	4.01	1.81	0.87	4.63	5.44
SME	6.26	3.47	1.63	4.62	11.25
SMN	4.24	2.70	1.23	4.44	6.68
SSE	10.78	6.74	4.07	10.17	13.58
SSN	4.00	3.41	1.65	6.29	2.78

Table 6.10: Transferability statistics for the SNS adaptations

distribution of the SDN adaptation is not significantly better than the SBN, SEN, SME, SMN and SSN adaptations. Results also indicate that adaptations with simulator noise (SBN, SDN, SEN, SMN, SSN) generally have a better transferability compared to the corresponding noiseless variations (SBE, SDE, SEE, SME, SSE).

Experimental runs with and without simulator noise are compared to each other. Experimental runs without simulator noise have a median transferability of 4.21 and an IQR between 2.61 and 8.91. All experimental runs with simulator noise have a median transferability of 2.29 and an IQR between 1.09 and 4.73. Simulator noise greatly improves the likely transferability of solutions when applying the SNS approach to a Snake robot.

6.6.4 Comparisons

The performance distributions between noise injected and noiseless SNS adaptations are relatively close but transferability distributions differ greatly. Adaptations without simulator noise tend to greatly overestimate the real-world distances travelled. For noiseless adaptations, the controller evolution process appears to exploit simulator weaknesses and discovers overly optimistic performance outcomes. Simulator noise reduces an adaptations susceptibility to exploit weaknesses in the simulator during the ER process.

	SBE	**SBN**	**SDE**	**SDN**	**SEE**	**SEN**	**SME**	**SMN**	**SSE**
SBN	3.57×10^{-6}	-							
SDE	4.86×10^{-3}	4.03×10^{-3}	-						
SDN	8.20×10^{-7}	3.63×10^{-1}	2.53×10^{-4}	-					
SEE	2.49×10^{-6}	1.67×10^{-1}	4.51×10^{-2}	1.17×10^{-2}	-				
SEN	8.29×10^{-6}	4.83×10^{-1}	2.27×10^{-3}	9.59×10^{-1}	5.19×10^{-2}	-			
SME	2.96×10^{-5}	2.40×10^{-1}	9.33×10^{-2}	6.15×10^{-2}	9.94×10^{-1}	1.19×10^{-1}	-		
SMN	2.68×10^{-6}	8.30×10^{-1}	9.88×10^{-3}	3.48×10^{-1}	2.97×10^{-1}	4.12×10^{-1}	3.79×10^{-1}	-	
SSE	2.77×10^{-1}	6.36×10^{-5}	8.50×10^{-2}	2.43×10^{-5}	8.66×10^{-5}	1.41×10^{-4}	1.00×10^{-3}	3.59×10^{-5}	-
SSN	2.43×10^{-5}	2.28×10^{-1}	1.26×10^{-1}	5.55×10^{-2}	8.42×10^{-1}	1.15×10^{-1}	8.42×10^{-1}	2.90×10^{-1}	9.52×10^{-4}

Table 6.11: Comparisons between the transferability distributions of adaptations for the SNS approach

The performance, transferability and failure rate of each SNS adaptation is now considered. Good SNS adaptations would ideally perform well in all these criteria. Figures 6.10 and 6.11 illustrate the relationships using three and two dimensional scatter plots. Axes include the number of solution failures, median performance and median transferability. Each SNS adaptation tested is plotted as a single point and annotated with the encoded adaptation name. Better adaptations have a high performance, a low transferability and few controller failures. In the scatter plot, the SEN, SDN and SEE adaptations are grouped together in the ideal zone. Adaptations consisting of an **Ensemble** simulator configuration, with or without simulator noise (SEN, SEE) or the **Dropout** simulator configuration with noise (SDN) are the overall best SNS adaptations when applied to a Snake robot.

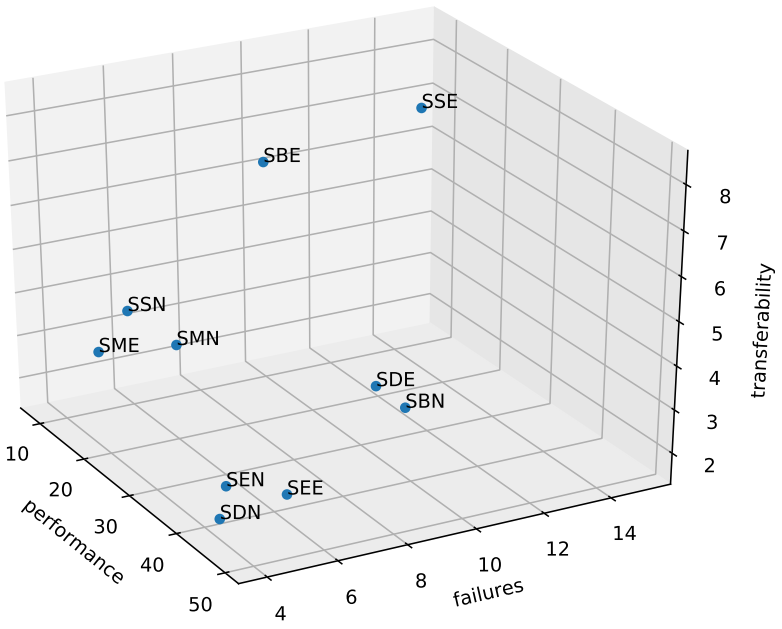
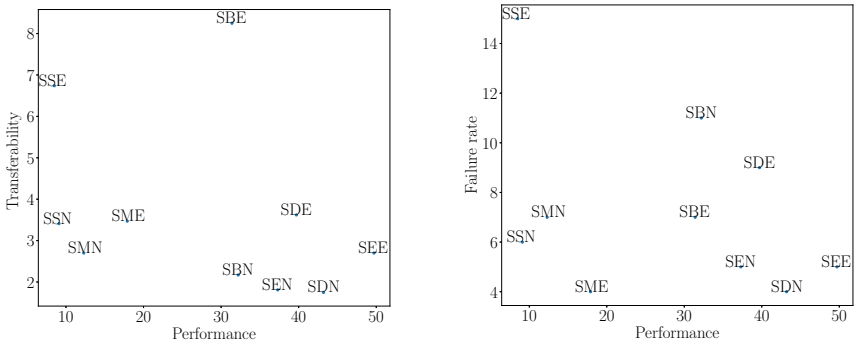


Figure 6.10: Scatter plot comparing the tested adaptations according to the number of failed controller solutions, median performance and median transferability



(a) Transferability versus performance (b) Failure rate versus performance

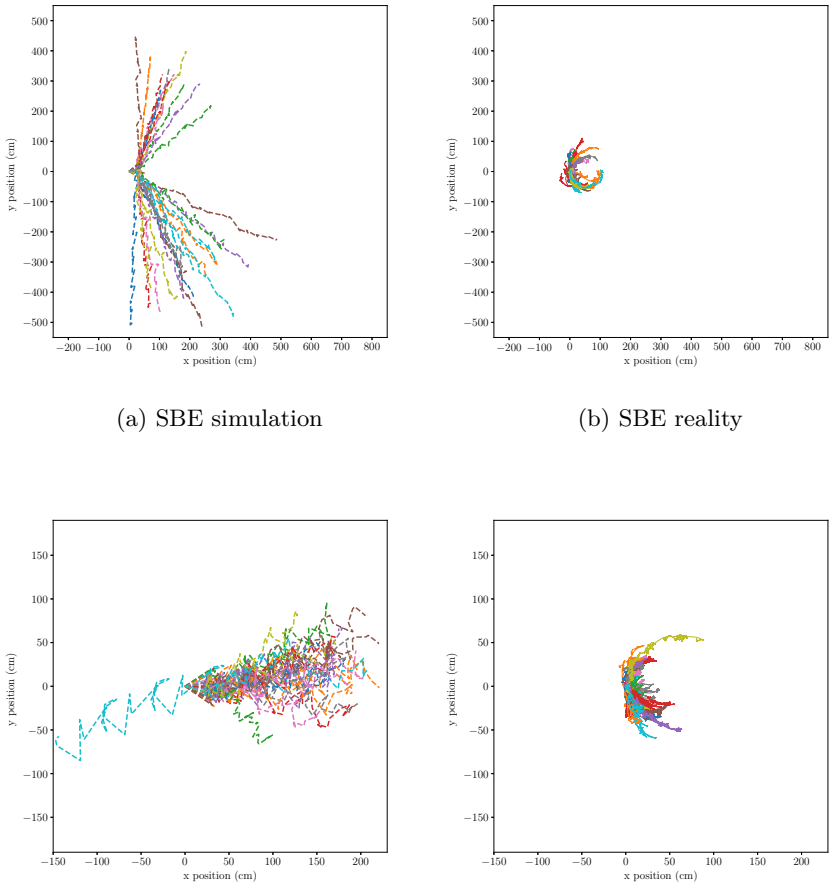
Figure 6.11: Scatter plot (2D) comparing the tested adaptations according to the number of failed controller solutions, median performance and median transferability

6.6.5 Simulated and Real-world Trajectories

The simulated and real-world trajectories of evaluated controller solutions for adaptations consisting of **Basic** and **Basic Multi-output** simulator configurations are illustrated in Figures 6.12 and 6.13. Simulated and real-world solution trajectories for adaptations using the **Dropout**, **Ensemble** and **Ensemble Multi-output** simulator configurations are presented in Figures 6.14 and 6.15. For each figure, the left sub-figures illustrate the simulated trajectories of solution controllers while the right sub-figures present the corresponding real-world trajectories. For each adaptation tested, simulated and real-world trajectories are presented side-by-side in order to easily visualise differences.

The SBE and SSE adaptations (Figure 6.12) obtained the two worst transferability distributions compared to the other tested adaptations. The controller evolution process greatly exploit weaknesses in the developed simulators. Compared to the SBE and SSE adaptations, the adaptations consisting of ensembles or enable dropout during the ER process demonstrate better transferability properties. The median simulated distances travelled by solution controllers is 398.3 centimetres for the SBE adaptation, however, real-world solution trajectories travelled a median distance of 42.4 centimetres.

Simulated solution trajectory distances are again significantly overestimated for the SSE adaptation but mostly in the positive x direction instead of either the positive or



(a) SBE simulation

(b) SBE reality

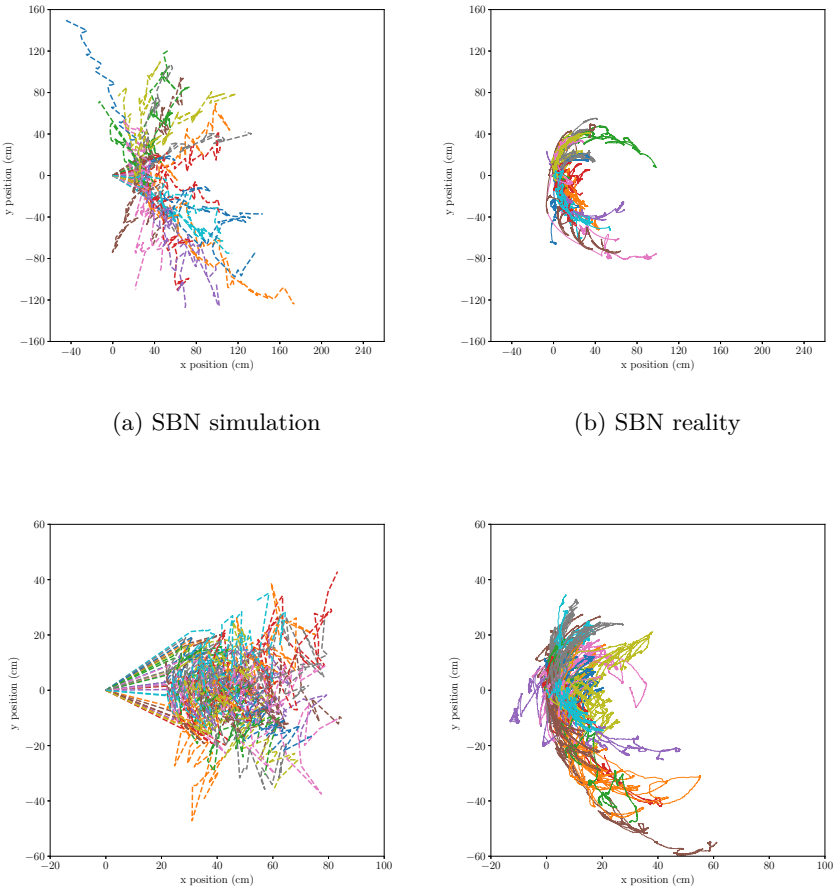
(c) SSE simulation

(d) SSE reality

Figure 6.12: Solution paths for the Basic and Basic Multi-output simulator configurations without noise

negative y direction. Only a single simulated solution travels in the negative x direction. No real-world trajectories travel a significant distance in the negative x direction. The extent of the simulated overestimation is reduced compared to the SBE adaptation. Simulated solutions from the SSE adaptations travel a median distance of 179.7 centimetres while real-world solutions travel a median distance of 26.8 centimetres. The **Basic Multi-output** simulator configuration can reduce simulated overestimations compared to the **Basic** simulator configuration.

The SBN and SSN adaptations (Figure 6.13) have a closer correspondence between



(a) SBN simulation

(b) SBN reality

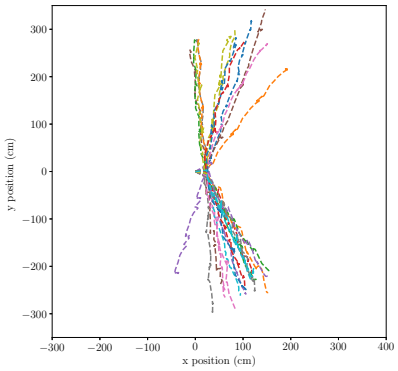
(c) SSN simulation

(d) SSN reality

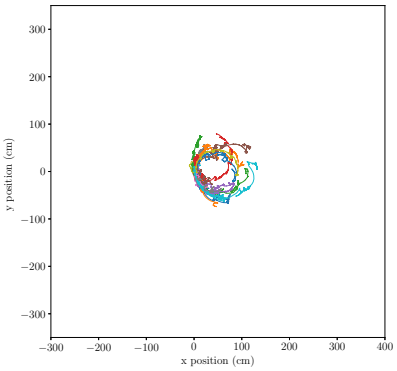
Figure 6.13: Solution paths for the Basic and Basic Multi-output simulator configurations with noise

simulated and real-world solution trajectories compared to the noiseless adaptations. The simulated and real-world median distances travelled are 115.4 and 43.7 centimetres, respectively. A subset of controller solutions transfer poorly and the real-world curvature of trajectories are not well captured.

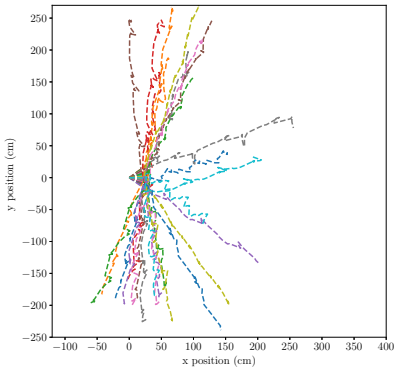
For the SSN adaptation, the median distances travelled for the simulated and real-world trajectories are 65.6 and 16.7 centimetres, respectively. The performance of solutions for the SSN adaptation are particularly poor. The simulated trajectories move a significant distance in the positive x direction but real-world trajectories do not display



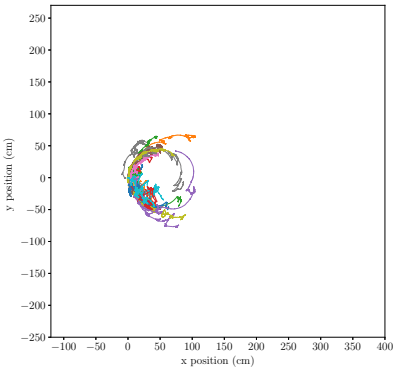
(a) SDE simulation



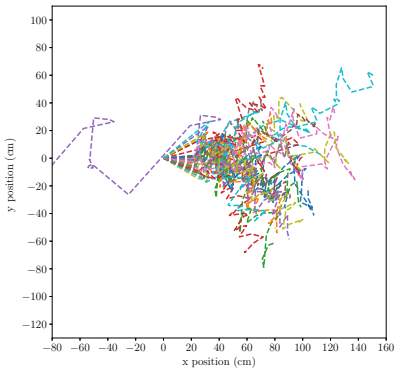
(b) SDE reality



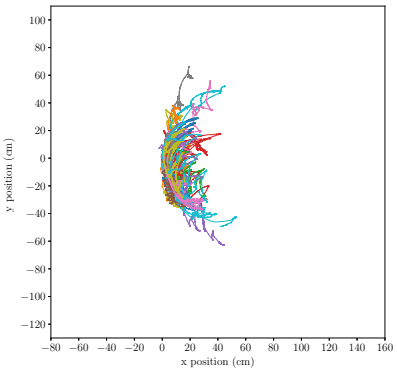
(c) SEE simulation



(d) SEE reality

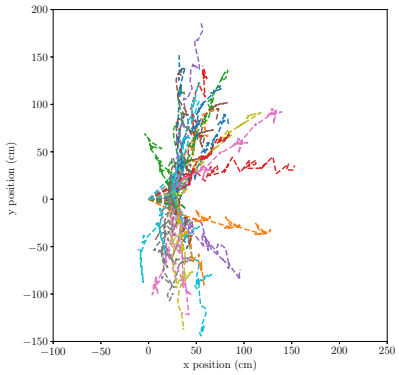


(e) SME simulation

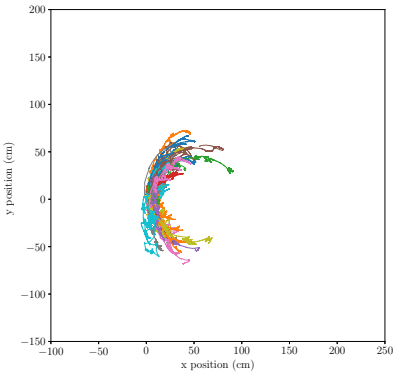


(f) SME reality

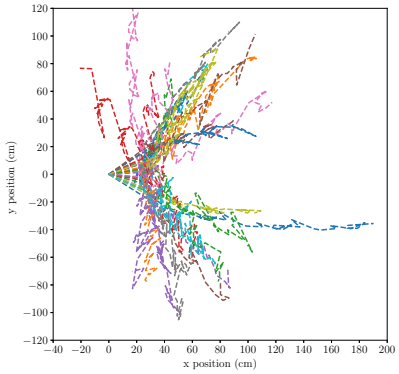
Figure 6.14: Solution paths for Dropout, Ensemble and Ensemble Multi-output configurations without noise



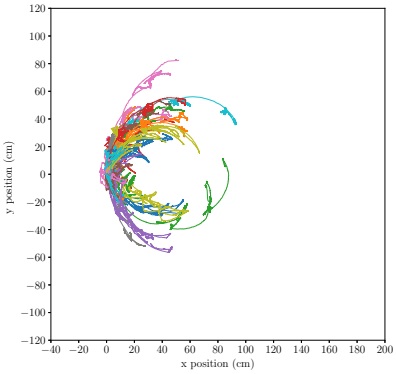
(a) SDN simulation



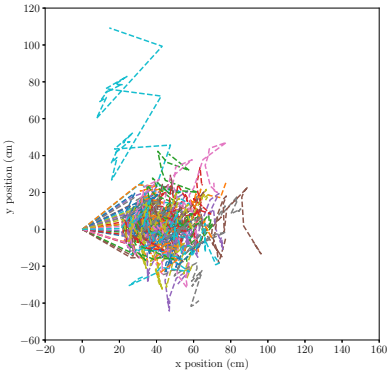
(b) SDN reality



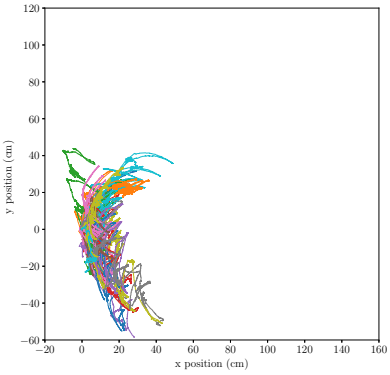
(c) SEN simulation



(d) SEN reality



(e) SMN simulation



(f) SMN reality

Figure 6.15: Solution paths for Dropout, Ensemble and Ensemble Multi-output configurations with noise

such behaviour. Few training patterns exhibit behaviours where the Snake robot travels a significant distance in the positive x direction without also moving a significant distance in the positive or negative y directions.

Solutions developed using noiseless simulator configurations that consist of ensembles (SEE, SME) or enable dropout (SDE) during the ER process (Figure 6.14) greatly overestimate the actual distances travelled compared to adaptation variations with simulator noise. However, the degree of overestimation is not as drastic compared to the SBE or SSE adaptations. The median simulated distances travelled for the SEE, SME and SDE adaptations are 200.9, 98.1 and 276.9 centimetres, respectively. The median real-world distances travelled for the SEE, SME and SDE adaptations are 60.6, 31.6 and 63.4 centimetres, respectively.

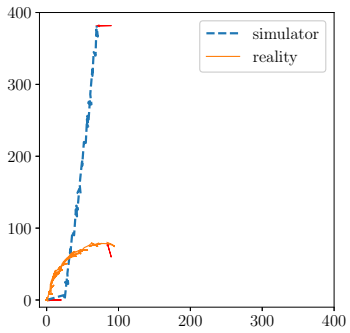
The median simulated distances travelled for solutions produced by the SEN, SMN and SDN adaptations (Figure 6.15) are 98.7, 63.0 and 120.6 centimetres, respectively. The median real-world solution distances travelled for the SEN, SMN and SDN adaptations are 45.8, 22.4 and 46.0 centimetres, respectively. Adaptations consisting of multi-output SNNs tends to traverse significantly shorter real-world distances compared to single-output SNNs. For the SMN adaptation, simulated solutions travelling a significant distance in the positive x-direction when in reality most movement is in the positive/negative y-directions. The simulated behaviours for the Snake robot appear to be greatly affected by the chosen simulator configuration.

The SDN and SEN adaptations demonstrate the most correspondence between simulated and real-world trajectories. However, the SDN adaptation has a greater correspondence between simulated and real-world behaviours compared to the SEN adaptation. The SEE adaptation generally performs well and has a low failure rate but there is little correspondence between simulated and real-world trajectories.

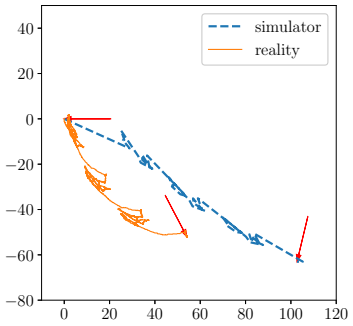
6.6.6 Best Controllers

The best performing solution to each SNS adaptation are illustrated in Figures 6.16 and 6.17. Each sub-figure has a different scaling dimension depending on distances travelled. Dashed lines represent the simulated trajectories of evaluated solution controller. Solid lines represent the real-world trajectories of solution controllers. The robot's initial head-

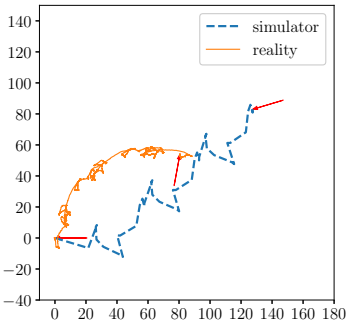
ing is illustrated by a red arrow facing the negative x-direction and pointing at the origin. The simulated and real-world final position headings for each solution are illustrated as red directional arrows.



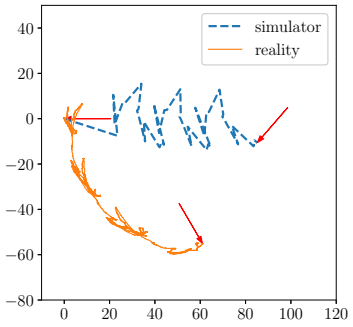
(a) Best SBE controller



(b) Best SBN controller



(c) Best SSE controller

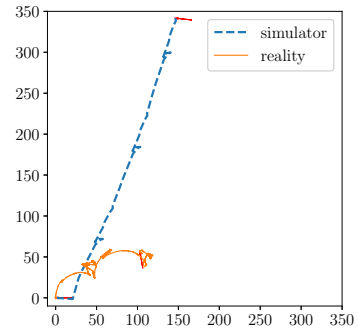


(d) Best SSN controller

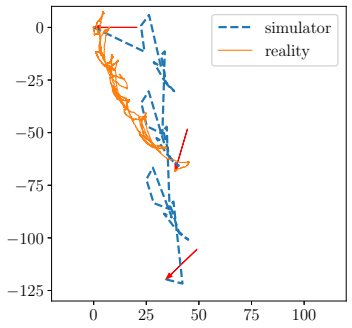
Figure 6.16: Best performing solutions for the Basic and Basic Multi-output simulator configurations

The best solution controllers for the SBE (Figure 6.16a) and SDE (Figure 6.17a) adaptations appear to have relatively similar simulated and real-world trajectories. The simulated distances travelled in the upward direction are greatly overestimated and poorly captures the real-world curvature of the trajectories.

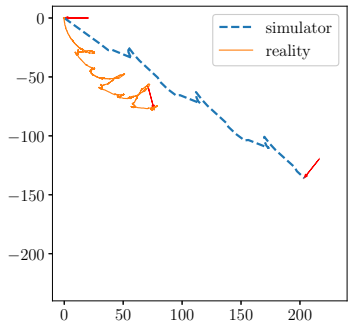
The SBN (Figure 6.16b) and SEE (Figure 6.17c) adaptations appear to have similar solutions trajectories for their best performing solutions. Both simulated trajectories overestimate the distances travelled but the overall direction is accurate. The simulated



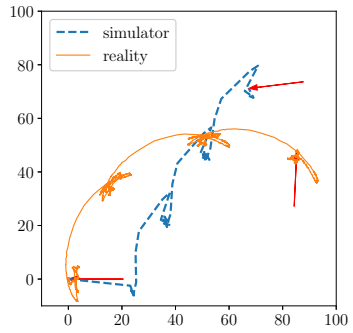
(a) Best SDE controller



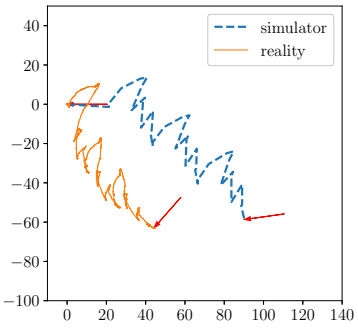
(b) Best SDN controller



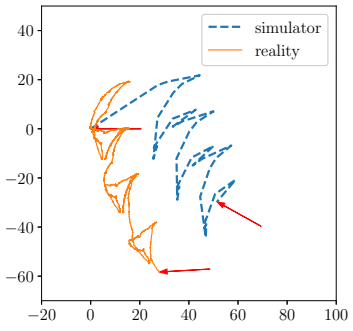
(c) Best SEE controller



(d) Best SEN controller



(e) Best SME controller



(f) Best SMN controller

Figure 6.17: Best performing solutions for the Dropout, Ensemble and Ensemble Multi-output configurations

trajectory for the best SEE adaptations is overestimated significantly more than than the best SBN solution. Trajectories move towards the bottom right. The slight real-world curvature is not particularly well captured in simulation.

For the best solutions to the SSE (Figure 6.16c) and SEN (Figure 6.17d) adaptations, controllers possess relatively similar real-world and simulated trajectories. The real-world curvature of the trajectories are not well simulated and there are significant differences in the final simulated and real-world headings. The overall directions and distances travelled are relatively accurate. The simulator overestimates the distance travelled for the SSE adaptation. The simulated distance travelled for the best SEN adaptation is not overestimated.

The best solution controller for the SDN adaptation (Figure 6.17b) has a relatively unique solution compared to the other best solutions. The simulated and real-world trajectories move downwards. The real-world distance travelled is overestimated but the simulated and real-world final headings are reasonably close.

The SME (Figure 6.17e) and SMN (Figure 6.17f) adaptations appear to have relatively similar solution trajectories for the best performing solutions. The simulate and real-world trajectories are similar in shape and trajectories move towards the bottom right in direction. There is good correspondence between the simulated and real-world headings and distances travelled.

The best solution controller for the SSN adaptation (Figure 6.16d) has a simulated trajectory moving towards the right while the real-world trajectory is towards the bottom right. The simulated distance travelled is relatively accurate but the direction is off. The simulated curvature and final heading are significantly different compared to reality.

Behavioural metrics for the best performing solution controller to each SNS adaptation is given in Table 6.12. The performance and transferability columns are defined as in previous sections. The position error is the Euclidean distance between a solution's final simulated and real-world positions. The actual distance is the Euclidean distance between the starting and final positions of the real-world trajectory. The simulated distance is the Euclidean distance between the starting and final positions in simulation. The heading error is the difference between the robot's final simulated and real-world position headings.

For the best performing solution controllers, solutions using noiseless adaptations tend

Adaptation	Performance	Transferability	Position Error (cm)	Actual Distance (cm)	Simulated Distance (cm)	Heading Error (degrees)
SBE	93.1	2.62	302.6	115.7	387.7	75.1
SBN	73.3	0.66	49.8	75.0	120.5	41.1
SDE	92.5	2.45	288.4	117.5	371.7	73.7
SDN	76.1	0.66	51.8	78.3	124.5	29.5
SEE	106.4	1.29	140.0	108.6	244.6	53.4
SEN	93.2	0.30	29.6	97.7	98.0	99.6
SME	71.4	0.60	46.1	76.9	107.7	40.7
SMN	53.4	0.57	37.0	64.7	59.4	32.4
SSE	97.0	0.57	55.1	97.0	152.0	117.1
SSN	67.3	0.62	50.7	82.2	85.8	71.1

Table 6.12: Behavioural metrics of the best controller in each adaptation

to perform better than the corresponding noise injected adaptation. In Table 6.12, the noiseless **Basic** (SBE), **Dropout** (SDE) and **Ensemble** (SEE) simulator configurations have particularly bad transferability values compared to the noise injected variations (SBN, SDN and SEN). However, the **Ensemble Multi-output** (SME and SMN) and **Basic Multi-output** (SSE and SSN) simulator configurations have similarly good transferability measures. For the final position errors, actual distances and simulated distances, the noiseless simulator configurations (SBE, SDE, SEE, SME, SSE) have higher values compared to the corresponding noise injected variations (SBN, SDN, SEN, SMN, SSN). However, the error between the simulated and real-world final position headings are higher for the noiseless simulator configuration, except for the **Ensemble** simulator configuration.

These top performer examples illustrate that simulating Snake robot behaviours accurately over many commands is a difficult problem. Practical applications in future work would likely require some form of real-time feedback mechanism in order to correct for drift. Top performing SNS adaptations might be dependent on the goal task and/or robot morphology used.

6.7 Comparing the SNS approach for the Hexapod and Snake robots

The experimental results identified different ideal SNS adaptation settings between the Hexapod and Snake robots. The best performing and highly transferable adaptation for the Hexapod robot consisted of a noiseless **Ensemble Multi-output** simulator configuration (HME). For the Hexapod robot, all noiseless adaptations perform better than adaptations with noise. Additionally, Hexapod simulator configurations with multi-output SNNs have significantly better transferability compared to single-output SNNs.

For the Snake robot, controllers can fail by turning upside down when evaluated. Compared to the Hexapod robot, the SNS approach is less consistent in finding effective controller solutions for the Snake robot. The Snake robot has more complex dynamics and a larger number of commands to execute. The complex Snake robot morphology used in this research is seldom used to investigate ER approaches in the literature. This is likely due to the difficulty with accurately simulating Snake robot behaviours and developing

transferable solutions using current ER approaches.

A noiseless **Ensemble Multi-output** adaptation (SME) performs relatively poorly when applied to the Snake robot. If performance, transferability and failure rates are considered together, the best Snake robot adaptations consisted of either the noisy **Dropout** simulator configuration (SDN) or the **Ensemble** simulator configurations (SEE and SEN). For the Snake robot, adaptations with noise tend to have better transferability properties compared their noiseless counterparts. The Snake robot simulators developed in this work required the modelling of the head orientation. The extra behavioural component increased the likelihood of inaccuracies building up during simulated controller evaluations.

The Hexapod and Snake robot simulators are trained using similar amounts of training patterns. Data collection for the Snake robot is slower. The data collection process can generate approximately 308 patterns per hour for the Hexapod robot and 204 patterns per hour for the Snake robot. The SNNs used to simulate the Snake robot are significantly larger compared to the Hexapod SNNs. This indicates the behaviours modelled for the Snake robot are more complex compared to the Hexapod robot. Controller evolution for the Snake robot is significantly slower compared to the Hexapod robot. For the Hexapod robot, a trial run of the SNS approach can take between 1.5 and 15 minutes to complete, depending on the simulator configuration used. For the Snake robot, a trial run of the SNS approach can take between 2 and 7.5 hours to complete, depending on the simulator configuration used.

6.8 Conclusion

The SNS approach can be used to discover effective gait controllers for a Snake robot. Many independent ER trial runs may be required in order to produce multiple solutions. This increases the likelihood of discovering effective solutions. This chapter is a first time demonstration of the viability of the SNS approach applied to a complex Snake robot without relying on controller designs with significant prior human knowledge.

The experimental work performed in this chapter is a repeat of the Hexapod SNS experimental work. Behavioural data is collected by evaluating randomly generated controllers on a real-world Snake robot. The collected behavioural data is used to independently train

various simulator configurations that model Snake robot behaviours. Proposed adaptations are investigated in the experimental work, producing representative samples of solution controllers for each SNS adaptation. The performance, failure and transferability properties of the tested adaptations were presented and analysed.

The **Ensemble Multi-output** simulator configuration without noise (SME) did not perform significantly better than other adaptations. The ideal SNS adaptation for the Snake robot differs from the ideal SNS adaptation for the Hexapod robot. The best adaptations (SEE, SDN and SEN) are identified by considering performance, transferability and failure properties of the tested SNS adaptations. The addition of simulator noise can significantly improve the likely transferability of controller solutions without negatively affecting performance outcomes.

Adaptations consisting of multi-output SNNs tend to have simulated trajectories that move from the head towards the centre of the robot. Single-output SNN configurations usually produce simulated trajectories where the robot shifts sideways. The overall behavioural differences of simulated trajectories between adaptations using single or multi-output SNNs indicates that the SNN architecture can significantly affect simulated behaviours and the effectiveness of the SNS approach. Future research should investigate actually evolving the simulator configurations and SNN architectures in order to improve transferability.

Chapter 7

SNAKE BOOTSTRAPPED NEURO-SIMULATION

7.1 Introduction

Prior BNS related research performed on the Snake robot relied on smart controller designs with prior knowledge of locomotion modes. No prior work has investigated the BNS approach on a Snake robot morphology using a controller design that does not rely on a high level of prior human knowledge. The BNS approach eliminates the data collection phase required by the SNS approach. Data collection, simulator training and controller evolution happen simultaneously. This chapter is a repeat of the Hexapod BNS experimental work but applied to a Snake robot. Details of the experimental procedures are discussed in Section 7.2.

In order to identify effective BNS adaptations, all combinations of adaptation settings are investigated completely in simulation (Section 7.3). A successful Snake robot solution controller is demonstrate in Section 7.4. Promising BNS adaptations are identified based on these simulated BNS experiments (Section 7.5). Promising adaptations are validated on a real-world Snake robot by way of a set of validation experiments (Section 7.6). For the Snake robot, the SNS and BNS experimental results are compared to each other in Section 7.7. Lastly, conclusions to the chapter are drawn in Section 7.8.

7.2 Experimental Procedure

The simulated BNS experiments (Section 7.3.1) follow the methodology illustrated in Figure 7.1 (Methodology B). Once promising BNS adaptations have been identified, these adaptations are demonstrated using Methodology C (Figure 7.2).

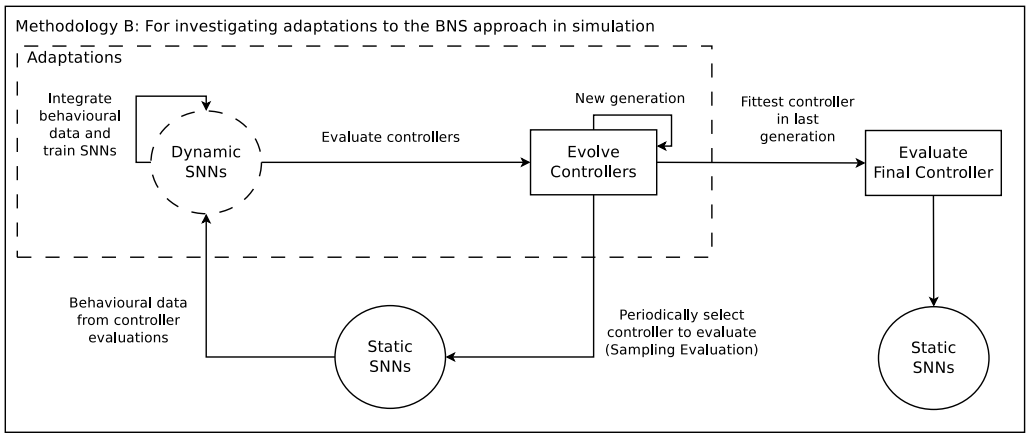


Figure 7.1: Methodology B: Simulated experimental trial of BNS adaptation

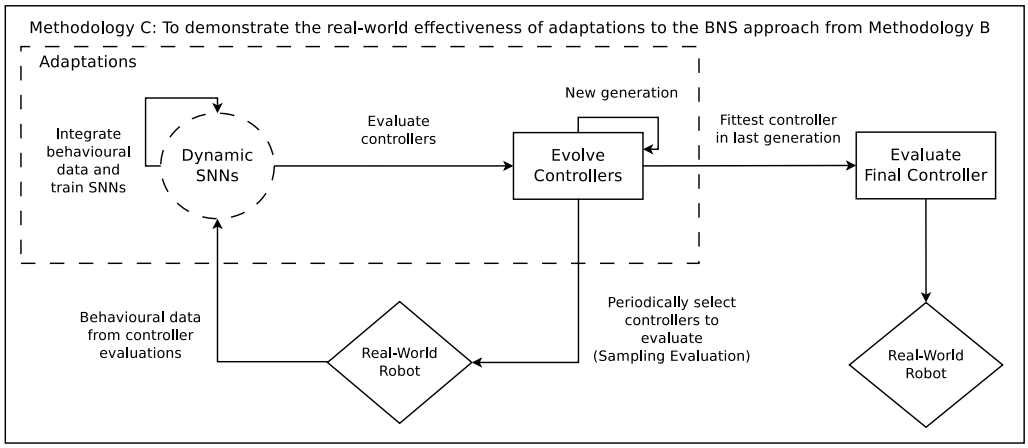


Figure 7.2: Methodology C: Real-world demonstration of BNS adaptation

The data acquisition process is discussed in Section 7.2.1. The controller design is similar to that used during the SNS experimental work for the Snake robot, however, certain modifications are required (Section 7.2.2). The simulator design and simulator training process are discussed in Section 7.2.3.

7.2.1 Hardware and Data Capture

The Snake robot morphology discussed in Section 3.2.2 remains unchanged from the SNS experimental work on the Snake robot. During the execution of the BNS process, controllers are selected from the evolving controller population in order to acquire behavioural data. The collected behavioural data is periodically integrated into the simulator training process. If the robot fails by turning upside down during a controller evaluation, the evaluation is paused and the robot is reset to the starting position. Once the robot is reset, the paused evaluation is resumed.

7.2.2 Controllers

The fitness function (Algorithm 3) remains unchanged from the SNS approach applied to the Snake robot. The Snake robot controller design remains mostly unchanged from the SNS experimental work (Section 6.2.1). However, uncertainty penalty normalisation requires modification. The uncertainty penalty normalisation parameter settings are initially unknown due to no behavioural dataset being available at the start of the BNS approach. The normalisation parameter settings are continually recalculated based on the maximum standard deviations observed when evaluating the controller population. The same calculation is performed for the BNS approach applied to the Hexapod robot (Section 5.2.2).

7.2.3 Simulator

The simulator design is similar to the SNS experimental work (Section 6.2.2). The BNS approach requires modifications to the behavioural dataset standardisation, simulator training and simulator noise generation. Similar modifications are required for the BNS approach when applied to the Hexapod robot (Section 5.2.3).

The behavioural dataset is standardised before any simulator training is performed. All behavioural data features and outputs are standardised to have a zero mean and unit standard deviation. The parameter setting values used to standardise the behavioural data are periodically recalculated (twice per sampling evaluation). Parameter setting values are calculated as the mean and standard deviation of the latest available behavioural dataset.

Dynamic SNNs are periodically trained after every 7 behavioural patterns are collected (twice per sampling evaluation). A sampling evaluation only executes a controller for a single cycle in order to reduce the total runtime required to complete a BNS trial run. For each behavioural pattern collected, there is an 80% probability it will be added to the training dataset and a 20% probability of being added to the validation dataset. Each training event involves 1000 iterations of the Adam training algorithm. Early stopping is used where training terminates when the validation MSE does not improve within 10 iterations. Dynamic SNN weights associated with the lowest validation MSE values are used after each training event.

The simulator noise parameter settings are dynamically calculated during the BNS approach. After every simulator training event, the parameter settings used to generate simulator noise are recalculated. The noise distribution of each behavioural component is Gaussian with a mean of zero and a standard deviation equal the standard deviation of the observed errors between the simulator and validation dataset.

7.3 BNS Experiments

The BNS experimental work consists of two phases. The first phase investigates all combinations of BNS adaptation settings in a completely simulated environment (Section 7.3.1). Once promising adaptations are identified, validation experiments are used to demonstrate the viability of the BNS approach on a real-world Snake robot (Section 7.3.2).

7.3.1 The Simulated BNS Experiments

Static SNNs developed in the previous chapter are used as an alternative to a real-world Snake robot. A large number of BNS trial runs are needed to complete the Simulated BNS Experimental work. Investigating a large number of BNS adaptations is practically infeasible in reality. Controller evolution parameter settings used for each BNS trial run remains unchanged from the SNS experimental work (Table 6.4). The **High Fitness** sampling strategy uses a tournament selection method with a tournament size of 280 randomly chosen controllers. This tournament size is chosen based on prior research [Woodford *et al.*, 2017].

A total number of 64 unique combinations of adaptation settings are tested (Tables 7.1 to 7.4). Encoded names for each adaptation are given in the first columns. The first letter of the encoding scheme stands for the Snake robot morphology (**S**: Snake robot). The second letter indicates the simulator configuration used (**B**: Basic, **E**: Ensemble, **D**: Dropout, **M**: Ensemble Multi-output, **S**: Basic Multi-output). The third letter indicates the resetting procedure (**N**: None, **C**: Controller, **S**: Simulator, **B**: Both). The fourth letter indicates if simulator noise is present (**N**: including simulator noise; **E**: exclude simulator noise). The sampling strategy (**T**: Tournament, **U**: Most uncertain) is indicated by the last letter of the encoded name. Thirty independent trial runs of the BNS approach are conducted per adaptation.

Adaptations	**Simulator Configuration**	**Resetting Procedure**	**Simulator Noise**	**Samplng Strategy**
SBNNT	Basic	None	Yes	High Fitness
SBNET	Basic	None	No	High Fitness
SENNT	Ensemble	None	Yes	High Fitness
SENNU	Ensemble	None	Yes	Most Uncertain
SENEU	Ensemble	None	No	Most Uncertain
SENET	Ensemble	None	No	High Fitness
SDNNT	Dropout	None	Yes	High Fitness
SDNET	Dropout	None	No	High Fitness
SDNNU	Dropout	None	Yes	Most Uncertain
SDNEU	Dropout	None	No	Most Uncertain
SMNEU	Ensemble Multi-output	None	No	Most Uncertain
SMNNT	Ensemble Multi-output	None	Yes	High Fitness
SMNNU	Ensemble Multi-output	None	Yes	Most Uncertain
SMNET	Ensemble Multi-output	None	No	High Fitness
SSNNT	Basic Multi-output	None	Yes	High Fitness
SSNET	Basic Multi-output	None	No	High Fitness

Table 7.1: BNS adaptations using no resetting procedure

The procedure for each Simulated BNS Experimental run is as follows:

1. A specific adaptation is chosen from the configurations listed in Tables 7.1 to 7.4.
2. The BNS approach described in Methodology B (Figure 7.1) is performed.
3. The ER process continues until 50 controllers have been evaluated using the substitute real-world. For each sampling evaluation, only a single cycle of the controller is

Adaptations	Simulator Configuration	Resetting Procedure	Simulator Noise	Sampling Strategy
SBCNT	Basic	Controller	Yes	High Fitness
SBCET	Basic	Controller	No	High Fitness
SECNT	Ensemble	Controller	Yes	High Fitness
SECNU	Ensemble	Controller	Yes	Most Uncertain
SECEU	Ensemble	Controller	No	Most Uncertain
SECET	Ensemble	Controller	No	High Fitness
SDCNT	Dropout	Controller	Yes	High Fitness
SDCET	Dropout	Controller	No	High Fitness
SDCNU	Dropout	Controller	Yes	Most Uncertain
SDCEU	Dropout	Controller	No	Most Uncertain
SMCEU	Ensemble Multi-output	Controller	No	Most Uncertain
SMCNT	Ensemble Multi-output	Controller	Yes	High Fitness
SMCNU	Ensemble Multi-output	Controller	Yes	Most Uncertain
SMCET	Ensemble Multi-output	Controller	No	High Fitness
SSCNT	Basic Multi-output	Controller	Yes	High Fitness
SSCET	Basic Multi-output	Controller	No	High Fitness

Table 7.2: BNS adaptations using the controller resetting procedure

Adaptations	Simulator Configuration	Resetting Procedure	Simulator Noise	Sampling Strategy
SBSNT	Basic	Simulator	Yes	High Fitness
SBSET	Basic	Simulator	No	High Fitness
SESNT	Ensemble	Simulator	Yes	High Fitness
SESNU	Ensemble	Simulator	Yes	Most Uncertain
SESEU	Ensemble	Simulator	No	Most Uncertain
SESET	Ensemble	Simulator	No	High Fitness
SDSNT	Dropout	Simulator	Yes	High Fitness
SDSET	Dropout	Simulator	No	High Fitness
SDSNU	Dropout	Simulator	Yes	Most Uncertain
SDSEU	Dropout	Simulator	No	Most Uncertain
SMSEU	Ensemble Multi-output	Simulator	No	Most Uncertain
SMSNT	Ensemble Multi-output	Simulator	Yes	High Fitness
SMSNU	Ensemble Multi-output	Simulator	Yes	Most Uncertain
SMSET	Ensemble Multi-output	Simulator	No	High Fitness
SSSNT	Basic Multi-output	Simulator	Yes	High Fitness
SSSET	Basic Multi-output	Simulator	No	High Fitness

Table 7.3: BNS adaptations using the simulator resetting procedure

Adaptations	Simulator Configuration	Resetting Procedure	Simulator Noise	Sampling Strategy
SBBNT	Basic	Both	Yes	High Fitness
SBBET	Basic	Both	No	High Fitness
SEBNT	Ensemble	Both	Yes	High Fitness
SEBNU	Ensemble	Both	Yes	Most Uncertain
SEBEU	Ensemble	Both	No	Most Uncertain
SEBET	Ensemble	Both	No	High Fitness
SDBNT	Dropout	Both	Yes	High Fitness
SDBET	Dropout	Both	No	High Fitness
SDBNU	Dropout	Both	Yes	Most Uncertain
SDBEU	Dropout	Both	No	Most Uncertain
SMBEU	Ensemble Multi-output	Both	No	Most Uncertain
SMBNT	Ensemble Multi-output	Both	Yes	High Fitness
SMBNU	Ensemble Multi-output	Both	Yes	Most Uncertain
SMBET	Ensemble Multi-output	Both	No	High Fitness
SSBNT	Basic Multi-output	Both	Yes	High Fitness
SSBET	Basic Multi-output	Both	No	High Fitness

Table 7.4: BNS adaptations using the controller and simulator resetting procedures

evaluated in order to reduce the time taken to complete a BNS trial run.

4. The fittest controller in the final generation is selected as the solution.

5. The solution trajectories generated by the Static and Dynamic simulators are collected for analysis.

The number of controller evolution generations iterated per sampling evaluation depends on the computational complexity of the simulator configuration used. Simulated controller evaluations are slower when using more complex simulator configurations. The number of controller evolution generations achieved per sampling evaluation for each simulator configuration are given in Table 7.5. The ER process can iterate 23 generations per sampling evaluation for the **Basic** simulator configuration, 10 generations for the **Dropout** simulator configuration, 8 generations for the **Ensemble** simulator configuration, 12 generations for the **Ensemble Multi-output** simulator configuration and 30 generations for the **Basic Multi-output** simulator configuration.

A BNS trial run consisting of an **Ensemble** simulator configuration can only produce 400 controller generations after 50 sampling evaluations while the **Basic Multi-output**

simulator configuration produces 1500 controller generations during the same time period. Adaptations using more complex simulator configurations have the disadvantage of producing fewer controller generations.

Simulator Configuration	Number Generations per Sampling Evaluation
Basic	23
Dropout	10
Ensemble	8
Ensemble Multi-output	12
Basic Multi-output	30

Table 7.5: Number of controller evolution generations iterated per sampling controller evaluation for the BNS approach on the Snake robot

7.3.2 The BNS Validation Experiments

Six promising BNS adaptations are identified from the Simulated BNS Experimental work and are validated in reality. Each chosen adaptation is validated with 5 independent trial runs of the BNS approach. The simulated and real-world solution trajectories are recorded and analysed in Section 7.6.

The procedure for each BNS validation experimental run is as follows:

1. A specific adaptation is chosen from the configurations listed in Tables 7.1 to 7.4.
2. The BNS approach described in Methodology C (Figure 7.2) is performed.
3. The ER process continues until 50 controllers have been evaluated using the real-world Snake robot.
4. The fittest controller in the last generation is selected as the solution.
5. The solution trajectories generated in reality and the simulator are collected for later analysis.

7.4 Successful BNS Snake Controller

The BNS approach is successfully used to develop effective distance maximising gait controllers for the Snake robot. Simulators developed using the BNS approach are often more accurate compared to simulators developed using the SNS approach. Significantly less behavioural data is collected when compared to the SNS approach in order to produce a single solution. Certain BNS adaptations investigated demonstrate better performance and transferability properties compared to the best SNS adaptations. None of the BNS adaptation solutions demonstrated the failure behaviours observed with some solutions developed using the SNS approach. The BNS approach is a more robust method for developing gait maximising controllers for the Snake robot. This section demonstrates a single example solution produced by the BNS approach.

A time-lapse demonstration[1] of a solution developed using the BNS approach is presented in Figure 7.3. The time-lapse is captured such that frames are 60 seconds apart. The first command positions all joints into an initial stance, after which 13 commands are executed 3 times in order to form cycles. The shape of the body is important for preventing rolling behaviours. The starting position (Figure 7.3a) begins with the robot being completely straight, upright and the entire body making contact with the ground. The head of the robot is towards the left and the tail is on the right. The head of the robot angles upwards and the tail-end moves slightly upwards after 60 seconds (Figure 7.3b). The middle and head portion of the body shifts downwards (Figure 7.3c), followed by the robot adjusting its orientation such that the head is upright (Figure 7.3d). In order to reach the position in Figure 7.3e, the middle part of the body shifts downwards and pulls the head and tail down too. The head orientation is again adjusted to be upright (Figure 7.3f). The head again moves downward (Figure 7.3g) and the body straightens in order to pull the tail further down (Figure 7.3h).

[1] https://youtu.be/NLy9bBuuz7Q

(a) Start

(b) 60 seconds

(c) 120 seconds

(d) 180 seconds

(e) 240 seconds

(f) 300 seconds

(g) 360 seconds

(h) 420 seconds

Figure 7.3: Solution controller demonstration (SMSNU adaptation)

7.5 The Simulated BNS Experiment Results

A high level performance comparison between groupings of adaptation settings is presented in Section 7.5.1. Similarly, transferability comparisons between adaptation settings are studied in Section 7.5.2. The convergence properties of the tested resetting procedures over time are covered in Section 7.5.3. The best performing adaptations are identified and presented in Section 7.5.4. Lastly, a summary of the Simulated BNS Experimental results is given in Section 7.5.5.

7.5.1 Overall Comparisons

The performance distributions of the tested adaptation settings are presented in this section. Each adaptation setting is grouped over all other adaptation settings and the performance distributions are analysed. Adaptation settings investigated include the Simulator Configuration (Section 7.5.1.1), Resetting Procedures (Section 7.5.1.2), Simulator Noise (Section 7.5.1.3) and Sampling Strategies (Section 7.5.1.4). A summary regarding the overall comparison results is given in Section 7.5.1.5.

7.5.1.1 Simulator Configurations

The performance distributions observed for the tested simulator configurations are illustrated in Figure 7.4 and summary statistics are given in Table 7.6. All statistical comparisons use a significance level of 5%. The Kruskal-Wallis H test is used to determine if performance distributions of the tested simulator configurations originate from the same distribution. Performance distributions for the tested simulator configurations are significantly different from each other, with a p-value of 5.52×10^{-7}. A post hoc analysis is performed using pairwise Mann-Whitney U tests (Table 7.7).

The **Basic Multi-output** simulator configuration has the highest mean and median performance. However, the **Basic Multi-output** simulator configuration only performs significantly better than the **Basic** and **Dropout** simulator configurations. The **Ensemble** and **Basic Multi-output** simulator configurations have similar median and IQR values, with the **Ensemble** simulator configuration having a lower standard deviation in performances. Adaptations using the **Basic Multi-output** simulator configuration ap-

pear to benefit greatly from producing the most controller generations. The **Dropout** simulator configuration has a significantly worse performance distribution compared to all other simulator configurations.

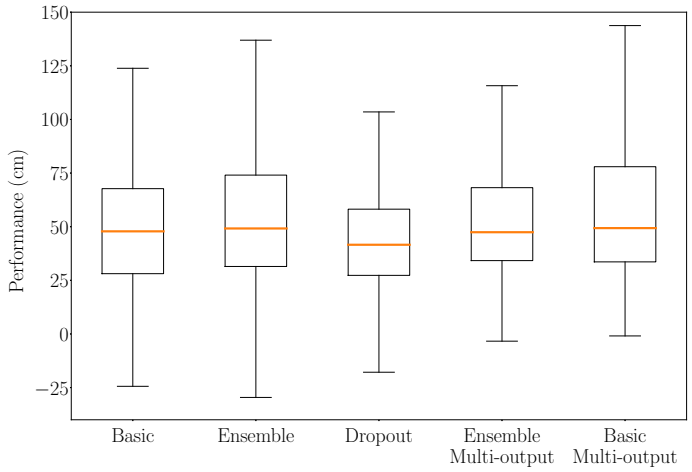


Figure 7.4: Performance comparison between simulator configurations

	Mean	**Median**	**Q_1**	**Q_3**	**Std. Dev.**
Basic	52.2	47.8	28.1	67.7	38.0
Ensemble	54.8	49.2	31.5	74.1	36.0
Dropout	43.5	41.6	27.3	58.2	24.2
Ensemble Multi-output	53.6	47.4	34.2	68.2	31.3
Basic Multi-output	*65.6*	*49.3*	33.6	78.0	55.2

Table 7.6: Summary Statistics for the simulator configuration performance distributions

7.5.1.2 Resetting Procedures

The performance distributions of the tested resetting procedures are illustrated in Figure 7.5 and summary statistics are given in Table 7.8. The Kruskal-Wallis H test is used to determine if performance distributions of the tested resetting procedures originate from the same distribution. The performance distributions were found to differ significantly from each other, with a p-value of 8.42×10^{-16}. The performance distributions of the

	Basic	**Ensemble**	**Dropout**	**E. Multi-output**
Ensemble	2.27×10^{-1}	-		
Dropout	1.61×10^{-2}	7.66×10^{-6}	-	
E. Multi-output	2.97×10^{-1}	7.99×10^{-1}	5.69×10^{-6}	-
Basic Multi-output	3.29×10^{-2}	2.00×10^{-1}	1.61×10^{-6}	1.46×10^{-1}

Table 7.7: The p-values of post hoc analysis comparing performance distributions between simulator configurations

tested resetting procedures are compared to each other using pairwise Mann-Whitney U tests (Table 7.9).

	Mean	**Median**	**Q_1**	**Q_3**	**Std. Dev.**
None	*61.5*	*52.3*	35.0	76.4	45.0
Controller	44.6	40.5	28.1	57.6	26.0
Simulator	59.3	52.0	33.4	77.8	40.1
Both	45.4	42.0	27.1	58.7	27.0

Table 7.8: Summary statistics for the resetting procedure performance distributions

	None	**Controller**	**Simulator**
Controller	7.54×10^{-11}	-	
Simulator	7.49×10^{-1}	1.73×10^{-9}	-
Both	1.87×10^{-9}	6.24×10^{-1}	2.07×10^{-8}

Table 7.9: The p-values of post hoc analysis comparing performance distributions between resetting procedures

The performance distributions do not appear particularly skewed towards either high or low values. The controller resetting procedure has a significantly worse performance distribution compared to the simulator resetting procedure. Periodically resetting only the simulator or not resetting anything obtained the best performance distributions. There is no significant performance difference between the simulator and no resetting procedures

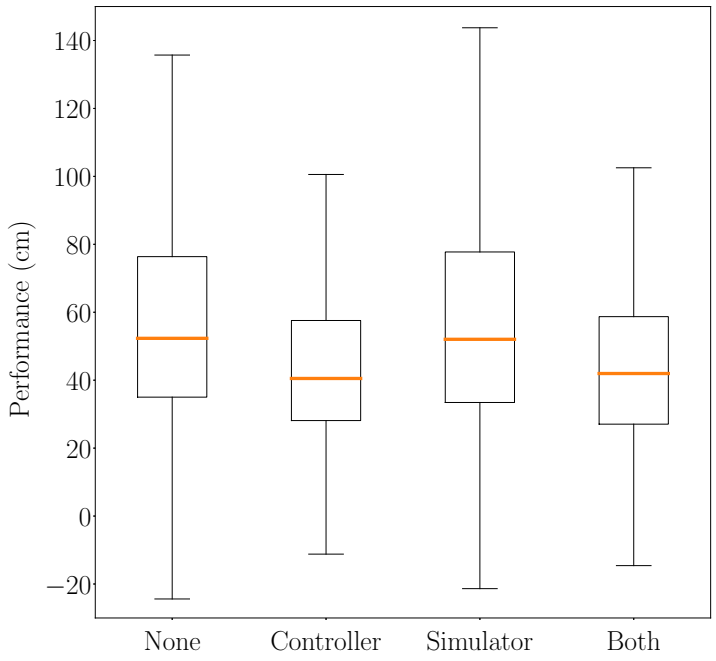


Figure 7.5: Performance comparison between resetting procedures

(p-value of 7.49×10^{-1}). Periodically resetting only the controller population and/or the simulator performs significantly worse compared to the other resetting procedures.

Not resetting controllers likely aids the ER process in exploiting good solutions. Controller resetting destroys good solutions before the ER process can further exploit them and the ER process cannot recover in time before the next reset. Simulator resetting modifies the fitness landscape but does not completely destroy existing good solutions. Not resetting or simply resetting the simulator preserves known good solutions over the lifetime of the BNS approach.

7.5.1.3 Simulator Noise

Performance distributions for the inclusion or exclusion of simulator noise are illustrated in Figure 7.6a and summary statistics given in Table 7.10. Not adding noise to simulated controller evaluations performs significantly better compared to trials including simulator

noise. The p-value for the Mann-Whitney U test is 3.39×10^{-47}.

	Mean	**Median**	**Q_1**	**Q_3**	**Standard Deviation**
Noise	40.6	39.1	25.7	54.0	22.2
Noiseless	*64.8*	*57.7*	37.5	82.3	43.0

Table 7.10: Summary statistics for the simulator noise performance distributions

Trial runs with simulator noise have a mean and median performance of 40.6 and 39.1 centimetres, respectively. Adaptations without simulator noise have a mean and median performance of 64.8 and 57.7 centimetres, respectively. Adding simulator noise can significantly complicate the fitness landscape and result in more exploration and less exploitation. Controller evolution may be unable to exploit known good solutions due to random changes in the fitness landscape. The median performance for noiseless solutions is higher than 75% of all noise injected solutions. However, noiseless adaptations have close to double the performance standard deviation compared to the noise injected adaptations. Simulator noise appears to improve the consistency in performance of discovered solutions but negatively affects likely performance.

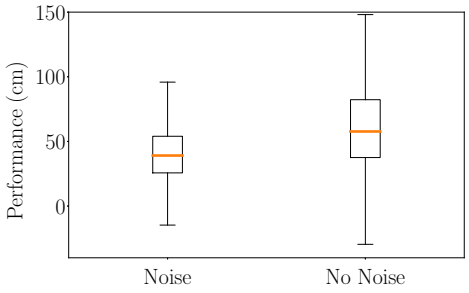
7.5.1.4 Sampling Strategies

Performance distributions of the tested sampling strategies is illustrated in Figure 7.6b and summary statistics are given in Table 7.11. No significant difference in performance is found between the **High Fitness** and **Most Uncertain** sampling strategies. The p-value for the Mann-Whitney U comparison test is 2.50×10^{-1}.

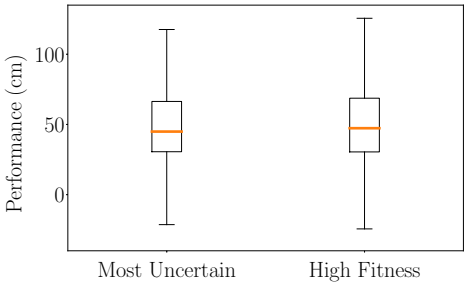
	Mean	**Median**	**Q_1**	**Q_3**	**Standard Deviation**
Most Uncertain	50.9	44.9	30.6	66.4	31.1
High Fitness	53.8	47.3	30.4	68.7	39.1

Table 7.11: Summary statistics for the sampling strategy performance distributions

The **High Fitness** sampling strategy has a slightly better mean and median performances. Little difference is observed between the sampling strategy IQR values. The **Most Uncertain** sampling strategy has a slightly lower standard deviation in performances.



(a) Performance distributions for simulator noise



(b) Performance distributions for sampling strategies

Figure 7.6: Performance distributions for simulator noise and sampling strategies

7.5.1.5 Summary

Performance distributions of the tested BNS adaptation settings are investigated and compared. Observations presented are achieved by benchmarking the proposed adaptations completely in a simulated environment. Results presented are based on high level aggregations over many different adaptation settings and may not indicate ideal adaptations settings for top performers.

The most influential adaptation setting for improving the likely performance outcomes of solution controllers is the exclusion of simulator noise. Not including simulator noise leads to a faster convergence in the ER process. The resetting procedure used is the second most important adaptation setting to consideration for improving performance. Using either simulator resetting or no resetting can significantly improve performance outcomes compared to the other tested procedures. The simulator configuration used is the third most important adaptation setting to consider. The **Basic Multi-output** and **Ensemble** simulator configurations produce the best performance outcomes. The tested sampling strategies do not significantly affect the likely performance of solutions.

7.5.2 Transferability

The transferability distributions of the tested adaptation settings are presented and discussed in this section. Each adaptation setting is grouped over all other adaptation settings and the transferability distributions are analysed. Adaptation settings investigated include

the Simulator Configuration (Section 7.5.2.1), Resetting Procedures (Section 7.5.2.2), Simulator Noise (Section 7.5.2.3) and the Sampling Strategy (Section 7.5.2.4). Finally, a summary of the transferability observations is given in Section 7.5.2.5.

7.5.2.1 Simulator Configurations

The transferability distributions of the tested simulator configurations are illustrated in Figure 7.7 and summary statistics are given in Table 7.12. Lower transferability values indicates a better correspondence between simulated and real-world solution trajectories. The Kruskal-Wallis H test is used to determine whether the tested simulator configuration transferability distributions originate from the same distribution. The p-value for the comparison is 3.19×10^{-30}. The transferability distributions are significantly different to each other.

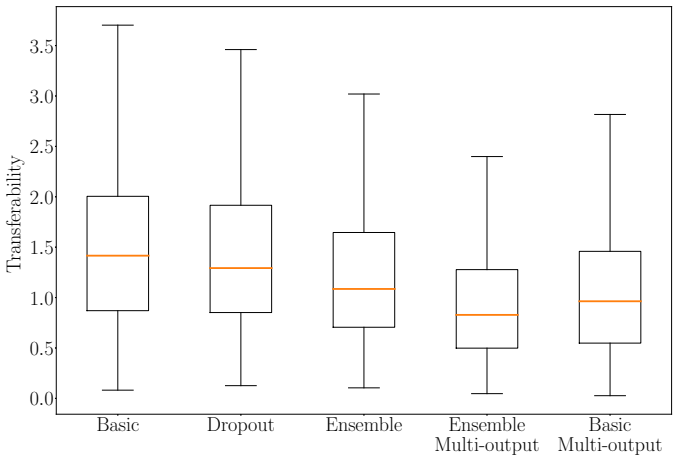


Figure 7.7: Transferability distributions for simulator configurations

A post hoc comparison of the simulator configuration transferability distributions is performed using pairwise Mann-Whitney U tests (Table 7.13). The **Basic** simulator configurations has a significantly worse transferability distribution compared to all other simulator configurations tested, except the **Dropout** simulator configuration. The **Basic** simulator configuration has the worst transferability distribution with an IQR between 0.87 and 2.00.

	Mean	**Median**	$\mathbf{Q_1}$	$\mathbf{Q_3}$	**Std. Dev.**
Basic	1.75	1.42	0.87	2.00	1.58
Dropout	1.56	1.29	0.85	1.92	1.15
Ensemble	1.31	1.09	0.71	1.65	0.96
Ensemble Multi-output	1.03	0.83	0.50	1.28	0.96
Basic Multi-output	1.13	0.96	0.55	1.46	0.87

Table 7.12: Transferability statistics for the simulator configurations

The **Ensemble Multi-output** simulator configuration has the best overall transferability compared to the other configurations tested, with an IQR between 0.50 and 1.28. The **Ensemble Multi-output** and **Basic Multi-output** simulator configurations are not significantly different from each other in terms of overall transferability with a p-value for the comparison being 6.06×10^{-2}. Simulator configurations consisting of multi-output SNNs demonstrate improved transferability properties compared to adaptations consisting of single-output SNNs. Additionally, grouping SNNs into ensembles also significantly improves the likely transferability of solution controllers.

	Basic	Dropout	Ensemble	E. Multi-output
Dropout	2.31×10^{-1}	-		
Ensemble	7.86×10^{-6}	4.32×10^{-5}	-	
E. Multi-output	9.28×10^{-20}	3.56×10^{-23}	3.54×10^{-10}	-
Basic Multi-output	6.99×10^{-10}	1.16×10^{-9}	2.57×10^{-3}	6.06×10^{-2}

Table 7.13: Transferability distribution p-values for comparisons between simulator configurations

7.5.2.2 Resetting Procedures

Transferability distributions of the tested resetting procedures is illustrated in Figure 7.8 and summary statistics are given in Table 7.14. The Kruskal-Wallis H test is used to determine if the transferability distributions for the tested resetting procedures originate from the same distribution. The p-value of the Kruskal-Wallis H test is 8.42×10^{-16}. The

transferability distributions of the tested resetting procedures are significantly different from each other.

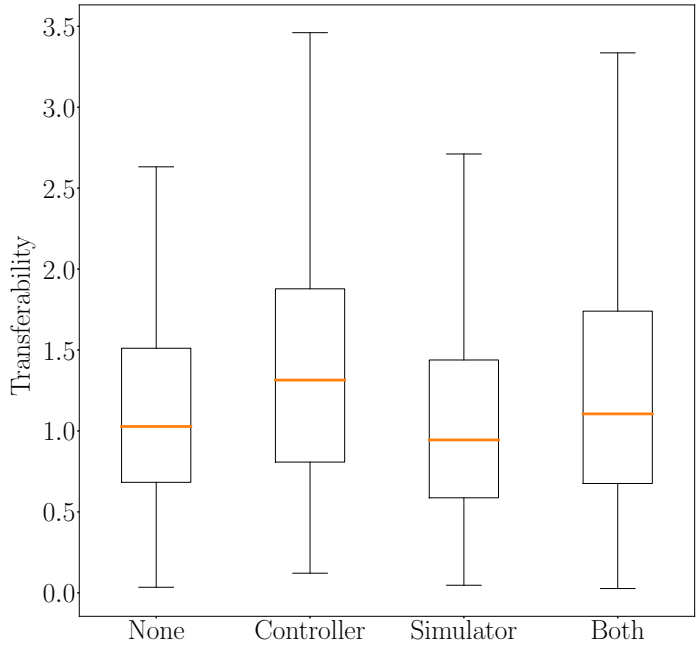


Figure 7.8: Transferability distributions for resetting procedures

	Mean	Median	Q_1	Q_3	Standard Deviation
None	1.25	1.03	0.68	1.51	1.04
Controller	1.52	1.31	0.81	1.88	1.12
Simulator	1.21	0.94	0.59	1.44	1.19
Both	1.37	1.11	0.68	1.74	1.10

Table 7.14: Transferability statistics for the resetting procedures

A post hoc comparison is performed between the tested resetting procedure transferability distributions (Table 7.15). The simulator resetting procedure has a significantly better transferability distribution compared to all other tested resetting procedures. Frequent simulator resets reduces the likelihood that the ER process will exploit weaknesses

in the developing simulator, leading to better transferability properties. The controller resetting procedure has a significantly worse transferability distribution compared to all other resetting procedures. Periodic controller resetting reduces the ability of the ER process to exploit good controller solutions and candidate solutions likely differ significantly between resets. The ER is less able to recover from frequent controller resets. The no resetting procedure and resetting both (simulator and controller population) do not have significantly different transferability distributions.

	None	**Controller**	**Simulator**
Controller	2.51×10^{-6}	-	
Simulator	3.54×10^{-2}	1.31×10^{-10}	-
Both	1.00×10^{-1}	4.80×10^{-3}	4.31×10^{-4}

Table 7.15: Transferability distribution p-values for comparisons between resetting procedures

7.5.2.3 Simulator Noise

The transferability distributions for trial runs that include or exclude of simulator noise are illustrated in Figure 7.9a and summary statistics are given in Table 7.16. For all BNS trial runs that include simulator noise, the mean and median transferability metrics are 1.29 and 1.00, respectively. For noiseless trial runs, the mean and median transferability is 1.38 and 1.18, respectively. Approaches that include simulator noise have a significantly better transferability distribution. The p-value for the Mann-Whitney U comparison test is 3.39×10^{-47}. The addition of simulator noise improves the likely transferability of solutions for the BNS approach when applied to the Snake robot morphology whereas the transferability worsens when applied to the Hexapod robot. The Snake robot simulators are relatively large in terms of the number of hidden layers and layer sizes. For the Snake robot, the ER process is probably more prone to exploiting simulator weaknesses compared to the Hexapod robot.

The point of simulator noise is intended to prevent the ER process from exploiting idiosyncrasies of the simulator not accurately representing reality. For the BNS approach,

	Mean	Median	Q_1	Q_3	Standard Deviation
Noise	1.29	1.00	0.64	1.54	1.21
Noiseless	1.38	1.18	0.71	1.73	1.02

Table 7.16: Transferability statistics for simulator noise

the simulator continually changes over time due to the constant integration of new training data. It can be argued that a constantly changing simulator encourages the ER process to pursue less brittle solutions.

7.5.2.4 Sampling Strategies

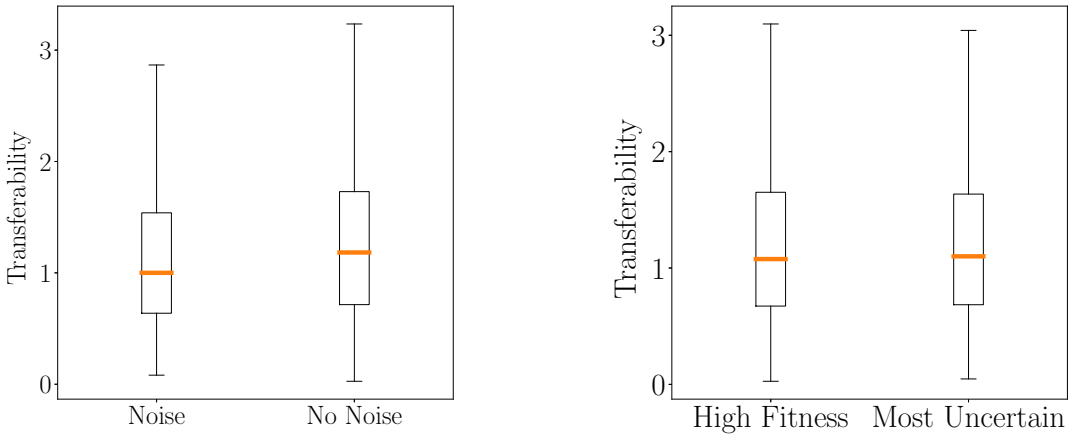
The observed transferability distributions of the tested sampling strategies is illustrated in Figure 7.9b and summary statistics are given in Table 7.17. The sampling strategies have almost equal mean, median and IQR values. The difference between the transferability standard deviations is relatively small. The p-value for the Mann-Whitney U test comparing the transferability distributions is 0.25. The tested sampling strategies do not have significantly different transferability distributions.

	Mean	Median	Q_1	Q_3	Standard Deviation
High Fitness	1.34	1.08	0.67	1.65	1.16
Most Uncertain	1.33	1.10	0.68	1.64	1.05

Table 7.17: Transferability statistics for the sampling strategies

7.5.2.5 Summary

The **Ensemble Multi-output** simulator configuration improves the likely transferability of solution controllers more than any other adaptation setting tested. Periodically resetting the simulator configuration is the second most influential adaptation setting for improving the transferability of solution controllers. Not periodically resetting the controller population or simulator is also a good choice for improving the transferability of solutions. Due to the slow rate at which generations are progressed, any adaptation settings that resets the controller population cannot recover quickly enough before the next reset. Solution



(a) Transferability distributions for simulator noise

(b) Transferability distributions for sampling strategies

Figure 7.9: Transferability distributions for simulator noise and sampling strategies

controllers evolved from adaptations that include simulator noise are likely to have better transferability properties compared to adaptations without simulator noise. The tested sampling strategies do not affect the likely transferability of solution controllers.

7.5.3 Convergence Properties

For all BNS trial runs, the fittest controller after every sampling evaluation is recorded. Trial runs are grouped according to the resetting procedure used. The median performance over time for each resetting procedure is calculated and illustrated in Figure 7.10. Controllers are evaluated using the Static SNNs as a "real-world" surrogate and the Dynamic SNNs produce simulated behaviours.

The median performance for each resetting procedure is poor during the early stages of the BNS approach but gradually improve over time. The median performances improves up until the 10^{th} sampling evaluation, after which the median performance drastically drops for resetting procedures including periodic controller resets. Controller/simulator resetting takes place every 10 sampling evaluations. Resetting the controller population with or without simulator resetting appears to result in a significant drop in performance

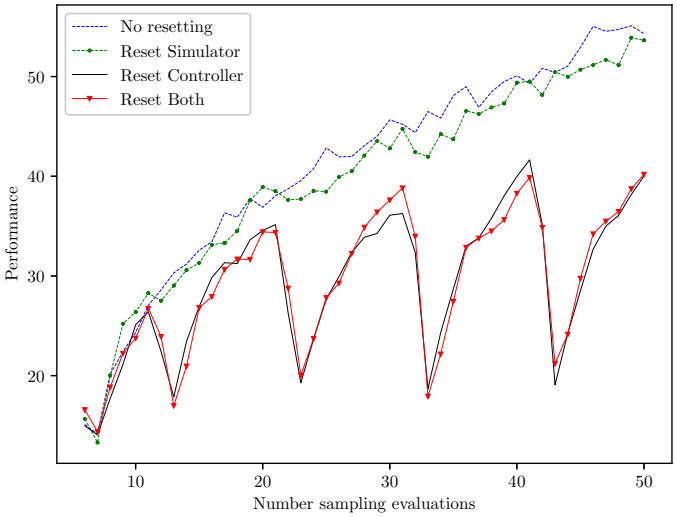


Figure 7.10: Performance over time for the best performer of each resetting procedure

every 10 sampling evaluations. The no resetting or simulator resetting procedures do not result in any drastic drops in performance. Past the 20^{th} sampling evaluation, the median performance of the no resetting and simulator resetting procedures increase almost linearly over time.

No resetting appears to perform marginally better over time compared to the simulator resetting procedure. Resetting procedures that include periodic controller resetting (Reset Controller, Reset Both) perform almost equally worse than the other tested resetting procedures procedures.

7.5.4 Top Performers

The 64 tested adaptations are ordered in descending order of performance and the top 10 best performing adaptations are presented in this section. Performance and transferability statistics of each tested adaptation is given in Appendices B.11 and B.12, respectively. The performance distributions of the 10 best performing BNS adaptations are illustrated in Figure 7.11 and summary statistics are given in Table 7.18. The top adaptations are compared using pairwise Mann-Whitney U tests (Table 7.19). For top performing adaptations, mean performances are between 75.2 and 130.7 centimetres and median performances are

between 66.0 and 113.5 centimetres. None of the best performing adaptations use the controller resetting procedure. All best performing adaptations do not include simulator noise.

The SSNET adaptation achieved the highest mean and median performance at 130.7 and 113.5 centimetres, respectively. The SSNET adaptation consists of a noiseless **Basic Multi-output** simulator configuration and used the no resetting procedure. The **Basic Multi-output** simulator configuration is the most computationally efficient simulator configuration. The BNS approach is able to process many more controller generations using the **Basic Multi-output** simulator configuration compared to the other configurations tested. The SSNET adaptation does have the highest standard deviation in performance. A Levene's statistical test for assessing equality of variance is used to compare the performance variances between the SSNET and SESEU adaptations. The SESEU adaptation is the 2^{nd} best performing BNS adaptation for the Snake robot. The Levene's statistical test achieved a p-value of 5.86×10^{-3} which indicates that the SSNET adaptation has a significantly higher performance variance compared to the SESEU adaptation.

	Mean	**Median**	$\mathbf{Q_1}$	$\mathbf{Q_3}$	**Std. Dev.**
SSNET	*130.7*	*113.5*	68.2	172.6	89.7
SESEU	98.6	97.0	76.4	130.5	44.1
SESET	87.3	84.7	73.0	117.5	50.4
SSSET	95.7	78.5	59.6	115.3	65.1
SENEU	84.7	88.1	65.8	110.2	41.2
SBNET	88.5	74.0	50.9	117.8	54.7
SENET	80.7	83.4	57.6	101.4	41.0
SMSET	82.2	73.8	56.7	112.5	41.8
SMSEU	78.0	78.6	52.0	103.0	35.7
SMNEU	75.2	66.0	38.9	105.8	42.0

Table 7.18: Performance distributions of the top 10 adaptations for the BNS approach

The 2^{nd} and 3^{rd} best performing adaptations (SESEU and SESET) consist of noiseless **Ensemble** simulator configurations and make use of simulator resetting. These two

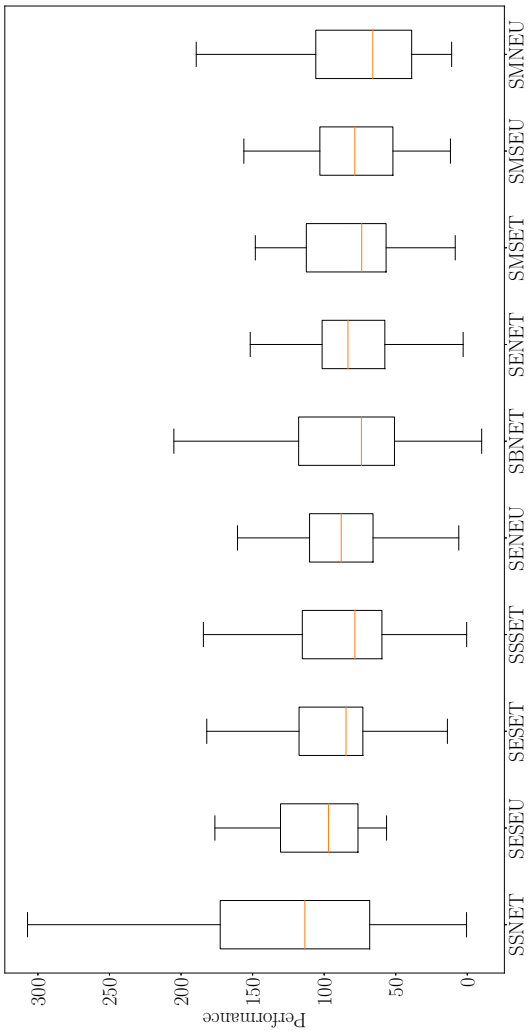


Figure 7.11: Performance distributions of the top 10 adaptations for the BNS approach

	SSNET	**SESEU**	**SESET**	**SSSET**	**SENEU**	**SBNET**	**SENET**	**SMSET**	**SMSEU**
SESEU	4.73×10^{-1}	-							
SESET	2.23×10^{-1}	3.87×10^{-1}	-						
SSSET	1.02×10^{-1}	1.49×10^{-1}	4.73×10^{-1}	-					
SENEU	7.01×10^{-2}	2.12×10^{-1}	6.52×10^{-1}	7.84×10^{-1}	-				
SBNET	4.84×10^{-2}	1.37×10^{-1}	4.73×10^{-1}	8.07×10^{-1}	8.19×10^{-1}	-			
SENET	3.15×10^{-2}	6.79×10^{-2}	3.11×10^{-1}	8.65×10^{-1}	5.30×10^{-1}	8.65×10^{-1}	-		
SMSET	3.15×10^{-2}	4.68×10^{-2}	2.46×10^{-1}	6.63×10^{-1}	4.64×10^{-1}	8.42×10^{-1}	9.82×10^{-1}	-	
SMSEU	2.61×10^{-2}	3.39×10^{-2}	1.86×10^{-1}	6.20×10^{-1}	3.79×10^{-1}	7.17×10^{-1}	8.65×10^{-1}	8.30×10^{-1}	-
SMNEU	9.07×10^{-3}	1.38×10^{-2}	8.24×10^{-2}	2.84×10^{-1}	2.23×10^{-1}	3.40×10^{-1}	4.64×10^{-1}	4.83×10^{-1}	6.73×10^{-1}

Table 7.19: Performance comparisons between the top 10 adaptations for the BNS approach

adaptations are close variations on each other. The only difference between the SESEU and SESET adaptations is the sampling strategy used.

The 4^{th} best performing adaptation (SSSET) is a variation on the SSNET adaptation but with simulator resetting. The SSSET adaptation consists of a noiseless **Basic Multi-output** simulator configuration and uses simulator resetting. The standard deviation in performance is the second highest amongst the top adaptations.

The 5^{th} and 7^{th} best performing adaptations (SENEU and SENET) consist of noiseless **Ensemble** simulator configurations and use the no resetting procedure. These two adaptations are variations on each other. The SENEU adaptation uses the **Most Uncertain** sampling strategy and SENET uses the **High Fitness** sampling strategy.

The SESEU, SESET, SENEU and SENET adaptations all consist of noiseless **Ensemble** simulator configurations. These adaptations are amongst the best performing adaptations even though the **Ensemble** simulator configuration is the most computationally expensive configuration. The SESEU and SESET adaptations use simulator resetting while the SENEU and SENET adaptations use the no resetting procedure.

The 6^{th} best adaptation (SBNET) uses the noiseless **Basic** simulator configuration and no resetting. Compared to the best performing adaptation (SSNET), the 6^{th} (SBNET) up to the 10^{th} (SMNEU) best adaptations perform significantly worse.

The 8^{th} and 9^{th} best performing adaptations (SMSET and SMSEU) consist of noiseless **Ensemble Multi-output** simulator configurations and use simulator resetting. The SMSET adaptation uses the **High Fitness** sampling strategy and the SMSEU adaptation uses the **Most Uncertain** sampling strategy. The 10^{th} best performing adaptation (SMNEU) consist of a noiseless **Ensemble Multi-output** simulator configuration and uses the no resetting procedure. The 2^{nd} best adaptation (SESEU) has a significantly better performance distribution compared to the 8^{th} (SMSET) up to 10^{th} (SMNEU) best adaptations.

For the best adaptations, the ER process is better able to exploit known good solution in the absence of simulator noise. Half the top performing adaptations use the simulator resetting procedure and the other half use the no resetting procedure. Adaptations using periodic controller resetting do not make it into the top 10 best performing adaptations.

The SSNET and SSSET adaptations perform disproportionately well due to the BNS

approach exploiting weaknesses in the Static Simulator. The SSNET and SSSET adaptations have a higher controller evolution rate compared to other adaptations, such as the SESEU and SESET adaptations. For the Snake robot, exploiting simulator weaknesses is a general problem encountered with certain adaptations. This problem was not seen for the Hexapod robot simulators. A density versus performance plot of the top four best performing BNS adaptations in this section are illustrated in Figure 7.12. This density plot demonstrates that the SSNET and SSSET adaptations achieve disproportionately high performances for many trial runs compared to the SESEU and SESET adaptations. Performances values greater than 200 centimetres are considered unrealistic in terms of real-world performance. As an example, the real-world, Dynamic Simulated and Static Simulated trajectories produced by the best performing SSNET adaptation is given in Figure 7.13. The Dynamic Simulator appears to have simulated a weakness in the Static Simulator that would not be possible in reality. The solution controller is evaluated on the real-world robot and the real-world trajectory is included for reference purposes. The Static Simulator overestimates the real-world distance travelled by just over 300%. The simulated performance outcomes for the SSNET and SSSET adaptations should be considered less reliable estimates of real-world performance compared to other BNS adaptations.

As previously mentioned, a Static Simulator is used as an alternative to a real-world robot during the Simulated BNS Experimental work. During the Simulated BNS Experimental work, simulator noise is always added to sampling evaluations produced through the Static Simulator, however, exploitation of weaknesses in the Static Simulator can still take place. Exploitations of the Static Simulator during these simulated experiments is a problem mainly seen for certain BNS adaptations applied to the Snake robot. Adaptations using the **Basic Multi-output** simulator configuration for the Snake robot are most affected by this issue. This research is cognisant of the fact that this is a flaw in the methodology and conclusions are drawn accordingly.

The transferability distributions for the top 10 best performing BNS adaptations are illustrated in Figure 7.14 and summarised in Table 7.20. The p-values for the Mann-Whitney U pairwise comparisons are given in Table 7.21.

The SMSEU adaptation achieved the best transferability distribution compared to the other best performing adaptations and had significantly better transferability compared

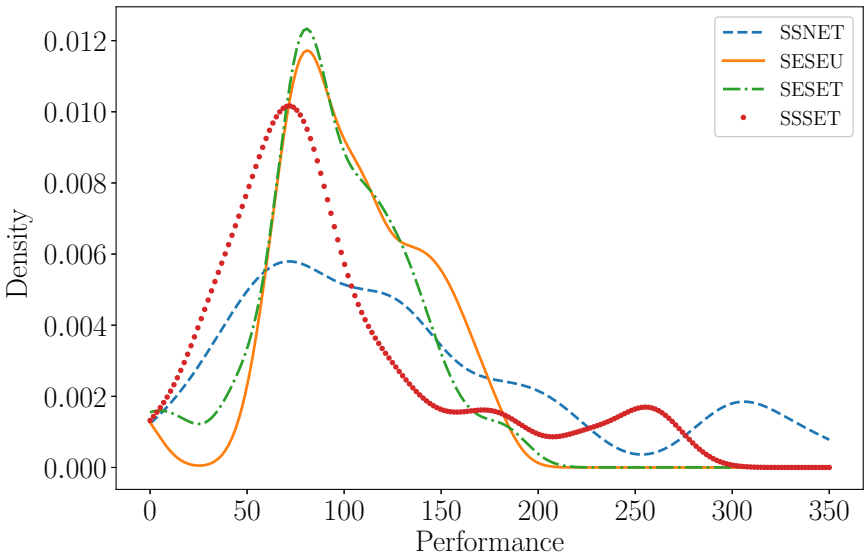


Figure 7.12: Density of performances for best performing adaptations

to the SESET, SENEU, SBNET, SENET and SMNEU adaptations. The SMSET and SSSET adaptations demonstrated particularly good transferability. The SBNET adaptation has a significantly worse transferability distribution compared to all other top performing adaptations. Solution controllers for the SSNET adaptation are observed to have relatively good median transferability properties compared to the other top performing adaptations. However, the SSSET adaptation does not have a consistently good transferability compared to the SMSET adaptation. Top adaptations using simulator resetting tend to have better overall transferability distributions compared to corresponding adaptations using the no resetting procedure.

7.5.5 Summary

The overall performance and transferability properties of the tested adaptation settings are investigated. Adaptations consisting of **Basic Multi-output**, **Ensemble** or **Ensemble Multi-output** simulator configurations have better overall performance and transferability distributions compared to adaptations using **Basic** or **Dropout** simulator configurations. The simulator resetting and no resetting procedures have equally good performance properties. However, adaptations using simulator resetting have better transferability

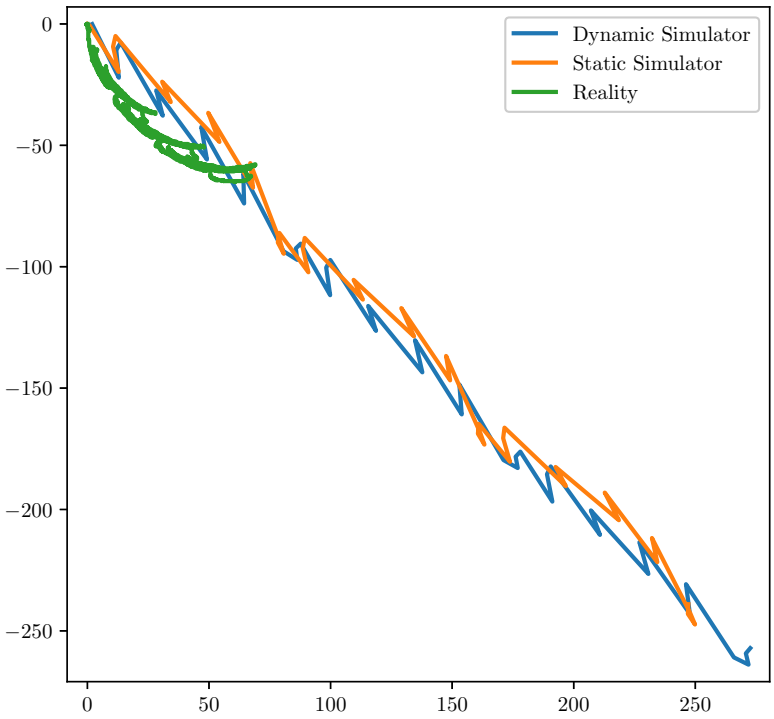


Figure 7.13: Best solution for the SSNET adaptations

properties. Noiseless adaptations are more likely to perform better than noisy adaptations. Noisy adaptations demonstrate better transferability properties compared with noiseless counterparts. The tested sampling strategies do not significantly affect the likely performance or transferability of solution controllers.

The top five best performing adaptations consist of either a **Basic Multi-output** or **Ensemble** simulator configuration. All top performing adaptations exclude simulator noise. Amongst the top performing adaptations, the **Basic** simulator configurations is observed to have the worst transferability. No top performing adaptations consist of a **Dropout** simulator configuration.

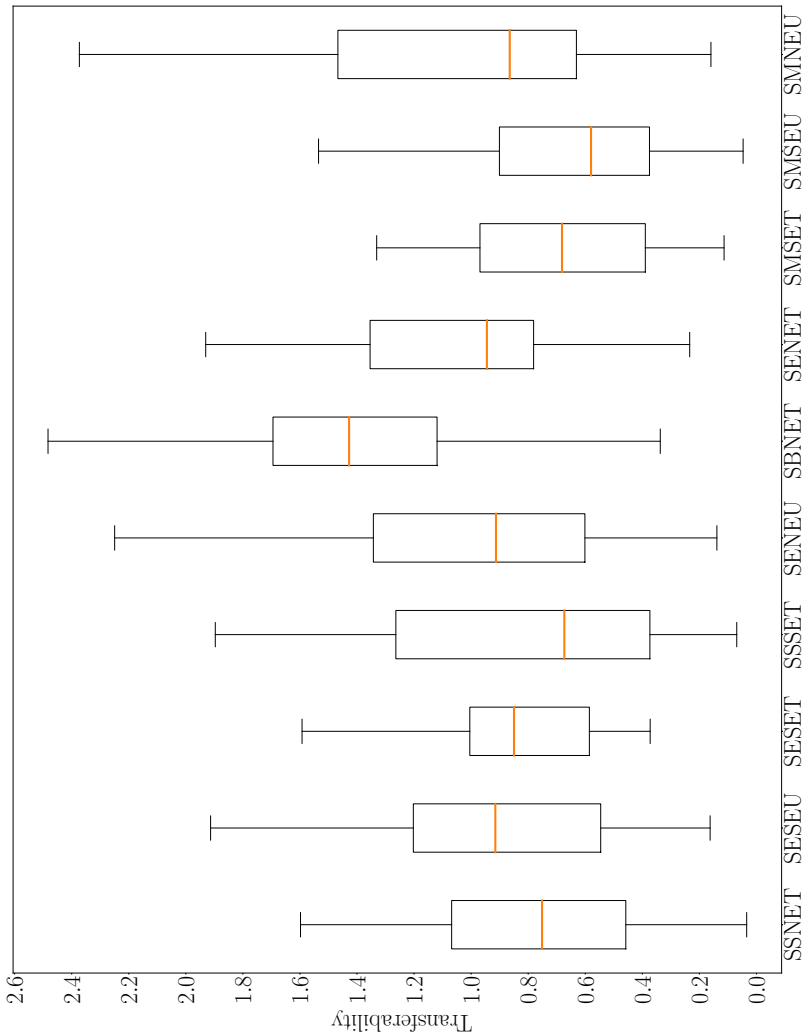


Figure 7.14: Transferability distributions for the top 10 BNS adaptations

	Mean	**Median**	**Q_1**	**Q_3**	**Std. Dev.**
SSNET	0.89	0.75	0.46	1.07	0.82
SESEU	0.94	0.92	0.55	1.20	0.54
SESET	0.93	0.85	0.59	1.00	0.47
SSSET	0.84	0.67	0.37	1.26	0.57
SENEU	1.11	0.91	0.60	1.34	0.78
SBNET	1.49	1.43	1.12	1.69	0.77
SENET	1.11	0.94	0.78	1.35	0.55
SMSET	0.74	0.68	0.39	0.97	0.57
SMSEU	0.79	0.58	0.38	0.90	0.81
SMNEU	1.02	0.86	0.63	1.47	0.60

Table 7.20: The transferability statistics for the top 10 adaptations for the BNS approach

	SSNET	SESEU	SESET	SSSET	SENEU	SBNET	SENET	SMSET	SMSEU
SESEU	4.12×10^{-1}	-							
SESET	3.79×10^{-1}	8.88×10^{-1}	-						
SSSET	9.00×10^{-1}	5.49×10^{-1}	5.69×10^{-1}	-					
SENEU	1.33×10^{-1}	5.69×10^{-1}	5.30×10^{-1}	2.17×10^{-1}	-				
SBNET	1.58×10^{-4}	1.44×10^{-3}	4.22×10^{-4}	6.55×10^{-4}	1.50×10^{-2}	-			
SENET	4.06×10^{-2}	2.23×10^{-1}	9.05×10^{-2}	7.98×10^{-2}	5.69×10^{-1}	2.07×10^{-2}	-		
SMSET	4.29×10^{-1}	7.48×10^{-2}	4.68×10^{-2}	3.79×10^{-1}	1.70×10^{-2}	4.74×10^{-6}	1.44×10^{-3}	-	
SMSEU	2.97×10^{-1}	5.94×10^{-2}	2.92×10^{-2}	4.73×10^{-1}	1.76×10^{-2}	2.43×10^{-5}	1.37×10^{-3}	8.19×10^{-1}	-
SMNEU	1.96×10^{-1}	6.95×10^{-1}	7.28×10^{-1}	1.91×10^{-1}	8.88×10^{-1}	1.56×10^{-2}	5.01×10^{-1}	4.84×10^{-2}	4.51×10^{-2}

Table 7.21: Transferability comparisons between the top 10 adaptations for the BNS approach

7.6 The BNS Validation Experiment Results

The purpose of this section is to demonstrates that the BNS approach can effectively be applied to a real-world Snake robot. Promising BNS adaptations are selected based on the Simulated BNS Experimental results. The most promising adaptations selected for validation are the SBSET, SESEU and SMSEU adaptations. These adaptations are selected due to their simulated high performances and in order to validate significantly different simulator configurations. A limited number of real-world experiments are possible and validation adaptations are carefully chosen based on better likely performance/transferability outcomes and in order to validate a diverse set of adaptation settings.

The first and fourth best performing adaptations (SSNET, SSSET) are not considered due to their simulated performances being deemed unreliable due to the exploitation of Static Simulator weaknesses. The second best performing adaptation in the Simulated BNS Experimental results is the SESEU adaptation chosen for validation. The third best adaptation (SESET) is essentially equivalent to the SESEU but using a different sampling strategy. No significant difference between the tested sampling strategies was found in the simulated results. The fifth up to seventh best performing adaptations (SENEU, SBNET, SENET) are avoided due to their use of the no resetting procedure. Simulated experiments indicate that adaptations using the simulator resetting procedure tend to produce solutions with better likely transferability properties compared to other resetting procedures. The eight and ninth best performing adaptations (SMSET, SMSEU) are essentially the same but use different sampling strategies. the SMSEU adaptation is chosen in order to correspond with the HMSEU adaptation chosen for the Hexapod validation work. For the Hexapod validation experiments, the HBSNT adaptation performed unexpectedly better than expected. In order to validate if this might be the case for the Snake robot, the SBSET adaptation is purposely chosen in order to investigate if its noisy variation too demonstrates better than expected performance in validation experiments. The SBSET adaptation is the fifteenth best performing adaptation in the simulated experimental work.

The corresponding adaptations with simulator noise are also selected for validation. Noise injected variations are validated in order to validate the effect that noise has when applied in reality. For similar reasons stated for the Hexapod validation experiments, the

use of simulator noise is validated. The noise injected adaptations are SBSNT, SESNU and SMSNU. The SBSET and SBSNT adaptations use the **Basic** simulator configuration with simulator resetting and the **High Fitness** sampling strategy. The SESEU and SESNU adaptations used the **Ensemble** simulator configuration with simulator resetting and the **Most Uncertain** sampling strategy. The SMSEU and SMSNU adaptations use the **Ensemble Muti-output** simulator configuration with simulator resetting and the **Most Uncertain** sampling strategy. The best performing Simulated BNS adaptation, SSNET, is not chosen to be validated. Only the top performing adaptations, common to both robot morphologies, are considered for validation and comparison purposes.

The real-world performances of validated adaptations are covered in Section 7.6.1. Real-world transferability of validated adaptations are presented in Section 7.6.2. The real-world and simulated trajectories of solution controllers are presented in Section 7.6.3. Lastly, a summary is given in Section 7.6.4.

7.6.1 Performance

Performances achieved in trial runs are provided in Table B.7. The performance distributions for the validated adaptations are illustrated in Figure 7.15 and summary statistics are given in Table 7.22. Each adaptations is validated by performing 5 independent trial runs of the BNS approach on a real-world Snake robot. The median performance over all validation trial runs is 46.0 centimetres with an IQR between 31.5 and 68.6 centimetres.

The SESEU adaptation achieved the highest median performance of 78.5 centimetres. The SMSNU adaptation has the second best median performance of 61.5 centimetres. The SBSET and SMSEU adaptations have median performances of 49.4 and 49.2 centimetres, respectively. The SBSNT and SESNU adaptations obtained the worst median performances at 39.5 and 33.7 centimetres, respectively. Outlier trial runs are seen for the SBSNT, SMSEU and SMSNU adaptations. The SBSNT adaptation does not perform unexpectedly better than the SBSET adaptation. This provides some evidence that the unexpectedly high performance seen with the HBSNT adaptation for the Hexapod robot might have been a statistical anomaly.

Experimental trial runs are separated into groups either including or excluding simulator noise. The performance distributions for the noise injected and noiseless simulator

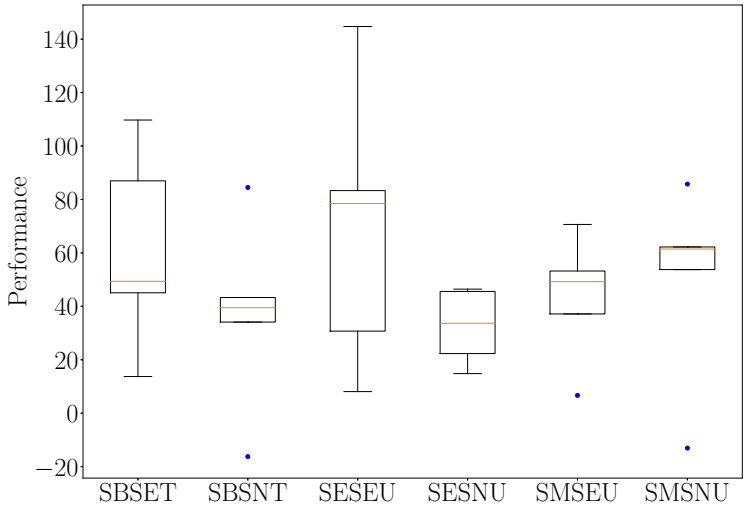


Figure 7.15: Validation experiment performance distributions

configuration solutions are illustrated in Figure 7.16 and summary statistics are given in Table 7.23. Trial runs using a noiseless simulator have a median performance of 49.4 centimetres and an IQR between 33.9 and 80.9 centimetres. Trial runs using simulator configurations with noise achieve a median performance of 43.3 centimetres and an IQR between 28.0 and 57.6 centimetres. In general, solutions developed using noiseless simulator configurations performed better than solutions produced from simulators with noise.

For the 30 validation solutions, 7 solution controllers are considered poor performers where performance measurements are less than 30 centimetres. Performances greater than 60 centimetres are considered excellent. Ten validation solution controllers demonstrated excellent performance outcomes. The remaining 13 validation solutions achieved acceptable performance values between 30 and 60 centimetres. Half of the validated solutions achieve a performance measure greater than 46 centimetres.

7.6.2 Transferability

Transferability values achieved in trial runs are provided in Table B.8. The transferability distributions for the validated BNS adaptations are illustrated in Figure 7.17 and summary statistics are given in Table 7.24. A lower transferability metric corresponds to a

	Mean	Median	$\mathbf{Q}_1$	$\mathbf{Q}_3$	Standard Deviation
SBSET	61.0	49.4	45.1	87.0	37.7
SBSNT	37.0	39.5	34.1	43.3	35.9
SESEU	69.1	78.5	30.7	83.3	52.9
SESNU	32.6	33.7	22.3	45.6	14.0
SMSEU	43.4	49.2	37.2	53.2	23.8
SMSNU	50.0	61.5	53.8	62.2	37.3

Table 7.22: BNS performance statistics on validation results

	Mean	**Median**	$\mathbf{Q}_1$	$\mathbf{Q}_3$	**Standard Deviation**
Noiseless	57.8	49.4	33.9	80.9	38.6
Noise	39.9	43.3	28.0	57.6	29.6

Table 7.23: BNS validation performance statistics grouped by simulator noise

closer correspondence between simulation and reality. The median transferability over all validated trial runs is 0.94 with an IQR between 0.54 and 2.44. Adaptations SBSET, SBSNT, SMSEU and SMSNU each have transferability outliers.

The SMSEU and SMSNU adaptations have the best median transferability values at 0.49 and 0.58, respectively. The second least transferable adaptation (SESEU) has an IQR between 0.94 and 2.59 and the least transferable adaptation (SBSET) has an IQR between 2.39 and 3.20. The SBSET adaptation demonstrates drastically worse validation transferability outcomes compared to the other validated adaptations. The reason for relatively high transferability values for the SBSET adaptation is due to the greatly overestimated simulated distances travelled. The transferability IQR for the SBSNT and SESNU adaptations are similar to each other.

For the SBSET, SBSNT, SESEU, SESNU adaptations, the noise injected adaptations (SBSNT and SESNU) demonstrate better transferability compared to the noiseless adaptations (SBSET and SESEU). The SMSEU and SMSNU adaptations have relatively similar transferability distributions for the simulated results. Simulator noise appears to improve the likely transferability of solutions for the validated results.

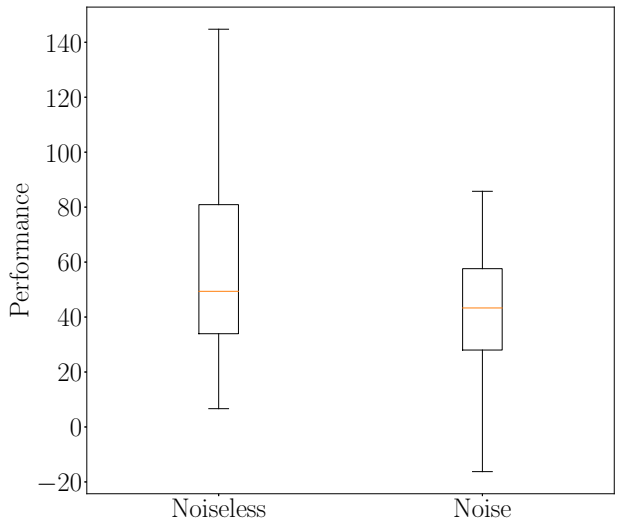


Figure 7.16: BNS validation performance distributions grouped by simulator noise

The validated trial solutions are grouped according to their inclusion or exclusion of simulator noise. The transferability distributions of the noise injected and noiseless simulator configurations are illustrated in Figure 7.18 and summary statistics are given in Table 7.25. The mean and median transferability of validated trial runs without simulator noise is 2.34 and 1.13, respectively, with an IQR between 0.60 and 2.90. Noise injected trial run solutions have mean and median transferability values of 1.34 and 0.70, respectively, with an IQR between 0.49 and 1.29. Including simulator noise greatly improves the likely transferability of solution controllers. This improvement is achieved because including simulator noise greatly reduces the level of overestimation in simulated distances travelled. The standard deviation of transferability for noiseless solutions is 2.70 while noise injected solutions demonstrated a transferability standard deviation of 1.78.

7.6.3 Validation Solutions

The simulated and real-world trajectories of solution controllers are presented in Figures 7.19 to 7.24. The SNS and BNS simulated paths generated from evaluating the solution controllers are presented for comparison purposes. The green solid line represents the

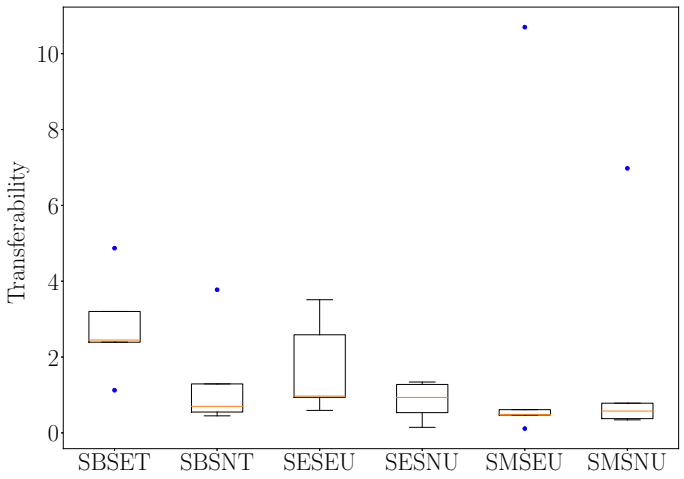


Figure 7.17: Validation experiment transferability distributions

trajectory followed by the tracked head of the Snake robot for the particular controller evaluation. The dashed blue and orange lines represent the BNS and SNS simulated trajectories, respectively. For most solutions, the BNS simulator demonstrates either better or similar transferability compared to the SNS simulator. All real-world trajectories move in the positive x-direction with varying degrees of positive or negative y-direction displacements.

Figure 7.19 illustrates solution controllers for the SESEU adaptation. The first, second and fourth trial runs have relatively similar simulated and real-world trajectory behaviours. The fourth solution moves in an upward trajectory while the first and second solutions have downward trajectories. The BNS simulator estimates the general directions of real-world trajectories relatively well but the simulated and real-world curvatures and headings diverge over time. The third and fifth trail runs demonstrate poor performance and transference with the BNS simulator significantly overestimating the real-world distances travelled.

For the SESNU adaptation, the real-world and simulated trajectories are illustrated in Figure 7.20. The first trial run produces a downward trajectory for the BNS simulator and real-world robot but the simulator overestimates the total real-world distance travelled. The Static simulator is unable to predict the direction of the real-world trajectory for the

	Mean	Median	$\mathbf{Q}_1$	$\mathbf{Q}_3$	Standard Deviation
SBSET	2.81	2.45	2.39	3.20	1.37
SBSNT	1.35	0.70	0.55	1.29	1.39
SESEU	1.72	0.97	0.94	2.59	1.27
SESNU	0.85	0.94	0.53	1.28	0.51
SMSEU	2.48	0.49	0.47	0.61	4.60
SMSNU	1.81	0.58	0.38	0.78	2.89

Table 7.24: BNS transferability statistics on validation results

	Mean	**Median**	$\mathbf{Q}_1$	$\mathbf{Q}_3$	**Standard Deviation**
Noiseless	2.34	1.13	0.60	2.90	2.70
Noise	1.34	0.70	0.49	1.29	1.78

Table 7.25: BNS validation transferability statistics grouped by simulator noise

first solution. The second and fourth trial runs have similar trajectories and relatively good correspondence between the real-world and BNS simulated trajectories but the final heading for the second solution is poorly estimated. The third trial solution is inaccurately estimated by the BNS simulator in terms of the overall direction of the trajectory but the final heading is accurately predicted. For the fifth trial solution, the BNS simulator correction estimates the upward trajectory and final heading but the distance travelled is overestimated.

Figure 7.21 illustrates trajectories for solutions produced using the SMSEU adaptation. For the first trial run, the BNS simulator greatly overestimates the distance travelled for the downward trajectory. The first solution's performance is poor and the real-world distance travelled is small. The Static simulator is also unable to simulate the real-world trajectory of the first trial run. The second, third and fifth trial runs are effective and relatively accurately estimated by the BNS simulator. The third and fifth trial runs are accurately simulated in terms of the final headings but the second solution heading is not accurately predicted. The BNS simulator predicts the real-world upward trajectory of the fourth trial run relatively well but the simulated and real-world final headings are off by

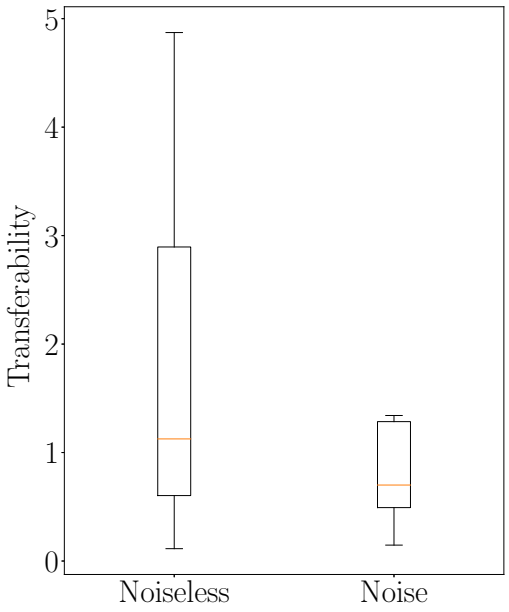
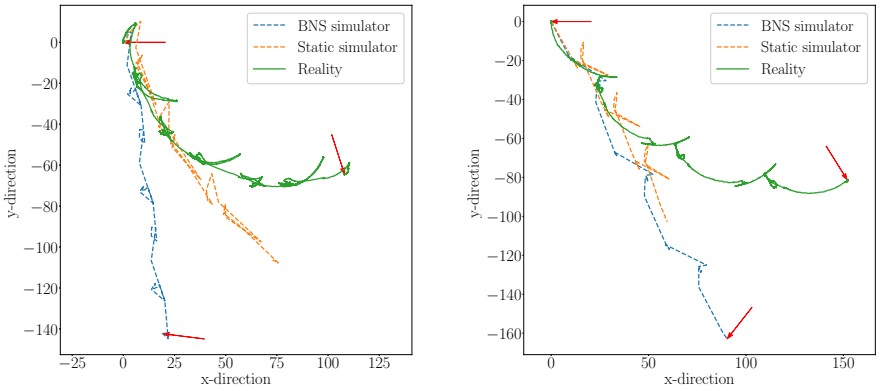


Figure 7.18: BNS validation transferability statistics grouped by noise

almost 90 degrees.

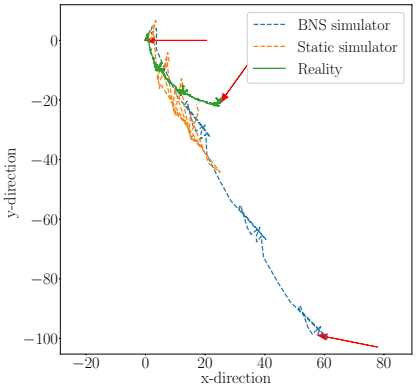
For the SMSNU adaptation, the real-world and simulated trajectories of solution controllers are illustrated in Figure 7.22. For the first solution, the BNS simulator accurately predicts that the trajectory moves towards the bottom right but the final predicted heading is not accurately estimated. The second and fifth trial runs demonstrate excellent transference from the BNS simulator into reality. The second run is accurately simulated in terms of the distance travelled and final heading. The BNS simulated distance for the fifth trial run is overestimated. The third trial solution demonstrates poor performance and the BNS and Static simulators are unable to accurately predict the real-world trajectory. For the fourth solution, the BNS simulator accurately predicts the distance travelled but is unable to account for the slight changes in the robot's heading over time which accumulates to create a curved trajectory.

The SBSET adaptation solution controllers are illustrated in Figure 7.23. The first trial run has a trajectory moving a significant distance towards the right. The BNS simulator poorly estimates the distance travelled and the final trajectory heading is off

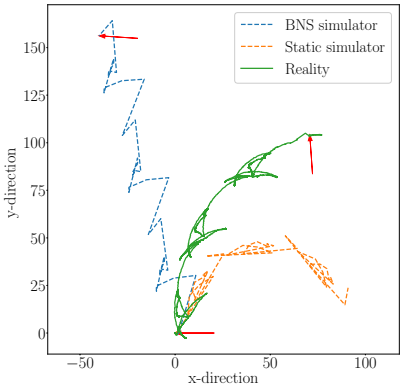


(a) First run

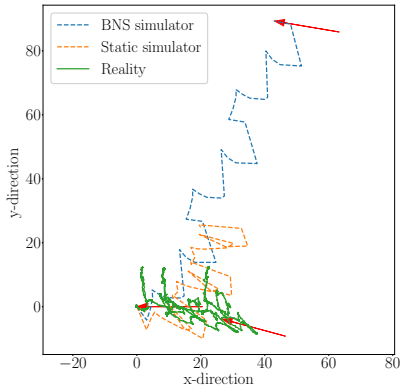
(b) Second Run



(c) Third run

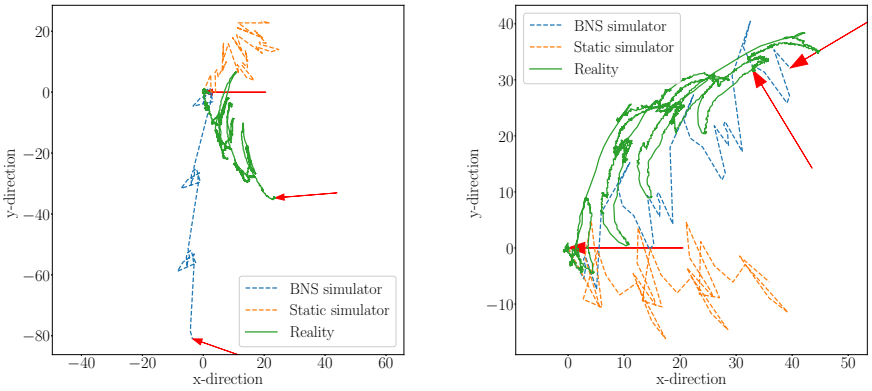


(d) Fourth run



(e) Fifth run

Figure 7.19: SESEU Real-world experiments



(a) First run

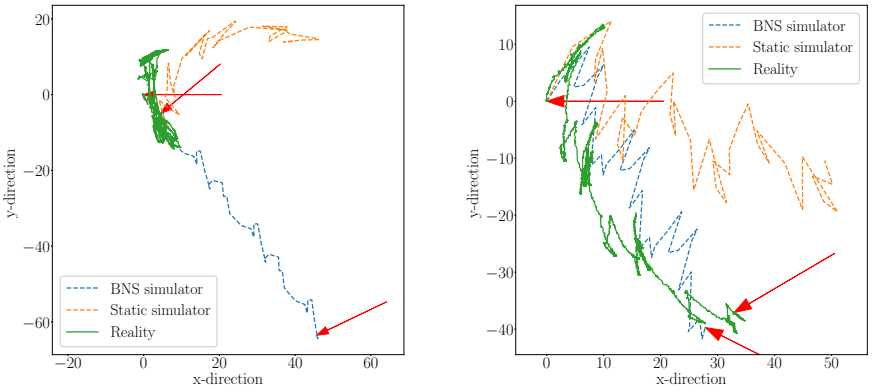
(b) Second Run

(c) Third run

(d) Fourth run

(e) Fifth run

Figure 7.20: SESNU Real-world experiments



(a) First run

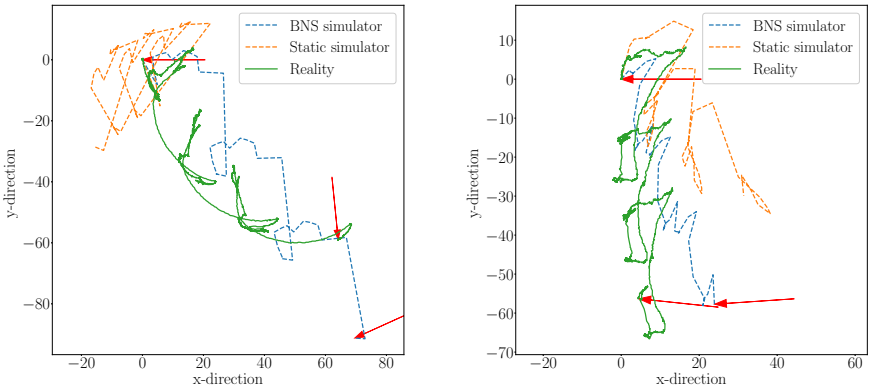
(b) Second Run

(c) Third run

(d) Fourth run

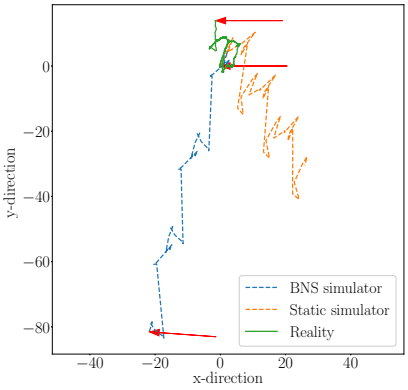
(e) Fifth run

Figure 7.21: SMSEU Real-world experiments

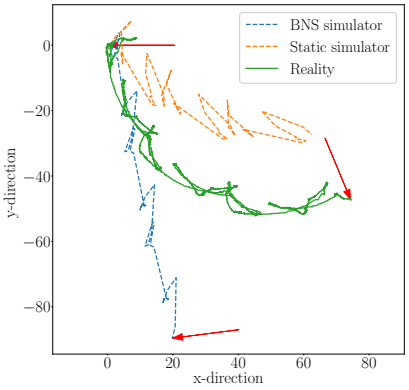


(a) First run

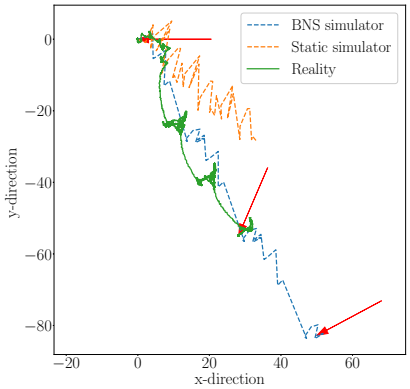
(b) Second Run



(c) Third run

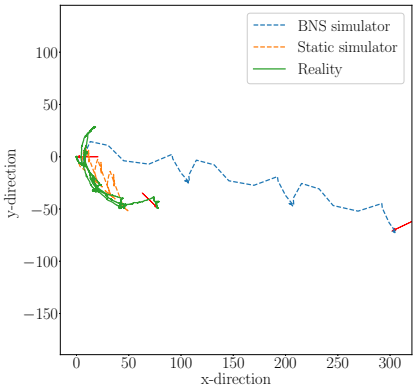


(d) Fourth run

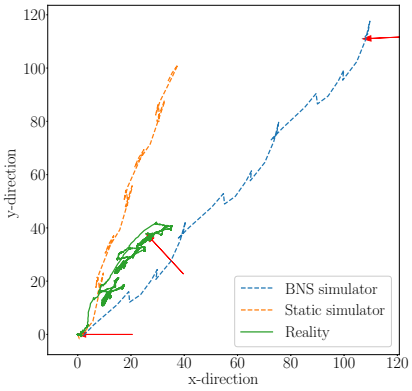


(e) Fifth run

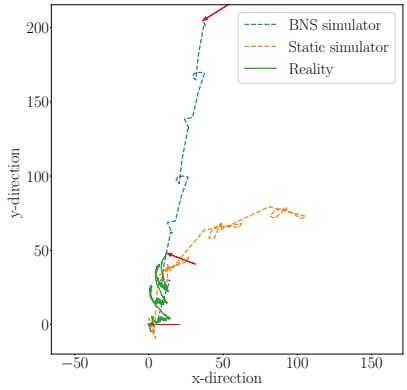
Figure 7.22: SMSNU Real-world experiments



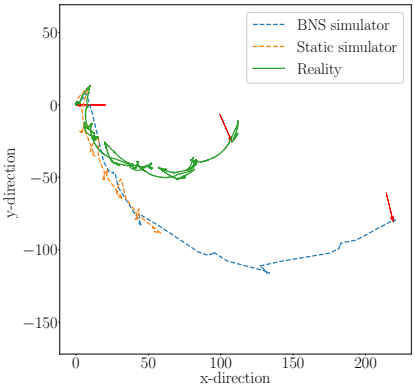
(a) First run



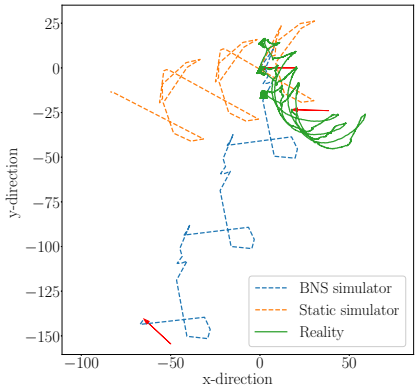
(b) Second Run



(c) Third run

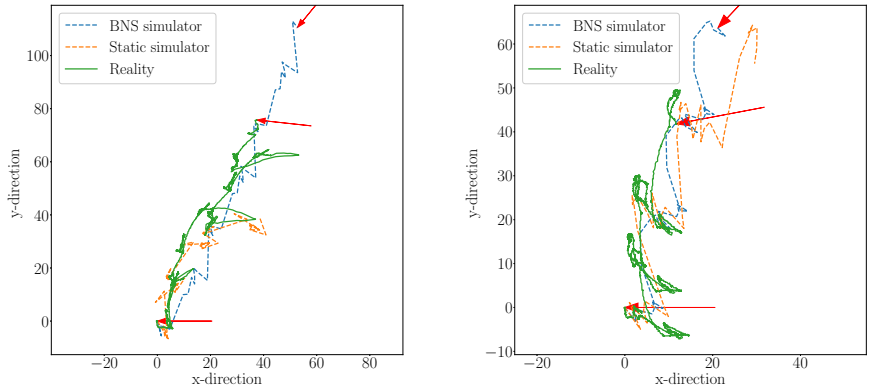


(d) Fourth run



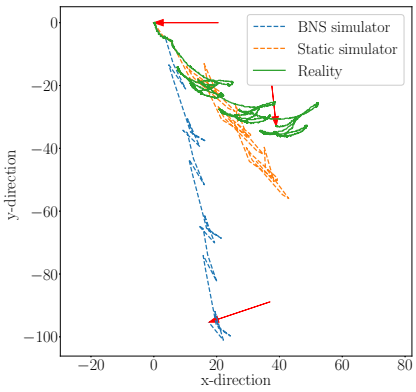
(e) Fifth run

Figure 7.23: SBSET Real-world experiments

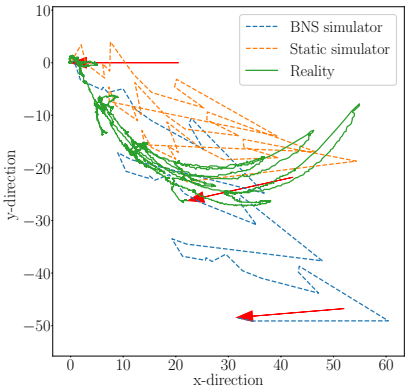


(a) First run

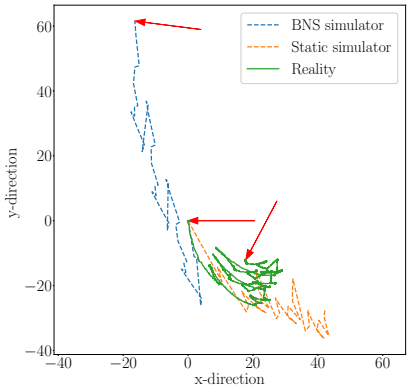
(b) Second Run



(c) Third run



(d) Fourth run



(e) Fifth run

Figure 7.24: SBSNT Real-world experiments

by more than 90 degrees. The Static simulator more accurately predicts the trajectory of the first solution compared to the BNS simulator. The BNS simulated distance travelled by the second and third solution controllers are greatly overestimated but the direction travelled is accurate. The fourth trial run has a distinct curved trajectory towards the right. The BNS simulated distance travelled by the fourth solution is overestimated but the general direction, heading and curvature are relatively accurate. The fifth trial run performs poorly and the distance travelled is greatly overestimated by the BNS simulator. For all SBSET solutions, the BNS simulator greatly overestimates the real-world distances travelled.

For the SBSNT adaptation, solution controllers are illustrated in Figure 7.24. The first and second trial runs have similar upward trajectories and the BNS simulated direction and distances travelled are relatively accurate. The third solution is more accurately simulated by the Static simulator compared to the BNS simulator. The BNS simulator appears to have failed to accurately simulate the heading of the robot which leads to an accumulation of errors in heading predictions. The fourth trial run is accurately simulated by the BNS simulator in terms of the heading and direction but the distance travelled is overestimated. Lastly, the fifth solution trajectory is poorly predicted by both the BNS and Static simulators with the real-world trajectory moving in the opposite direction predicted by the BNS simulator.

7.6.4 Summary

The validated adaptation using a noiseless **Ensemble** simulator configuration, simulator resetting and the **Most Uncertain** sampling strategy (SESEU) achieved the best overall performance amongst the validated adaptations. Adaptations consisting of **Ensemble Multi-output** simulator configurations with and without noise, use simulator resetting and the **Most Uncertain** sampling strategy (SMSEU, SMSNU) demonstrated particularly good transferability.

Most of the validated solutions achieved either excellent or acceptable performance levels. Only 7 out of the 30 validation solutions perform poorly with performance metrics less than 30 centimetres. The transference of simulated behaviours into reality is not nearly as accurate or reliable compared to when the BNS approach is applied to the

Hexapod robot. However, the Snake robot involves more complex behaviours and there is a greater accumulation of simulated errors over time compared to the Hexapod robot. Results demonstrate that the tested BNS adaptations can be applied to a real-world Snake robot.

7.7 SNS and BNS Comparisons

The Static Simulators for the SNS approach are trained from 4990 behavioural patterns. A single experimental run of the BNS approach only consists of 700 behavioural patterns (50 sampling controller evaluations). Including rest periods, the SNS approach had a data collection period of approximately 22 hours. Once the behavioural data collection phase is complete, training Static Simulators takes less than 15 minutes. The BNS approach requires about 4 hours per trial run and includes data collection, training and controller evolution.

The best performing adaptation for the SNS approach has a median performance of 49.7 centimetres and an IQR between 21.8 and 64.0 centimetres. The best performing validated BNS adaptation achieved a median performance of 78.5 centimetres and has an IQR between 30.7 and 83.3 centimetres. The best SNS adaptation performs less well compared to the best validated BNS adaptation. A statistical comparison between the SNS and BNS approaches is not possible due to the small sample sizes of the validated BNS adaptation results. None of the validated BNS solutions failed by turning upside down while many solution controllers for the SNS approach do indeed fail. This indicates that the BNS approach is more robust in terms of avoiding failures.

The transferability distributions of the tested SNS adaptations are compared to the validated BNS adaptations. The median transferability of all BNS validation solutions is 0.94 with an IQR between 0.54 and 2.44. The median transferability for each tested SNS adaptation is greater than 1.75 with all first quartiles being higher than 0.86 and the lowest third quartile achieved is 3.37. The validated BNS adaptations demonstrate better transferability compared to all the tested SNS adaptation. The BNS approach improves the likely transferability of solution controllers without sacrificing performance compared to the SNS approach while also minimising data collection.

7.8 Conclusions

This chapter demonstrates for the first time that the BNS approach can effectively develop Snake robot gaits using a controller design that does not rely on a significant amount of prior knowledge. Novel adaptation proposals are investigated and compared to each other in order to identify high performance adaptation settings. Promising BNS adaptations are validated using a real-world Snake robot.

Top performing adaptations mostly consist of either **Ensemble** or **Basic Multi-output** simulator configurations. Performing no resetting or periodically resetting only the simulator improves the likely performance outcomes of controller solutions. Adaptations that only periodically reset the simulator have significantly better transferability properties compared to the other tested resetting procedures.

According to the Simulated BNS Experiments, a noiseless simulator configuration improves the likely performance outcomes of solutions while including simulator noise improves the likely transferability of solution controllers. These performance and transferability properties related to simulator noise are confirmed during the Validation Experimental work. Performance and transferability outcomes with respect to simulator noise seem dependent on the robot morphology used and may be subject to change if the goal task is changed. The tested sampling strategies do not demonstrate significant changes in the likely performance or transferability of solution controllers. The BNS approach improves the likely transferability and robustness of solutions while achieving higher performance levels compared to the SNS approach.

Chapter 8

CONCLUSIONS AND FUTURE WORK

8.1 Introduction

In the previous four chapters, the SNS and BNS approaches are applied to Hexapod and Snake robot morphologies. Various novel adaptation settings are studied. The performance, transferability and behavioural properties of adaptation settings are explored during the experimental work. Top performing adaptations are identified and validated on real-world hardware. The aim of this chapter is to draw conclusions from the experimental work and discuss how research objective were achieved. Experimental outcomes of the SNS and BNS approaches applied to the two robots are compared and general conclusions are drawn.

An overall summary of the experimental results is given in Section 8.2. Research objectives are discussed in the context of the experimental findings (Section 8.3). Research contributions and limitations of this study are discussed in Sections 8.4 and 8.5, respectively. Recommendations and a discussion of future work is covered in Section 8.6. Lastly, an overall summary of the chapter is presented in Section 8.7.

8.2 Overview of Experimental Results

For each robot morphology, experimental observations are collected for the SNS and BNS approaches. For each approach, various combinations of proposed improvements are investigated and compared to each other. The scalability, generalisability, effectiveness and feasibility properties associated with the SNS and BNS approach, as defined in Section 1.3, are discussed in Table 8.1.

Conclusions drawn from the SNS experimental work are given in Table 8.2. Similarly, conclusions specific to the BNS approach are discussed in Table 8.3.

8.3 Outcomes of Research Objectives

Research objectives mentioned in Section 1.2 in relation to the experimental results achieved in this work are now discussed:

Identify relevant trends in the existing ER literature.

A literature review covering the integration of simulators into the ER process was conducted in Section 2.5. Simulator accuracy can improve the likely effectiveness of the evolutionary process. However, better accuracy usually comes at the expense of increased simulator complexity. Most ER approaches make use of physics-based simulation techniques that rely on a great amount of prior knowledge of the given robotic system. Simulation approaches that improve simulator accuracy with little prior human knowledge of the given robotic system are a significant contribution to the ER field.

Solution controllers evolved in simulation usually suffer from the *reality-gap* problem. Prior work indicates that simulator noise can play an role in reducing the **reality-gap** problem. Prior work also shows that bidirectional ER approaches can improve simulator accuracy and reduce *reality-gap* problems. Prominent bidirectional ER approaches are studied in Section 2.5.3. A high level comparison between relevant ER approaches is covered in Section 2.6.

Bidirectional ER approaches have certain advantages over static ER approaches. For bidirectional ER approaches, real-world behaviours observed during the ER process can be

	SNS Approach	BNS Approach
Scalability	Experimental results demonstrate that viable solutions are produced for the tested complex robot morphologies. Adaptations applied to the Hexapod robot achieve more consistently good results compared to the Snake robot.	Viable solutions are produced for the tested complex robot morphologies. Resulting solution outcomes are more consistently good for the Hexapod robot when compared to the Snake robot.
Generalisability	Viable controller solutions are successfully developed for both the Hexapod and Snake robot morphologies.	Viable solution controllers are successfully developed for both the Hexapod and Snake robot morphologies.
Effectiveness	High performance solutions are produced for the Hexapod and Snake robots. Producing high performance Snake controllers is more challenging compared to the Hexapod robot.	Validation experiments produce many highly effective solutions for the tested robots. Producing high performance solutions for the Snake robot is more challenging compared to the Hexapod robot.
Feasibility	The behavioural data collection phase consisted of 4942 and 4990 behavioural patterns for the Hexapod and Snake robots, respectively. The behavioural data collection phase took approximately 16 and 22 hours for the Hexapod and Snake robots, respectively. The data collection phase is only performed once. When data collection is complete, simulator training takes less than 15 minutes for any given simulator configuration. Developing a controller solution takes between 1.5 and 15 minutes for the Hexapod robot and between 2.0 and 7.5 hours for the Snake robot, depending on the adaptation used. Trained simulators can be reused for developing multiple independent solution controllers for arbitrary robot tasks.	The BNS approach collects 1100 and 700 behavioural patterns for the Hexapod and Snake robots, respectively. If sufficient rest periods are included, the total runtime is approximately 3.5 and 4.0 hours for the Hexapod and Snake robots, respectively. Assuming only a single solution controller is developed, the BNS approach is significantly less time-consuming to perform compared to the SNS approach.

Table 8.1: Summary of high level findings

	SNS Approach	
	Hexapod	**Snake**
Simulator Configuration	The best performing adaptation uses a noiseless **Ensemble Multi-output** simulator configuration. The combination of using ensembles, uncertainty penalties and multi-output SNNs resulted in improved performance outcomes. Adaptations consisting of either **Ensemble Multi-output** or **Basic Multi-output** simulator configurations have better transferability compared to other simulator configurations.	When considering the performance, failure rate and transferability of the tested adaptations, ideal adaptations consist of either the **Ensemble** or **Dropout** simulator configurations. Uncertainty penalisations in the fitness function likely helps the ER process to avoid simulated inaccuracies. The aggregation of simulated behaviours using ensembles or dropout improves the overall likely outcome of solutions.
Simulator Noise	All noiseless adaptations perform significantly better than adaptations that include simulator noise. Including simulator noise does not greatly affect the likely transferability of solution controllers.	Simulator noise did not significantly affect general performance outcomes of the tested adaptations. However, adaptations with simulator noise demonstrate greatly improved transferability properties compared to adaptations without simulator noise. The exploitation of simulator weaknesses during the ER process is greatly reduced through the inclusion of simulator noise.

Table 8.2: Summary of adaptation result for the SNS approach

	BNS Approach	
	Hexapod	**Snake**
Simulator Configuration	Adaptations demonstrating good performance and transferability properties use either **Ensemble** or **Ensemble Multi-output** simulator configurations.	High performance adaptations consist mainly of either **Ensemble** or **Basic Multi-output** simulator configurations. However, simulated observations for the **Basic Multi-output** simulator configuration are less reliable due to simulator exploitation problems. Adaptations consisting of **Ensemble Multi-output** or **Basic Multi-output** simulator configurations tend to have better transferability properties.
Simulator Noise	Most high performance adaptations do not include simulator noise. Noiseless simulator configurations also tend to have better transferability properties compared to adaptations that include simulator noise.	Adaptations without simulator noise are more likely to perform better than adaptations with simulator noise. Including simulator noise improves the likely transferability of solutions produced which is the opposite finding compared to the Hexapod robot.
Resetting	High performance adaptations at least periodically reset the simulator. Simulator resetting also helps improving the transferability properties of solutions.	High performance adaptations use either no resetting or simulator resetting. Adaptations using the simulator resetting procedure tend to produce more transferable solutions compared to other procedures.
Sampling Strategy	The tested sampling strategies do no affect the likely performance or transferability of controller solutions	The tested sampling strategies do not affect the likely performance or transferability of controller solutions

Table 8.3: Summary of adaptations results for the BNS approach

used to identify weaknesses in an existing simulator. These simulator weaknesses can either be corrected or avoided during controller evolution. Less behavioural data is collected because the simulator is only corrected or augmented based on evolved behaviours. This reduces the effort required to develop accurate simulators.

Existing research into the use of SNNs in ER has produced promising results. Prior work shows that the use of SNNs to simulate robot behaviours during the ER process reduces the need for specialised human knowledge compared to physics-based approaches. The simulator development phase can then be greatly simplified, shortened and automated through the use of SNNs.

Identify shortcomings of existing ER approaches.

Physics-based simulation approaches usually require significant prior knowledge of the robotic system relevant to the given problem. Some bidirectional ER approaches optimise physics-based model structures and/or parameter settings. The scalability and generalisability of current ER approaches that use physics-based simulators for different classes of robot morphologies is not well established. Automating and simplifying a significant portion of the simulator development process is not rigorously proposed or researched. Automating physics-based approaches without relying on a significant level of human knowledge is a difficult problem. In the existing body of work, few researchers are proposing and actively pursuing alternative simulation approaches that are simpler to construct and do not rely on significant levels of prior knowledge.

This research brings up the concept of demonstrated generality for a given ER approach. The viability and effectiveness of most ER approaches are not investigated across significantly different classes of robots. For example, certain approaches might be easy to implement for gait optimisation on limbed robots but infeasibly complex for crawling, limbless robot morphologies. When covering the existing literature, a complex Snake robot morphology is rarely used to demonstrate significant ER approaches.

Most prior studies investigate the SNS and BNS approaches on simple robot morphologies with few degrees of freedom. Studies that make use of a complex robot morphology for investigating the SNS and BNS approaches have relied on simplified controller/simulator designs with prior built-in knowledge. The scaling up of SNN-based approaches for high-

dimensional, low-level controller designs on complex robots has not been extensively studied. An exception is prior work investigating the SNS approach for a Hexapod robot morphology [Pretorius *et al.*, 2019]. No studies compare the SNS and BNS approaches to each other or identify differences in effectiveness between classes of complex robot morphologies. Prior work has been mainly focused on studying the **Basic** simulator configuration with limited exploration of alternative simulator configurations.

Demonstrate the scalability, generality, effectiveness and feasibility of the SNS and BNS approaches on complex robots using controllers that do not utilise prior knowledge.

The experimental results in this research demonstrate that the SNS and BNS approaches can scale for use on complex Hexapod and Snake robot morphologies. The SNS and BNS approaches are effectively applied to two significantly different complex robot morphologies. The SNS and BNS approaches can therefore be considered generalisable.

For the Hexapod and Snake robots, top performing SNS adaptations are effective in producing high performance solutions. Similarly, results demonstrate that the BNS approach can effectively produce high performance solutions when applied to the tested robot morphologies. However, the SNS and BNS approaches are more effective when applied to the Hexapod robot. The Snake robot morphology is a significantly more difficult platform to discover effective solutions.

For the Hexapod and Snake robots, Static SNNs are trained using similar amounts of behavioural training data. Data collection is spread out over 3 to 5 days for the SNS approach. Long periods of data collection cause overheating, damage the robot and require many manual interventions. However, prior studies have shown that the SNS approach can be easier to implement compared to physics-based simulators [Pretorius *et al.*, 2014, 2019]. Static SNNs are re-usable across trial runs. The BNS approach is considerably less time-consuming and more feasible to use than the SNS approach in producing a single solution controller quickly. A controller can be created with little prior knowledge or data using the BNS approach in about 4 hours for either of the tested robot morphologies.

The SNS approach is less time-consuming to use when producing many independent solutions or in scenarios involving many different goal tasks. The Static SNNs can be

re-used for simulated robot behaviours for any given task. Simulators developed using the BNS approach are unlikely to be effective for developing controllers for many difference problems. The effectiveness of re-using BNS simulators has not been studied.

Propose and investigate potential improvements to the SNS and BNS approaches.

The best SNS adaptation consisted of a noiseless **Ensemble Multi-output** simulator configuration for the Hexapod robot. Generally, SNS adaptations without simulator noise perform significantly better than adaptations with noise for the Hexapod robot. For the Hexapod robot, SNS adaptations consisting of simulator configurations using multi-output SNNs have better transferability compared to those using single-output SNNs.

For the Snake robot, the best SNS adaptations consisted of either a **Dropout** simulator configuration with noise or an **Ensemble** simulator configuration with/without noise. Simulator noise does not generally affect the likely performance of SNS adaptations for the Snake robot but noise does improve the likely transferability of solutions.

The best performing BNS adaptations for the Hexapod robot use either the noiseless **Ensemble** or noiseless **Ensemble Multi-output** simulator configuration with simulator resetting. Additionally, these best performing adaptations have relatively good transferability profiles. Non-intuitively, noiseless BNS adaptations tend to have better transferability properties compared to adaptations with noise for the Hexapod robot.

For the Snake robot, top performing BNS adaptations consist of either a noiseless **Ensemble** or noiseless **Basic Multi-output** simulator configuration. The best BNS adaptations for the Snake robot all use either a no resetting or simulator resetting procedure. Simulator noise and simulator resetting are both shown to improve the likely transferability of BNS adaptation solutions for the Snake robot. Larger more complicated simulator architectures are more likely to consist of many inaccuracies or weaknesses in simulated behaviours. Periodically resetting the simulator likely prevents the ER process from exploiting simulator weaknesses or inaccuracies. For more complex simulator architectures, noise seems to prevent the ER process from exploiting simulator weaknesses.

High performance adaptations tend to produce uncertainty information which penalises robot behaviours that may be inaccurately simulated. Contributing factors such as en-

semble techniques additionally contribute towards improving performance outcomes due to the aggregation of simulator inaccuracies.

High performance BNS adaptations common to both robots use a noiseless **Ensemble** simulator configuration with periodic simulator resetting. The tested sampling strategies do not affect on the likely performance or transferability of solution controllers. Simulator resetting is found to improve the transferability of solution controllers for both the Hexapod and Snake robot morphologies. BNS adaptations without simulator noise improve the likely transferability of solutions for the Hexapod robot but simulators with noise improve the likely transferability of solutions for the Snake robot. This is likely due to the Snake robot simulator being more susceptible to having its weaknesses exploited during the ER process. It is likely that the size of the Snake robot SNNs are too large and can be simplified for the BNS approach.

Determine the relative advantages and disadvantages of the SNS and BNS approaches.

Data collection, simulator training and controller evolution take place sequentially for the SNS approach. After SNNs are trained, the SNS approach does not require the use of any real-world hardware until the final solution is tested. This allows multiple SNS trials to be run over many computers at the same time in order to produce many independent solutions. However, the BNS approach requires constant access to a real-world robot. If a large number of independent solution controllers needs to be found, the SNS approach is a more appropriate choice compared to the BNS approach.

Simulators developed using the BNS approach are specialised towards simulating particular behaviours for a single solution. Simulators developed through the SNS approach are capable of generalising for use across multiple problems and/or behavioural search spaces. The SNS approach is suitable for ER problems where multiple goal tasks are specified or many independent solutions are produced.

The BNS approach can be used to produce a single solution controller relatively quickly without a lengthy data collection phase. In scenarios where data collection needs to be minimised and only a single effective solution is required, the BNS approach is the best choice. If the robot morphology or environment are likely to change frequently, the BNS

approach is also advantageous.

For the SNS approach applied to a Hexapod robot, it takes approximately between 16.3 and 16.5 hours to produce the first solution controller, depending on the adaptation used. Similarly, it takes approximately between 24.3 and 29.8 hours to produce the first solution controller for the Snake robot, depending on the adaptation used. For the BNS approach, the first solution controller is produced after 3.5 and 4.0 hours for the Hexapod and Snake robots, respectively.

The following discussion assumes that only a single instance of the SNS approach is run at any given time and SNS trials are not run simultaneously across many computers. In order to produce 30 solution controllers, the SNS approach would take between 17.0 and 23.8 hours for the Hexapod robot, depending on the adaptation used. For the SNS approach applied to a Snake robot and depending on the adaptation used, it can take between 82.3 and 247.3 hours to produce 30 solution controllers. However, if more computing resources are available and SNS trials can be performed simultaneously, producing 30 solution controllers in the same amount of time taken to produce the first solution.

For the BNS approach to produce 30 independent solution controllers, it would take approximately 105 and 120 hours for the Hexapod and Snake robots, respectively. The SNS approach is a better choice for producing 30 solution controllers for both the Hexapod and Snake robots. Performing over 100 hours of evaluations on either robot would definitely cause significant damage to motors, would be financially costly and require significant manual interventions. This also assumes that data sharing between BNS trial runs is not considered.

8.4 Contributions and Recommendations

This thesis confirms prior research that the SNS approach is scalable and generalisable. The BNS approach is demonstrated for the first time to be viable on a Hexapod robot. The BNS approach can then be considered generalisable due to the approach being demonstrated on two complex robots. For the SNS and BNS approaches, the use of non-simplified controller designs are investigated for the first time on a Snake robot morphology.

The study presents a first time investigation of the inclusion or exclusion of simulator

noise for the BNS approach. Results demonstrate that not including simulator noise for the SNS and BNS approaches can increase the likely performance outcomes of solutions.

Experimental outcomes found that the relationship between transferability and simulator noise depends on the robot morphology used. Adding simulator noise improves transferability for the Snake robot. However, excluding simulator noise improves the likely transferability of solution controllers for the Hexapod robot. This may be dependent on the size of the SNN architecture used.

For the SNS approach, the chosen simulator configuration can significantly influence the behavioural strategies seen in solution trajectories. Adaptations using certain simulator configurations can produce diverse sets of solution behaviours while the use of other simulator configurations can produce solutions with relatively similar behaviours.

Prior BNS research used predetermined parameter setting values for behavioural data standardisation and simulator noise generation. This thesis proposes and validates a method for dynamically calculated all these parameter setting values during the BNS approach.

Novel adaptation settings are proposed and investigated for the SNS and BNS approaches. **Dropout**, **Ensemble** and **Ensemble Multi-output** simulator configurations are novel proposals. For the SNS and BNS approaches, generating uncertainty information using dropout or ensembles are novel proposals. Multi-output and single-output SNNs are for the first time compared for complex robot morphologies. Contrary to prior research, experimental results found that single-output SNNs are not always a better choice over multi-output SNNs. Controller and simulator resetting procedures are also novel proposals. The **Most Uncertain** sampling strategy is a novel proposal.

If given an new robot morphology and problem, the recommended approach and adaptation to use would depend on certain goals. If the goal is to discover a viable solution quickly, it is recommended to use a noiseless **Ensemble** simulator configuration and to use the BNS approach with simulator resetting. However, if the problem is such that a good transferability is important, an **Ensemble Multi-output** simulator configuration might produce better solution outcomes. Alternatively, if the goal is to produce a large number of diverse solution controllers, the SNS approach is more appropriate but using a **Basic** simulator configuration.

8.5 Limitations

This research has certain limitations to consider when discussing the results. Restrictions placed on the scope of the study contribute to limitations.

The SNS and BNS approaches are found to be viable and effective for the tested robot morphologies and problems. Results indicate that approaches presented in this work are likely to be effective for other robot morphologies. However, this work does not demonstrate that the SNS and BNS approaches are guaranteed to perform well on every possible robot morphology or given problem. Further research is required in order to fully confirm the extent of the effectiveness and generalisability of the SNS and BNS approaches.

The robotic problem explored in this research is limited to distance maximising open loop walking or crawling gait optimisation. This research indicates that not including simulator noise improves the likely performance outcomes of solutions. This observation might be problem specific and simulator noise could be more important for goal tasks not investigated in this research. The chosen goal task in this work is likely more forgiving towards adaptations not including simulator noise.

Robot control policies do not use real-time sensor or position feedback during evaluations. SNNs presented in this research are limited to simulating changes in position, heading and orientations but do not simulate interactions with other robots or objects.

SNNs are trained from observations collected from real-world controller evaluations. The tracking system used to collect behavioural data does not have perfect accuracy and tracked behaviours may contain errors. Due to the high dimensional behavioural search space of the controller design chosen for this work, a fully representative training sample is practically impossible to collect. Trained SNNs will inevitably contain weaknesses.

For the BNS approach, the validated adaptations have small test sample sizes. These sample sizes are too small to achieve statistically significant results for comparisons. The validation experiments are mainly used to confirm the viability of the BNS approach on real-world hardware.

8.6 Future Research

Alternative methods for behavioural data tracking could be proposed and studied. This could be particularly beneficial for Snake-like robot morphologies. Whole body tracking could provide better accuracy and flexibility in evolving particular behaviours or penalising undesired ones.

Ensemble configurations are shown to be particularly effective, however, only a limited set of ensemble configurations are investigated. Future work could investigate other types of ensemble configurations, with ensembles of different types of SNNs. The simulator configuration and even SNN architectures could be evolved during training. Ensembles consisting of multiple types of regression modelling techniques seen in Machine Learning could be investigated.

The BNS approach could be extended, such that behavioural data could be reused between BNS trial runs, or even changing the robotic task between trial runs. Investigations into the use of SNNs for robot damage recover or changing environments is a promising research topic for future work. The BNS approach in particular could be adapted for changing robotic systems.

The BNS approaches explored in this research use SNN architectures identified as part of the SNS experimental work. Ideally, SNN architectures should be dynamically calculated during the BNS approach. Simpler SNN architectures would be utilised during the early stages of the BNS approach with slow increments in the size and complexity as more behavioural data is collected.

8.7 Summary

Simulating and successfully evolving target robot behaviours on different classes of complex robots is a non-trivial task. Controller and simulator designs in the ER field are often simplified using specialised knowledge. Simulators and controllers in this work are designed with little prior knowledge specific to the robotic systems.

Few ER approaches in the literature have been demonstrated on different classes of robot morphologies. This study investigates the SNS and BNS approaches on two com-

plex robot morphologies. The Hexapod and Snake robots generate different classes of behaviours. Experimental results suggest that the SNS and BNS approaches could be successfully applied to other classes of robots in future work.

Proposed improvements to the SNS and BNS approaches are investigated and compared. Results helped to identify high performance adaptation settings common to both robot morphologies. This research leads to recommendations for ideal adaptation settings that could be successfully applied to untested robot morphologies or problems.

Most ER researchers have spent a great deal of energy learning existing physics-based frameworks and techniques. This prior knowledge is often taken for granted when assessing ER approaches. For any particular simulation approach, the amount of human gained specialised knowledge required to build an effective simulator should be considered. ER approaches that reduce the required human knowledge needed to solve particular problems are a valuable contribution to the ER field. The SNS and BNS approaches are promising ER approaches for reducing simulation complexity and human interventions.

Bibliography

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, and M. Devin. TensorFlow: Large-scale machine learning on heterogeneous systems. `http://tensorflow.org/`, 2015.

C. Au and P. Jin. Investigation of serpentine gait of a snake robot with a wireless camera. In *2016 12th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA)*, pages 1–6. IEEE, 2016.

J. Auerbach and J. Bongard. Environmental influence on the evolution of morphological complexity in machines. *PLoS computational biology*, 10(1):e1003399, 2014.

D. Beasley, R. Martin, and D. Bull. An overview of Genetic Algorithms: Part 1. Fundamentals. *University Computing*, 15(2):58–69, 1993.

R. Bellman. Adaptive control processes: A guided tour, 1961.

D. Belter and P. Skrzypczyński. A biologically inspired approach to feasible gait learning for a hexapod robot. *International Journal of Applied Mathematics and Computer Science*, 20(1):69–84, 2010.

Y. Bengio. Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2(1):1–127, 2009.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

J. Bongard. Evolutionary robotics. *Communications of the ACM*, 56(8):74–83, 2013.

J. Bongard and G. Hornby. Combining fitness-based search and user modeling in evolutionary robotics. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 159–166. ACM, 2013.

J. Bongard and H. Lipson. Automated damage diagnosis and recovery for remote robotics. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 4, pages 3545–3550. IEEE, 2004a.

J. Bongard and H. Lipson. Once more unto the breach: Co-evolving a robot and its simulator. In *Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems (ALIFE9)*, pages 57–62, 2004b.

J. Bongard and H. Lipson. Nonlinear system identification using coevolution of models and tests. *IEEE Transactions on Evolutionary Computation*, 9(4):361–384, 2005.

J. Bongard, V. Zykov, and H. Lipson. Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121, December 2006a.

J. Bongard, V. Zykov, and H. Lipson. Automated synthesis of body schema using multiple sensor modalities. In *Proc. of the Int. Conf. on the Simulation and Synthesis of Living Systems (ALIFEX)*. Citeseer, 2006b.

Encyclopedia Britannica. 2012.

R. Brooks. Artificial life in real robots. *Artificial Intelligence*, 48:3–10, 1992.

Erwin C. Bullet physics engine. `https://pybullet.org`, 2019.

K. Chatzilygeroudis, R. Rama, R. Kaushik, D. Goepp, V. Vassiliades, and J. Mouret. Black-box data-efficient policy search for robotics. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 51–58. IEEE, 2017.

N. Cheney, J. Clune, and H. Lipson. Evolved electrophysiological soft robots. In *Artificial Life Conference Proceedings 14*, pages 222–229. MIT Press, 2014.

F. Chollet. Keras. `https://keras.io`, 2015.

A. Cully, J. Clune, D. Tarapore, and J. Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

R. De Nardi. *Automatic Design of Controllers for Miniature Vehicles through Automatic Modelling*. PhD thesis, University of Essex, 2010.

R. De Nardi and O. Holland. Coevolutionary modelling of a miniature rotorcraft. In *10th International Conference on Intelligent Autonomous Systems (IAS10)*, pages 364–373, 2008.

L. Deng and D. Yu. Deep learning: methods and applications. *Foundations and Trends in Signal Processing*, 7(3–4):197–387, 2014.

P. Domingos. A few useful things to know about machine learning. *Commun. acm*, 55 (10):78–87, 2012.

S. Doncieux and J. Mouret. Beyond black-box optimization: a review of selective pressures for evolutionary robotics. *Evolutionary Intelligence*, 7(2):71–93, 2014.

S. Doncieux, N. Bredeche, J. Mouret, and A. Eiben. Evolutionary robotics: what, why, and where to. *Frontiers in Robotics and AI*, 2:4, 2015.

K. Dowling. *Limbless locomotion: Learning to crawl with a snake robot*. PhD thesis, NASA, 1996.

S. Easterbrook, J. Singer, M. Storey, and D. Damian. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*, pages 285–311. Springer, 2008.

A. Engelbrecht. *Computational intelligence: An introduction*. John Wiley & Sons, 2007.

D. Floreano and F. Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. *From animals to animats*, pages 421–430, 1994.

D. Floreano, P. Husbands, and S. Nolfi. Evolutionary robotics. *Springer handbook of robotics*, pages 1423–1451, 2008.

S. Fortmann-Roe. Understanding the bias-variance tradeoff. 2012.

Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International conference on machine learning*, pages 1050–1059, 2016.

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

C. Gong, M. Travers, H. Astley, L. Li, J. Mendelson, D. Goldman, and H. Choset. Kinematic gait synthesis for snake robots. *The International Journal of Robotics Research*, 35(1-3):100–113, 2016.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

D. Gray. *Doing research in the real world*. Sage, 2009.

J. Grefenstette and C. Ramsey. An approach to anytime learning. In *Machine Learning Proceedings 1992*, pages 189–195. Elsevier, 1992.

J. Hallam and A. Ijspeert. Using evolutionary methods to parameterize neural models: a study of the lamprey central pattern generator. In *Biologically inspired robot behavior engineering*, pages 119–142. Springer, 2003.

I. Harvey, P. Husbands, D. Cliff, A. Thompson, and N. Jakobi. Evolutionary robotics: the sussex approach. *Robotics and autonomous systems*, 20(2-4):205–224, 1997.

I. Harvey, E. Paolo, R. Wood, M. Quinn, and E. Tuci. Evolutionary robotics: A new scientific tool for studying cognition. *Artificial life*, 11(1-2):79–98, 2005.

S. Hasanzadeh and A. Akbarzadeh. Development of a new spinning gait for a planar snake robot using central pattern generators. *Intelligent Service Robotics*, 6(2):109–120, 2013.

S. Hasanzadeh and A. Tootoonchi. Ground adaptive and optimized locomotion of snake robot moving with a novel gait. *Autonomous Robots*, 28(4):457–470, 2010.

Y. Hong and B. Lee. Evolutionary optimization for optimal hopping of humanoid robots. *IEEE Transactions on Industrial Electronics*, 64(2):1279–1283, 2017.

G. Hornby, S. Takamura, T. Yamamoto, and M. Fujita. Autonomous evolution of dynamic gaits with two quadruped robots. *IEEE Transactions on Robotics*, 21(3):402–410, 2005.

D. Hu, J. Nirody, T. Scott, and M. Shelley. The mechanics of slithering locomotion. *Proceedings of the National Academy of Sciences*, 106(25):10081–10085, 2009.

G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer, 2016.

L. Iocchi, F. Libera, and E. Menegatti. Learning humanoid soccer actions interleaving simulated and real data. In *Second Workshop on Humanoid Soccer Robots*, 2007.

N. Jakobi. Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive behavior*, 6(2):325–368, 1997.

N. Jakobi. *Minimal simulations for Evolutionary Robotics*. PhD thesis, University of Sussex, 1998a.

N. Jakobi. Running across the reality gap: Octopod locomotion evolved in a minimal simulation. In *European Workshop on Evolutionary Robotics*, pages 39–58. Springer, 1998b.

N. Jakobi, P. Husbands, and I. Harvey. Noise and the reality gap: The use of simulation in Evolutionary Robotics. In *Advances in artificial life*, pages 704–720. Springer, 1995.

A. Kamimura, H. Kurokawa, E. Yoshida, S. Murata, K. Tomita, and S. Kokaji. Automatic locomotion design and experiments for a modular robotic system. *Mechatronics, IEEE/ASME Transactions on*, 10(3):314–325, 2005.

S. Kamio and H. Iba. Evolutionary construction of a simulator for real robots. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, pages 2202–2209. IEEE, 2004.

S. Kamio and H. Iba. Adaptation technique for integrating genetic programming and reinforcement learning for real robots. *Evolutionary Computation, IEEE Transactions*, 9(3):318–333, 2005.

R. Kaushik, K. Chatzilygeroudis, and J. Mouret. Multi-objective model-based policy search for data-efficient learning with sparse rewards. *arXiv preprint arXiv:1806.09351*, 2018.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

G. Klaus, K. Glette, and J. Tørresen. A comparison of sampling strategies for parameter estimation of a robot simulator. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 173–184. Springer, 2012.

S. Koos, A. Cully, and J. Mouret. Fast damage recovery in robotics with the T-resilience algorithm. *The International Journal of Robotics Research*, 32(14):1700–1723, 2013a.

S. Koos, J. Mouret, and S. Doncieux. The transferability approach: Crossing the reality gap in evolutionary robotics. *Evolutionary Computation, IEEE Transactions on*, 17(1): 122–145, 2013b.

J. Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413, 2017.

J. Lee, M. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. Srinivasa, M. Stilman, and C. Liu. DART: Dynamic animation and robotics toolkit. *The Journal of Open Source Software*, 3(22):500, 2018.

T. Lee, U. Nehmzow, and R. Hubbold. Mobile Robot Simulation by Means of Acquired Neural Network Models. In *ESM*, pages 465–469, 1998.

T. Lee, U. Nehmzow, and R. Hubbold. Computer simulation of learning experiments with autonomous mobile robots. *Proceedings of TIMR*, 99, 1999.

J. Liang and C. Xue. Self identification and control of four-leg robot based on biological evolutionary mechanisms. In *2010 5th IEEE Conference on Industrial Electronics and Applications*, pages 958–961. IEEE, 2010.

H. Lipson and J. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974–978, 2000.

H. Lipson, J. Bongard, V. Zykov, and E. Malone. Evolutionary robotics for legged machines: From simulation to physical reality. In *IAS*, pages 11–18, 2006.

D. Lizotte, T. Wang, M. Bowling, and D. Schuurmans. Automatic Gait Optimization with Gaussian Process Regression. In *IJCAI*, volume 7, pages 944–949, 2007.

H. Lund. Co-evolving control and morphology with LEGO Robots. In *Morpho-functional Machines: The New Species*, pages 59–79. Springer, 2003.

H. Lund and O. Miglino. From simulated to real robots. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 362–365. IEEE, 1996.

L. Mainini and K. Willcox. Surrogate modeling approach to support real-time structural assessment and decision making. *AIAA Journal*, 53(6):1612–1626, 2015.

A. Maren, C. Harston, and R. Pap. *Handbook of neural computing applications*. Academic Press, 2014.

M. Matarić and D. Cliff. Challenges in evolving controllers for physical robots. *Robotics and autonomous systems*, 19(1):67–83, 1996.

K. Melo, M. Hernandez, and D. Gonzalez. Parameterized space conditions for the definition of locomotion modes in modular snake robots. In *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*, pages 2032–2038. IEEE, 2012.

O. Miglino, K. Nafasi, and C. Taylor. Selection for wandering behavior in a small robot. *Artificial Life*, 2(1):101–116, 1994.

O. Miglino, H. Lund, and S. Nolfi. Evolving mobile robots in simulated and real environments. *Artificial life*, 2(4):417–434, 1995.

R. Moeckel, Y. Perov, A. Nguyen, M. Vespignani, S. Bonardi, S. Pouya, A. Sproewitz, J. van den Kieboom, F. Wilhelm, and A. Ijspeert. Gait optimization for roombots modular robots - matching simulation and reality. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 3265–3272. Ieee, 2013.

J. Montanier and N. Bredeche. Surviving the tragedy of commons: emergence of altruism in a population of evolving autonomous agents. 2011.

J. Mouret and K. Chatzilygeroudis. 20 years of reality gap: a few thoughts about simulators in evolutionary robotics. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1121–1124. ACM, 2017.

J. Mouret, S. Koos, and S. Doncieux. Crossing the reality gap: a short introduction to the transferability approach. In *In Proceedings of the ALIFE workshop "evolution in physical systems"*, 2012.

V. Nair and G. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

S. Nakamura and S. Hashimoto. Hybrid learning strategy to solve pendulum swing-up problem for real hardware. In *2007 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1972–1977. IEEE, 2007.

S. Nakamura, R. Saegusa, and S. Hashimoto. A hybrid learning strategy for real hardware of swing-up pendulum. *Journal of Advanced Computational Intelligence & Intelligent Informatics (JACIII)*, 11(8), 2007.

U. Nehmzow, D. Kerr, and S. Billings. Accurate robot simulation. Technical Report ACSE Research Report no. 992, University of Sheffield, 2009.

D. Nguyen-Tuong and J. Peters. Model learning for robot control: a survey. *Cognitive processing*, 12(4):319–340, 2011.

S. Nolfi, J. Bongard, P. Husbands, and D. Floreano. Evolutionary robotics. In *Springer Handbook of Robotics*, pages 2035–2068. Springer, 2016.

R. Olson, A. Hintze, F. Dyer, D. Knoester, and C. Adami. Predator confusion is sufficient to evolve swarming behaviour. *Journal of The Royal Society Interface*, 10(85):20130305, 2013.

G. Parker. Co-evolving model parameters for anytime learning in evolutionary robotics. *Robotics and Autonomous Systems*, 33(1):13–30, 2000.

G. Parker. Punctuated anytime learning for hexapod gait generation. In *IROS*, pages 2664–2671, 2002.

J. Pollack, H. Lipson, S. Ficici, P. Funes, G. Hornby, and R. Watson. Evolutionary techniques in physical robotics. In *Evolvable Systems: from biology to hardware*, pages 175–186. Springer, 2000.

D. Pratihar. Evolutionary robotics - A review. *Sadhana*, 28(6):999–1009, 2003.

C. Pretorius. *Artificial Neural Networks as simulators for behavioural evolution in Evolutionary Robotics*. MasterâĂŹs thesis, Nelson Mandela Metropolitan University, 2010.

C. Pretorius, M. du Plessis, and C. Cilliers. Towards an Artificial Neural Network-based simulator for behavioural evolution in Evolutionary Robotics. In *Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*, pages 170–178. ACM, 2009.

C. Pretorius, M. du Plessis, and C. Cilliers. Simulating robots without conventional physics: A neural network approach. *Journal of Intelligent & Robotic Systems*, 71(3-4): 319–348, 2013.

C. Pretorius, M. du Plessis, and J. Gonsalves. A comparison of Neural Networks and physics models as motion simulators for simple robotic evolution. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 2793–2800. IEEE, 2014.

C. Pretorius, M. du Plessis, and J. Gonsalves. Neuroevolution of inverted pendulum control: a comparative study of simulation techniques. *Journal of Intelligent & Robotic Systems*, 86(3-4):419–445, 2017.

J. Pretorius, M. du Plessis, and J. Gonsalves. Evolutionary robotics applied to hexapod locomotion: a comparative study of simulation techniques. *Journal of Intelligent & Robotic Systems*, pages 1–23, 2019.

M. Quinn, L. Smith, G. Mayley, and P. Husbands. Evolving controllers for a homogeneous system of physical robots: Structured cooperation with minimal sensors. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 361(1811):2321–2343, 2003.

S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

S. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

E. Şahin, S. Girgin, L. Bayindir, and A. Turgut. Swarm robotics. In *Swarm intelligence*, pages 87–100. Springer, 2008.

M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

H. Schwefel. *Evolution and optimum seeking: the sixth generation*. John Wiley & Sons, Inc., 1993.

O. Shmakov. Snakelike robots locomotions control. *Mechatronics–Foundations and Applications*, 2006.

F. Silva, M. Duarte, S. Oliveira, L. Correia, and A. Christensen. The case for engineering the evolution of robot controllers. In *Artificial Life Conference Proceedings 14*, pages 703–710. MIT Press, 2014.

F. Silva, M. Duarte, L. Correia, S. Oliveira, and A. Christensen. Open issues in evolutionary robotics. *Evolutionary computation*, 24(2):205–236, 2016.

R. Smith. Open dynamics engine. 2005.

D. Sofge, M. Potter, M. Bugajska, and A. Schultz. Challenges and opportunities of evolutionary robotics. In *Proceedings of the Second International Conference on Computational Intelligence*. Robotics and Autonomous Systems, 2003.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

J. Togelius, R. De Nardi, H. Marques, R. Newcombe, S. Lucas, and O. Holland. Nonlinear dynamics modelling for controller evolution. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 324–333. ACM, 2007.

L. Wang. A hybrid genetic algorithm - neural network strategy for simulation optimization. *Applied Mathematics and Computation*, 170(2):1329–1343, 11 2005.

S. Wischmann, D. Floreano, and L. Keller. Historical contingency affects signaling strategies and competitive abilities in evolving populations of simulated robots. *Proceedings of the National Academy of Sciences*, 109(3):864–868, 2012.

S. Wittmeier, M. Jäntsch, K. Dalamagkidis, and A. Knoll. Physics-based modeling of an anthropomimetic robot. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4148–4153. IEEE, 2011.

G. Woodford. *Concurrent Controller and Simulator Neural Network Development in the Evolutionary Robotics Process*. Masters thesis, Nelson Mandela University, 2016.

G. Woodford and M. du Plessis. Robotic snake simulation using ensembles of artificial neural networks in evolutionary robotics. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 173–180. ACM, 2018.

G. Woodford and M. du Plessis. Complex morphology neural network simulation in evolutionary robotics. *Robotica*, pages 1–17, 2019.

G. Woodford, M. du Plessis, and C. Pretorius. Evolving snake robot controllers using artificial neural networks as an alternative to a physics-based simulator. In *Computational Intelligence, 2015 IEEE Symposium Series on*, pages 267–274. IEEE, 2015.

G. Woodford, C. Pretorius, and M. du Plessis. Concurrent controller and simulator neural network development for a differentially-steered robot in evolutionary robotics. *Robotics and Autonomous Systems*, 76:80–92, 2016.

G. Woodford, M. du Plessis, and C. Pretorius. Concurrent controller and simulator neural network development for a snake-like robot in evolutionary robotics. *Robotics and Autonomous Systems*, 88:37–50, 2017.

J. Zagal and J. Ruiz-del Solar. Combining simulation and reality in evolutionary robotics. *Journal of Intelligent and Robotic Systems*, 50(1):19–39, March 2007. ISSN 0921-0296.

J. Zagal, J. Delpiano, and J. Ruiz-del Solar. Self-modeling in humanoid soccer robots. *Robotics and Autonomous Systems*, 57(8):819–827, 2009.

A. Zavoianu, E. Lughofer, G. Bramerdorfer, W. Amrhein, and E. Klement. An effective ensemble-based method for creating on-the-fly surrogate fitness functions for multi-objective evolutionary algorithms. In *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 235–242. IEEE, 2013.

V. Zykov, A. Chan, and H. Lipson. Molecubes: An open-source modular robotics kit. In *IROS-2007 Self-Reconfigurable Robotics Workshop*, pages 3–6, 2007.

Appendix A

Research Output by the Author

Conference proceedings:

- Woodford, G., du Plessis, M. and Pretorius, C. Evolving snake robot controllers using Artificial Neural Networks as an alternative to a physics-based simulator, *IEEE Symposium Series on Computational Intelligence*, 267-274, 2015.
- Woodford, G., and du Plessis, M. Robotic snake simulation using ensembles of artificial neural networks in Evolutionary Robotics, *In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, ACM, 173-180, 2018.†
- Leonard, B., du Plessis, M. and Woodford, G. Damage recovery for evolved autonomous agents using Bootstrapped Neuro-Simulation, *International Conference on Computer Science and Information Technology (ICCSIT)*, 2018.†

Peer-reviewed journal publications:

- Woodford, G., Pretorius, C. and du Plessis, M. Concurrent controller and Simulator Neural Network development for a differentially-steered robot in Evolutionary Robotics, *Robotics and Autonomous Systems*, **76**: 80-92, 2016.
- Woodford, G., du Plessis, M. and Pretorius, C. Concurrent controller and Simulator Neural Network development for a snake-like robot in Evolutionary Robotics, *Robotics and Autonomous Systems* **88**: 37-50, 2017.

†produced as a result of research for this thesis

- Woodford, G. and du Plessis, M. Complex morphology neural network simulation in Evolutionary Robotics, *Robotica*, Cambridge University Press, pages 1-17, 2019.†

- Woodford, G. and du Plessis, M. Bootstrapped Neuro-Simulation for complex robots, *Robotics and Autonomous Systems*, 2019. Under review.†

- Leonard, B., du Plessis, M. and Woodford, G. Bootstrapped Neuro-Simulation as a method of Concurrent Neuro-Evolution and Damage Recovery, *Robotics and Autonomous Systems*, 2019. Accepted.†

Appendix B

Experimental Results

Benchmarking results for identifying the ideal Snake robot SNN architectures and the Simulated Experimental trial results for the Hexapod and Snake robots can be found here: https://github.com/gwwoodford/PhD

For the tested SNS adaptations, the performance measurements of the 30 trial runs for the Hexapod and Snake robots are given in Tables B.1 and B.3, respectively. Similarly, the transferability measurements for the Hexapod and Snake robots are given in Tables B.2 and B.4, respectively.

The performance and transferability values for each trial run of the Hexapod Validation BNS Experiments are given in Tables B.5 and B.6, respectively. The performance and transferability values for each trial run of the Snake Validation BNS Experiments are given in Tables B.7 and B.8.

The adaptation performance and transferability statistics for the Hexapod Simulated BNS Experiments are given in Tables B.9 and B.10, respectively.

The adaptation performance and transferability statistics for the Snake Simulated BNS Experiments are given in Tables B.11 and B.12, respectively.

HBE	HBN	HDE	HDN	HEE	HEN	HME	HMN	HSE	HSN
46.7	41.8	52.5	39.9	44.0	47.9	66.5	51.3	50.3	49.7
69.9	41.2	45.8	57.3	62.7	37.1	54.3	36.2	46.7	53.8
51.6	55.4	44.3	33.3	30.2	47.2	54.7	51.7	54.9	50.1
57.7	42.5	69.9	31.3	30.8	33.4	54.2	49.4	60.1	29.0
57.5	54.6	59.1	50.3	42.5	50.3	72.0	27.4	55.2	46.9
57.4	30.4	57.0	16.2	62.5	41.1	58.3	37.3	47.4	47.9
49.2	40.7	49.3	40.0	49.0	37.9	50.6	32.4	46.8	39.3
39.3	45.8	52.2	51.4	56.5	42.0	66.9	30.9	60.1	38.3
61.9	36.5	43.2	26.4	46.8	33.4	69.8	41.1	57.2	30.9
33.4	40.5	55.4	36.4	61.4	57.7	67.1	27.3	52.9	41.5
60.7	42.2	65.1	56.4	52.7	23.7	62.2	35.8	67.1	33.4
49.5	36.4	38.9	31.0	45.1	34.8	64.1	42.2	49.9	43.6
47.7	49.6	67.3	44.2	60.0	39.1	51.0	48.2	53.6	30.6
63.0	40.8	58.9	24.5	63.7	34.4	63.3	38.4	39.1	49.4
74.7	51.1	52.0	27.3	38.8	35.8	58.0	50.7	70.8	37.1
63.0	40.8	30.0	27.7	74.1	32.6	59.1	46.6	56.3	44.7
67.3	58.6	52.0	42.5	50.9	51.9	71.3	35.7	67.1	40.8
46.9	29.5	44.4	41.9	49.9	27.0	68.3	32.1	55.7	42.4
61.7	34.2	58.9	33.4	68.2	27.6	62.2	40.8	54.9	47.8
46.6	31.7	59.7	62.5	48.1	42.5	67.4	43.3	54.6	32.6
52.3	46.9	71.9	38.4	52.0	59.4	78.5	33.1	39.5	45.9
60.1	53.6	18.3	55.0	61.0	38.0	42.9	50.1	55.4	40.0
29.5	39.9	54.7	51.8	55.2	49.6	53.7	32.7	77.5	52.7
52.5	51.5	53.3	46.8	55.2	40.1	58.4	36.6	60.5	31.7
53.9	31.1	48.2	42.1	56.1	38.0	55.7	47.3	59.7	28.7
47.4	49.2	39.0	33.9	50.9	46.9	60.0	41.2	68.2	34.2
53.3	40.2	69.4	42.6	53.4	26.1	62.9	46.4	47.8	37.3
46.2	38.7	36.4	39.6	49.7	24.7	49.4	24.9	52.6	36.9
55.3	38.7	62.3	34.9	57.1	38.0	63.7	47.1	55.6	52.6
58.5	16.3	54.3	37.4	72.6	32.5	39.7	50.1	52.6	41.7

Table B.1: Performance results for the SNS approach applied to the Hexapod robot

HBE	HBN	HDE	HDN	HEE	HEN	HME	HMN	HSE	HSN
1.29	0.71	0.56	0.93	1.07	0.54	0.18	0.04	0.67	0.17
0.44	1.05	0.84	0.13	0.56	0.93	0.64	0.50	0.74	0.11
0.97	0.50	0.87	0.84	1.95	0.36	0.37	0.27	0.48	0.20
0.99	0.68	0.27	1.01	1.64	1.04	0.35	0.23	0.32	0.83
0.84	0.46	0.63	0.40	1.26	0.37	0.28	0.78	0.55	0.18
0.83	1.18	0.67	2.68	0.48	0.62	0.39	0.39	0.74	0.31
1.16	0.62	0.86	1.02	1.04	0.63	0.44	0.58	0.81	0.55
1.38	0.43	0.85	0.61	0.80	0.25	0.43	0.63	0.36	0.49
0.82	0.94	1.34	2.15	1.02	0.84	0.24	0.41	0.49	0.71
1.90	0.84	0.46	0.70	0.62	0.23	0.41	1.18	0.43	0.37
0.49	0.29	0.52	0.36	0.83	1.46	0.50	0.53	0.28	0.54
0.95	0.76	1.37	1.12	1.06	1.10	0.34	0.28	0.70	0.29
0.84	0.71	0.41	0.64	0.74	0.57	0.56	0.40	0.56	0.60
0.81	0.80	0.69	1.23	0.67	1.04	0.47	0.25	1.03	0.30
0.32	0.28	0.90	0.90	1.65	0.86	0.50	0.20	0.25	0.27
0.72	0.87	2.07	1.49	0.33	0.98	0.58	0.15	0.42	0.12
0.55	0.28	0.82	0.69	0.77	0.39	0.12	0.47	0.10	0.41
1.43	1.08	1.11	0.92	0.96	1.32	0.22	0.58	0.56	0.33
0.38	0.99	0.43	1.12	0.44	1.31	0.49	0.49	0.53	0.19
1.08	1.28	0.41	0.35	0.98	0.67	0.51	0.49	0.43	0.65
0.70	0.82	0.35	0.66	0.80	0.20	0.11	0.61	0.78	0.22
0.65	0.55	2.95	0.16	0.57	0.66	0.50	0.18	0.49	0.26
1.82	0.79	0.61	0.35	0.68	0.44	0.53	0.47	0.25	0.12
0.95	0.24	0.50	0.68	0.81	0.65	0.35	0.10	0.36	0.88
0.74	0.92	0.78	0.46	0.68	0.78	0.33	0.26	0.48	0.75
1.13	0.54	1.24	0.89	0.83	0.42	0.43	0.29	0.27	0.32
0.66	0.73	0.39	0.42	0.79	1.22	0.41	0.27	0.70	0.45
1.08	1.09	1.09	0.88	0.95	1.44	0.63	0.97	0.45	0.41
0.93	0.47	0.41	0.69	0.67	0.92	0.24	0.29	0.60	0.22
0.58	2.93	0.70	0.74	0.45	1.08	0.98	0.13	0.66	0.57

Table B.2: Transferability results for the SNS approach applied to the Hexapod robot

SBE	SBN	SDE	SDN	SEE	SEN	SME	SMN	SSE	SSN
-16.0	110.9	-54.4	68.0	54.7	36.6	14.7	7.6	23.5	0.6
65.5	5.6	-8.3	34.2	74.6	25.3	12.4	5.2	12.1	43.1
30.8	22.2	76.7	34.6	-30.8	25.2	-21.4	49.2	-4.4	5.4
-100.0	68.8	68.9	72.4	33.9	-2.9	6.9	24.3	-6.8	11.6
62.0	64.7	19.6	76.1	48.7	39.3	8.9	0.4	-2.6	21.7
64.8	6.4	-7.0	13.8	62.0	-22.3	26.1	33.3	19.6	-0.6
-1.3	2.6	14.0	2.1	52.8	38.1	6.8	15.3	28.8	14.7
66.1	64.7	92.5	48.8	85.8	56.3	71.4	3.4	8.3	8.0
32.0	55.1	11.1	-18.4	76.2	58.1	20.0	25.4	-0.8	3.2
-19.9	73.3	40.9	-1.5	79.9	-7.7	34.5	-1.1	23.3	10.5
39.1	44.3	56.9	45.9	5.9	-3.6	9.7	13.4	-1.6	6.9
79.7	70.8	17.4	0.7	49.8	-3.4	19.0	8.3	97.0	10.1
-19.5	44.2	13.3	54.7	9.5	58.2	46.1	40.3	-4.3	67.3
-2.7	42.9	75.8	74.1	46.9	27.6	50.4	35.7	2.0	49.0
44.9	17.3	54.2	45.4	87.1	66.7	14.2	1.8	13.7	-8.6
66.5	42.8	69.2	68.3	18.6	-1.6	35.4	52.3	7.0	7.1
2.4	-15.1	58.2	44.2	60.2	93.2	39.5	10.9	17.9	1.6
93.1	31.7	45.0	1.7	35.5	61.9	16.8	3.8	10.1	23.2
45.4	-4.7	38.5	41.3	49.6	63.8	34.4	23.9	11.4	29.7
24.5	73.1	7.8	40.7	25.9	87.6	-1.4	6.6	6.6	4.5
-64.4	20.8	35.1	-19.9	31.3	-11.5	12.9	23.4	8.6	-6.4
58.9	-9.0	34.0	18.3	20.4	41.8	20.5	13.6	42.3	27.9
25.6	23.8	-6.4	69.9	64.6	21.9	11.3	6.4	7.3	7.6
65.1	41.5	51.7	42.9	18.5	48.3	4.6	16.3	-0.4	16.5
74.7	47.0	81.0	37.8	65.7	69.3	48.3	26.0	28.7	5.1
0.2	-61.0	4.2	-34.7	52.1	-10.2	44.2	6.0	22.1	12.9
4.8	32.8	50.6	74.0	106.4	73.4	23.9	4.9	0.2	23.6
1.2	13.3	87.3	68.0	-22.2	60.4	1.1	11.1	-1.6	-1.9
76.6	9.5	-4.6	43.5	58.2	-5.9	21.5	53.4	5.6	-1.5
-15.6	27.9	85.0	61.8	-44.8	18.4	1.5	5.2	17.6	15.2

Table B.3: Performance results for the SNS approach applied to the Snake robot

SBE	SBN	SDE	SDN	SEE	SEN	SME	SMN	SSE	SSN
12.61	0.45	3.53	1.06	2.20	1.61	3.43	4.58	4.25	7.63
4.16	3.95	31.24	1.84	2.59	3.73	4.64	2.97	10.45	0.60
10.95	1.28	2.17	2.09	6.38	2.65	9.41	0.56	7.79	4.85
4.75	0.55	1.88	1.13	4.65	8.31	5.89	0.99	5.21	3.55
4.31	1.41	13.58	0.66	1.74	2.01	3.51	4.02	28.94	1.26
7.75	8.58	29.32	9.98	3.36	2.15	2.78	0.98	5.78	4.00
144.00	23.89	12.72	6.77	2.48	0.90	3.60	1.67	3.07	2.47
3.84	0.63	2.45	1.13	1.51	0.96	0.60	2.90	14.48	8.13
13.43	1.36	2.87	8.50	1.47	2.26	4.25	1.16	3.19	6.31
10.80	0.66	6.45	6.23	2.20	14.11	1.55	36.69	3.21	4.52
6.33	2.31	3.73	2.04	7.03	25.59	9.17	3.52	5.74	1.62
2.99	1.04	15.51	151.14	2.25	11.04	2.55	8.13	0.57	3.92
18.17	1.64	3.36	0.57	14.05	0.86	1.22	1.60	8.97	0.62
46.22	1.39	2.60	0.37	2.75	1.25	0.81	0.89	72.91	0.57
7.07	3.42	3.71	1.49	1.70	0.10	4.57	5.07	9.35	6.35
6.42	2.00	2.36	0.58	6.86	4.20	1.25	0.33	8.33	6.21
36.82	2.04	3.42	0.44	3.00	0.30	1.57	1.44	4.08	10.04
2.62	2.40	1.87	9.16	2.61	0.68	4.04	12.88	14.68	1.09
3.75	15.52	6.17	1.66	3.55	1.08	1.97	1.47	8.68	1.73
15.93	0.70	27.44	2.89	4.56	0.56	3.52	2.60	5.77	3.38
8.75	3.17	7.28	6.40	4.36	8.27	3.95	1.60	6.87	9.01
4.39	12.86	4.65	3.52	2.28	1.09	2.39	4.01	2.05	1.42
9.23	4.47	3.09	0.49	1.09	2.99	6.07	3.04	20.92	2.89
7.43	4.11	4.10	2.03	8.57	1.04	5.14	2.43	6.62	3.44
2.94	1.23	2.10	2.03	2.84	0.36	0.88	1.10	4.07	2.34
113.56	3.40	75.28	2.56	2.37	6.32	1.01	5.45	7.31	2.69
66.11	1.91	2.77	1.09	1.29	0.68	2.40	5.39	3.51	1.35
43.48	5.20	2.41	1.05	11.79	0.80	45.89	2.80	18.33	7.02
3.74	7.26	30.10	1.55	2.65	9.63	1.82	0.57	24.91	8.47
9.20	2.62	2.00	0.78	2.91	4.78	47.78	6.36	3.41	2.65

Table B.4: Transferability results for the SNS approach applied to the Snake robot

HBSET	HBSNT	HESEU	HESNU	HMSEU	HMSNU
53.1	55.6	67.6	50.1	36.8	41.2
39.1	73.0	41.0	81.3	51.9	50.8
42.8	61.7	46.3	49.1	38.3	50.8
69.5	65.8	62.1	38.0	44.6	63.2
52.6	20.6	57.7	31.7	59.5	40.0

Table B.5: Validation performance results for the Hexapod robot

HBSET	HBSNT	HESEU	HESNU	HMSEU	HMSNU
0.54	0.09	0.09	0.50	0.35	0.78
0.41	0.12	0.66	0.22	0.45	0.41
0.48	0.42	0.72	0.38	0.55	0.57
0.16	0.28	0.13	1.05	0.30	0.21
0.50	1.34	0.47	0.88	0.06	0.73

Table B.6: Validation transferability results for the Hexapod robot

SBSNT	SESNU	SESEU	SBSET	SMSNU	SMSEU
84.5	33.7	78.5	87.0	85.8	6.7
43.3	45.6	144.7	45.1	53.8	49.2
39.5	14.8	30.7	49.4	-13.1	53.2
34.1	46.4	83.3	109.7	62.2	70.7
-16.3	22.3	8.1	13.7	61.5	37.2

Table B.7: Validation performance results for the Snake robot

SBSET	SBSNT	SESEU	SESNU	SMSEU	SMSNU
2.45	0.45	0.94	1.28	10.70	0.38
2.39	0.55	0.59	0.15	0.11	0.34
3.20	1.29	2.59	1.34	0.61	6.98
1.13	0.70	0.97	0.53	0.49	0.78
4.87	3.78	3.51	0.94	0.47	0.58

Table B.8: Validation transferability results for the Snake robot

	Mean	Median	First Quartile	Third Quartile	Standard Deviation
HBBET	27.8	28.0	22.2	30.9	14.3
HBBNT	26.6	24.8	14.7	29.5	17.2
HBCET	19.5	18.6	13.1	22.8	12.0
HBCNT	14.3	13.6	8.0	19.3	8.4
HBNET	13.9	9.6	7.3	18.4	11.6
HBNNT	13.8	13.4	8.2	16.5	9.5
HBSET	37.4	37.3	24.8	47.0	16.5
HBSNT	33.7	28.9	21.4	41.3	17.3
HDBET	11.9	12.5	6.7	16.2	8.0
HDBEU	13.2	12.7	10.5	16.6	5.3
HDBNT	10.7	11.4	7.9	14.7	5.6
HDBNU	13.2	13.0	9.6	15.7	6.3
HDCET	7.7	8.3	2.4	12.3	5.8
HDCEU	9.9	9.2	5.6	14.4	6.0
HDCNT	9.2	9.3	5.5	13.2	5.0
HDCNU	10.7	10.5	7.6	14.0	6.8
HDNET	9.2	8.0	3.5	14.3	7.2
HDNEU	8.3	8.9	4.3	12.6	5.4
HDNNT	8.5	8.2	5.7	10.9	4.5
HDNNU	7.5	7.2	5.2	10.7	4.9
HDSET	21.6	21.9	13.9	28.5	14.1
HDSEU	23.9	23.1	17.2	27.5	12.2
HDSNT	22.5	23.6	14.1	28.1	11.1
HDSNU	26.7	23.6	18.0	32.4	12.5
HEBET	24.6	22.5	16.0	33.9	12.6
HEBEU	27.0	24.8	20.8	35.3	13.3
HEBNT	16.1	17.8	8.2	22.2	9.3
HEBNU	16.2	13.8	10.3	18.7	7.8
HECET	17.7	18.4	12.3	23.0	8.0
HECEU	16.1	15.2	9.7	21.9	8.0
HECNT	11.8	11.8	8.3	14.3	5.4
HECNU	13.3	13.7	7.8	17.1	6.8
HENET	17.0	14.0	10.3	18.9	11.7
HENEU	17.0	15.4	11.9	20.7	11.9
HENNT	8.3	7.7	2.8	13.4	6.9
HENNU	11.0	9.2	6.0	15.6	8.2

HESET	45.9	41.6	34.0	59.7	18.0
HESEU	47.9	42.0	35.3	61.2	17.1
HESNT	35.8	34.3	23.3	45.7	18.1
HESNU	32.1	32.4	19.4	37.5	16.6
HMBET	37.0	37.3	25.4	45.0	15.1
HMBEU	40.7	36.7	28.4	52.9	17.5
HMBNT	24.1	22.2	13.7	27.7	16.6
HMBNU	23.1	22.3	18.0	28.8	10.2
HMCET	25.2	24.7	21.7	30.4	9.8
HMCEU	21.7	19.3	13.6	31.3	12.5
HMCNT	16.5	17.9	12.2	23.0	7.3
HMCNU	15.8	15.7	12.6	18.3	4.9
HMNET	23.1	20.0	14.5	28.6	14.9
HMNEU	21.4	17.0	11.6	26.8	15.4
HMNNT	14.4	14.6	7.7	21.1	8.2
HMNNU	16.2	15.3	11.3	20.2	12.2
HMSET	43.1	40.0	32.1	52.5	13.5
HMSEU	42.7	42.6	30.2	49.7	19.6
HMSNT	37.0	32.6	26.5	47.2	17.0
HMSNU	36.8	35.7	27.4	43.6	17.3
HSBET	36.4	35.9	23.7	40.1	19.0
HSBNT	26.4	23.6	13.7	33.9	17.0
HSCET	17.3	16.4	11.3	23.5	9.5
HSCNT	15.9	15.8	11.0	20.0	9.3
HSNET	18.3	18.0	8.9	24.2	12.2
HSNNT	15.8	14.6	9.9	23.2	8.9
HSSET	37.4	32.9	20.8	42.6	23.2
HSSNT	30.9	31.9	16.9	38.9	17.0

Table B.9: Performance summary for the simulated BNS experiments of the Hexapod robot

	Mean	**Median**	**First Quartile**	**Third Quartile**	**Standard Deviation**
HBBET	1.14	0.87	0.60	1.16	0.96
HBBNT	1.15	0.88	0.59	1.27	0.84
HBCET	2.40	2.13	1.32	3.06	1.44

HBCNT	3.28	2.48	1.84	3.35	2.43
HBNET	4.82	3.86	1.75	6.92	3.59
HBNNT	4.42	2.94	1.84	5.19	3.88
HBSET	0.70	0.54	0.24	0.94	0.65
HBSNT	0.77	0.72	0.37	0.92	0.62
HDBET	3.96	2.87	1.82	5.28	3.25
HDBEU	3.41	2.94	1.82	3.35	2.69
HDBNT	5.06	3.13	1.90	4.48	7.40
HDBNU	2.95	2.23	1.80	3.61	1.92
HDCET	8.16	4.24	2.85	8.93	9.35
HDCEU	4.85	4.00	2.73	5.76	2.81
HDCNT	5.21	4.78	2.82	6.17	3.43
HDCNU	4.83	3.76	2.61	4.87	4.13
HDNET	6.02	5.41	2.97	7.95	4.05
HDNEU	9.72	4.87	3.41	7.38	13.67
HDNNT	6.17	4.27	3.06	6.53	6.48
HDNNU	8.72	5.66	3.91	9.20	9.60
HDSET	2.34	1.28	0.93	2.17	2.49
HDSEU	1.50	1.22	0.87	1.63	1.23
HDSNT	1.78	1.05	0.77	1.82	1.98
HDSNU	1.22	1.07	0.83	1.60	0.61
HEBET	1.19	0.77	0.52	1.33	1.27
HEBEU	1.15	0.63	0.43	1.12	1.57
HEBNT	2.13	1.17	0.55	1.88	3.92
HEBNU	1.33	1.30	0.56	1.87	0.86
HECET	2.26	1.49	1.17	2.25	1.93
HECEU	2.59	1.88	1.06	3.12	2.22
HECNT	3.57	2.81	1.86	3.34	3.40
HECNU	3.03	2.25	1.49	3.93	2.39
HENET	2.89	2.70	1.17	3.50	2.32
HENEU	2.78	1.88	1.38	3.95	2.22
HENNT	8.13	3.63	1.94	5.53	13.19
HENNU	4.90	3.52	1.76	5.64	6.05
HESET	0.40	0.38	0.17	0.53	0.27
HESEU	0.39	0.36	0.18	0.55	0.24
HESNT	0.56	0.35	0.17	0.69	0.57

HESNU	0.68	0.49	0.36	0.83	0.54
HMBET	0.65	0.47	0.32	0.94	0.48
HMBEU	0.61	0.47	0.32	0.75	0.53
HMBNT	1.02	0.84	0.45	1.38	0.83
HMBNU	1.37	0.73	0.41	1.07	2.65
HMCET	1.42	1.28	0.79	1.63	0.99
HMCEU	2.42	1.51	0.86	2.31	3.02
HMCNT	3.77	1.60	1.17	2.78	8.14
HMCNU	2.20	1.93	1.56	2.64	1.24
HMNET	2.30	1.73	1.01	2.35	2.56
HMNEU	2.39	2.20	0.95	3.09	1.75
HMNNT	2.92	2.10	1.61	3.53	2.28
HMNNU	3.05	2.29	1.32	3.28	4.10
HMSET	0.50	0.44	0.31	0.65	0.28
HMSEU	0.96	0.49	0.25	0.77	2.35
HMSNT	0.90	0.49	0.32	0.79	2.19
HMSNU	0.63	0.51	0.21	0.75	0.70
HSBET	0.87	0.54	0.38	1.15	0.79
HSBNT	1.44	0.88	0.40	1.85	1.57
HSCET	2.98	2.38	1.43	3.64	2.11
HSCNT	5.17	2.65	1.76	4.62	9.64
HSNET	3.20	2.71	1.20	3.65	2.67
HSNNT	3.55	2.51	1.76	4.95	2.76
HSSET	0.97	0.62	0.36	1.35	0.91
HSSNT	1.06	0.64	0.32	1.27	1.21

Table B.10: Transferability summary for the simulated BNS experiments of the Hexapod robot

	Mean	**Median**	**First Quartile**	**Third Quartile**	**Standard Deviation**
SBBET	55.9	54.7	45.5	69.9	33.8
SBBNT	33.2	31.6	21.7	45.7	18.1
SBCET	59.8	59.3	37.6	76.0	32.9
SBCNT	37.8	37.2	21.3	50.2	21.6
SBNET	88.5	74.0	50.9	117.8	54.7
SBNNT	37.0	36.1	21.2	53.7	30.7

SBSET	63.6	60.1	42.1	79.0	44.0
SBSNT	41.9	43.6	26.7	56.0	22.6
SDBET	40.4	41.9	22.8	56.3	26.3
SDBEU	43.4	40.0	26.7	53.1	20.4
SDBNT	45.0	43.1	22.2	64.0	29.3
SDBNU	36.5	35.4	20.9	49.4	21.9
SDCET	36.6	35.1	25.6	44.9	17.2
SDCEU	37.5	35.4	29.0	46.9	21.2
SDCNT	33.5	37.8	20.6	44.3	24.2
SDCNU	39.4	39.7	25.4	51.4	20.7
SDNET	53.6	54.5	35.8	70.0	25.2
SDNEU	55.6	53.0	40.2	70.5	23.4
SDNNT	46.8	46.6	30.3	63.7	23.5
SDNNU	43.6	44.4	30.0	56.3	22.2
SDSET	45.5	44.4	30.9	61.4	26.0
SDSEU	52.8	42.3	35.9	71.3	28.4
SDSNT	38.9	35.3	21.7	52.1	25.6
SDSNU	47.4	44.2	31.5	61.7	21.0
SEBET	46.2	40.9	29.3	66.4	25.6
SEBEU	55.6	54.0	34.8	76.0	33.2
SEBNT	39.7	36.9	31.3	49.7	17.4
SEBNU	39.1	36.8	28.8	53.4	18.3
SECET	44.1	44.3	32.5	53.8	27.3
SECEU	47.0	42.1	29.5	65.2	22.3
SECNT	43.4	37.1	28.4	58.3	24.5
SECNU	30.6	31.8	16.8	45.7	17.4
SENET	80.7	83.4	57.6	101.4	41.0
SENEU	84.7	88.1	65.8	110.2	41.2
SENNT	43.7	47.1	29.2	54.8	21.8
SENNU	42.4	43.6	33.8	53.0	21.5
SESET	87.3	84.7	73.0	117.5	50.4
SESEU	98.6	97.0	76.4	130.5	44.1
SESNT	45.6	39.3	24.2	62.5	29.0
SESNU	48.1	45.1	29.5	64.8	25.5
SMBET	56.6	54.9	39.4	72.8	25.4
SMBEU	52.6	50.1	38.4	65.1	21.3

SMBNT	43.6	42.6	35.4	53.3	16.9
SMBNU	35.3	33.0	22.4	46.5	17.0
SMCET	57.3	55.8	36.6	70.3	27.3
SMCEU	53.7	44.7	29.5	67.9	31.3
SMCNT	43.5	43.9	35.2	51.1	21.5
SMCNU	38.9	35.3	25.5	51.8	22.1
SMNET	71.1	60.3	53.0	73.7	41.2
SMNEU	75.2	66.0	38.9	105.8	42.0
SMNNT	43.7	41.8	27.9	58.3	21.6
SMNNU	44.2	40.5	31.3	51.7	24.1
SMSET	82.2	73.8	56.7	112.5	41.8
SMSEU	78.0	78.6	52.0	103.0	35.7
SMSNT	38.9	39.5	29.6	49.8	21.8
SMSNU	42.7	41.6	34.3	53.5	15.2
SSBET	73.4	69.4	46.1	85.5	42.1
SSBNT	30.4	32.0	16.4	43.9	18.4
SSCET	69.8	67.9	45.8	92.9	30.8
SSCNT	40.7	41.3	25.6	51.7	19.8
SSNET	130.7	113.5	68.2	172.6	89.7
SSNNT	41.9	36.8	28.1	57.0	21.7
SSSET	95.7	78.5	59.6	115.3	65.1
SSSNT	42.0	38.8	28.8	48.7	21.5

Table B.11: Performance summary for the simulated BNS experiments of the Snake robot

	Mean	**Median**	**First Quartile**	**Third Quartile**	**Standard Deviation**
SBBET	1.78	1.39	1.02	2.08	1.23
SBBNT	1.48	1.09	0.77	1.82	1.56
SBCET	1.91	1.59	1.08	2.36	1.18
SBCNT	1.89	1.40	0.84	2.65	1.51
SBNET	1.49	1.43	1.12	1.69	0.77
SBNNT	1.93	1.55	0.94	2.55	1.48
SBSET	1.76	1.34	0.85	1.68	1.66
SBSNT	1.73	1.21	0.68	1.80	2.67
SDBET	1.97	1.51	1.22	2.68	1.28
SDBEU	1.76	1.72	1.13	2.27	0.99

SDBNT	1.02	0.86	0.58	1.16	0.74
SDBNU	1.63	1.13	0.74	2.11	1.42
SDCET	1.87	1.64	1.32	2.19	1.25
SDCEU	1.82	1.72	1.07	2.32	1.01
SDCNT	1.51	1.10	0.74	1.64	1.66
SDCNU	1.51	1.50	0.78	2.07	1.03
SDNET	1.68	1.29	0.85	1.58	1.42
SDNEU	1.30	1.19	0.77	1.70	0.69
SDNNT	1.26	1.09	0.72	1.36	1.03
SDNNU	1.24	1.08	0.76	1.44	0.69
SDSET	1.65	1.58	1.23	2.18	0.72
SDSEU	1.65	1.38	0.94	1.97	1.11
SDSNT	1.94	1.50	0.91	2.13	1.61
SDSNU	1.22	1.12	0.76	1.37	0.73
SEBET	1.86	1.66	1.18	2.18	1.39
SEBEU	1.71	1.51	1.13	1.99	0.98
SEBNT	1.01	0.86	0.48	1.28	0.71
SEBNU	1.22	1.07	0.67	1.53	0.76
SECET	1.75	1.49	1.29	2.22	0.75
SECEU	1.94	1.55	1.29	2.40	1.14
SECNT	1.19	0.98	0.71	1.16	1.00
SECNU	1.45	1.38	0.92	1.63	0.89
SENET	1.11	0.94	0.78	1.35	0.55
SENEU	1.11	0.91	0.60	1.34	0.78
SENNT	1.16	0.90	0.56	1.57	0.81
SENNU	1.24	1.10	0.73	1.40	0.79
SESET	0.93	0.85	0.59	1.00	0.47
SESEU	0.94	0.92	0.55	1.20	0.54
SESNT	1.25	0.97	0.58	1.30	1.46
SESNU	1.13	0.84	0.55	1.34	0.99
SMBET	1.01	0.89	0.53	1.21	0.63
SMBEU	1.06	1.01	0.68	1.29	0.55
SMBNT	0.86	0.68	0.45	1.02	0.71
SMBNU	1.11	0.89	0.61	1.28	0.98
SMCET	1.16	0.97	0.55	1.74	0.75
SMCEU	1.53	1.19	0.67	1.91	1.38

SMCNT	0.91	0.78	0.46	1.14	0.60
SMCNU	1.30	1.01	0.57	1.64	1.23
SMNET	0.97	0.90	0.73	1.20	0.46
SMNEU	1.02	0.86	0.63	1.47	0.60
SMNNT	1.05	0.87	0.63	1.41	0.61
SMNNU	1.35	0.78	0.45	1.40	2.28
SMSET	0.74	0.68	0.39	0.97	0.57
SMSEU	0.79	0.58	0.38	0.90	0.81
SMSNT	0.95	0.76	0.46	1.16	0.79
SMSNU	0.74	0.70	0.40	0.97	0.40
SSBET	0.94	0.69	0.35	1.20	0.78
SSBNT	1.54	1.18	0.70	1.93	1.31
SSCET	1.36	1.22	0.88	1.72	0.81
SSCNT	1.17	1.13	0.79	1.48	0.67
SSNET	0.89	0.75	0.46	1.07	0.82
SSNNT	1.27	0.91	0.66	1.69	0.96
SSSET	0.84	0.67	0.37	1.26	0.57
SSSNT	1.03	0.97	0.58	1.29	0.68

Table B.12: Transferability summary for the simulated BNS experiments of the Snake robot