CONCEPTUAL DESIGN METHODOLOGY OF DISTRIBUTED INTELLIGENCE LARGE SCALE SYSTEMS

A Thesis
Presented to
The Academic Faculty

by

Bassem R. Nairouz

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
The Daniel Guggenheim School of Aerospace Engineering

Georgia Institute of Technology
August 2013

CONCEPTUAL DESIGN METHODOLOGY OF DISTRIBUTED INTELLIGENCE LARGE SCALE SYSTEMS

Approved by:

Professor Dimitri Mavris,
Committee Chair
School of Aerospace Engineering
Georgia Institute of Technology

Professor Daniel Schrage
School of Aerospace Engineering
Georgia Institute of Technology

Doctor Neil Weston
Aerospace Systems Design Laboratory
Georgia Institute of Technology

Doctor Frank Ferrese
Naval Surface Warfare Center
US Navy

Doctor Brian German
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: 20 June 2013

ACKNOWLEDGEMENTS

I cannot find enough words to express my deepest appreciation to my adviser and committee chair, Professor Dimitri Mavris, who was a mentor, supporter and academic adviser. I owe immense gratitude to my committee co-chair, Doctor Neil Weston, whose advice and persistent feedback went beyond academics.

In addition, I would like to thank my committee members Doctor Frank Ferrese and Doctor Brian German for their guidance and feedback, as well as Professor Daniel Schrage for his encouraging comments and suggestions.

I consider it an honor to have been a member of the Aerospace Systems Design Laboratory. I am indebted to Loretta Carroll who was a continuous helper and guide through the institutes administrative system. I thank my colleagues Doctor Samer Tawfik, Doctor Michael Balchanos, Doctor Matt Hoepfer, Doctor Reza Rezvani, Allison Hill, Mark Danielson, Kara Kelch, and all the ASDL members and staff.

This thesis would have remained a dream had it not been for my parents who believed in me, and pointed at the right path early on. I am extremely grateful for the encouragement, support and help of my wife. She was a fellow traveler in the thesis journey, and her faith in me never failed.

I give all the credit to my Lord and Savior Jesus Christ for being the Friend who sticks closer than a brother.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii

LIST OF TABLES x

LIST OF FIGURES xi

SUMMARY xvi

I INTRODUCTION AND MOTIVATION 1

1.1 Problem Statement 2

1.2 Proposed Method 3

1.3 Intelligent Systems Design Methodology Drivers 5

1.3.1 Physical System / Control Architecture Interaction 6

1.3.2 Control Architecture Consistent Comparison Framework . . 8

1.4 Basic Concepts 10

1.4.1 Large Scale System of Systems 10

1.4.2 Complex Emergent Systems 10

1.4.3 Intelligent Systems 11

1.4.4 Relevant Complex Systems Dissertation Scope 11

1.5 Complex System Design Practice 13

1.6 Implication of the Intelligence Requirement on Complex Systems . 14

1.7 Intelligent Systems Design Practices 15

1.7.1 Modeling 17

1.7.2 Simulation 22

1.7.3 Design for Capability 25

1.8 Software Engineering - Control / Decision Making Architecture Relationship 26

1.9 Motivation 27

1.10 Need for Intelligent Large-Scale Systems 29

1.10.1 The Smart Grid 29

1.10.2 Next Generation Naval Vessels 31

II FUNDAMENTAL CONCEPTS 34

2.1 Control System Principles 34

2.2 Control Architecture Configuration 36

2.2.1 Centralized Control 36

2.2.2 Hierarchical Control 37

2.2.3 Decentralized Control 37

2.2.4 Distributed Control 38

2.3 Control Architecture Methods 39

2.3.1 Classical and Gain Control Methods 40

2.3.2 Model Predictive Control 40

2.3.3 Cooperative Control 41

2.3.4 Network Control 41

2.4 System Stability 41

2.5 Simulation and Modeling Concepts 43

2.5.1 Simulation Based Engineering 44

2.5.2 Model Driven Engineering 45

2.5.3 Metamodels 46

2.5.4 Behavioral Models 47

2.5.5 Forward Dynamic Models 47

2.5.6 Inverse Dynamics Models 47

2.5.7 Dissertation Modeling and Simulation Approach 48

III LITERATURE REVIEW 49

3.1 Decentralized and Distributed Control Architectures 49

3.1.1 Centralized vs. Distributed Control 49

3.2 Intelligent Control 50

3.2.1 Supervisory Control 52

3.2.2 Supervisory Control Architecture Example - Ship Damage Control 53

3.2.3 Agent-Based Control 56
3.2.4 Artificial Intelligence 57
3.2.5 Resource Allocation 58
3.3 Network Control 59
3.4 Model-Based Architecture Representation 59
3.4.1 Unified Modeling Language (UML) 59
3.4.2 Architecture Analysis and Design Language (AADL) 60
3.4.3 Domain Specific Modeling Languages (DSML) Semantics . . 60
3.5 The Reference Control System 61
3.6 Case Study: The F-16 Flight Control System 62
3.6.1 The F-16 Fly-By-Wire System 63
3.6.2 Redundancy Study 67
3.6.3 Performance Limits and Constraints 70
3.7 General Observations 73
3.7.1 Intelligent Systems Design 74
3.7.2 Control Architecture Structure 74
3.7.3 Control Architecture Standards 75
IV INTELLIGENT SYSTEMS DESIGN METHODOLOGY 77
4.1 Traditional Systems Design Process Methodology 77
4.2 Traditional Control System Design Steps 78
4.3 The Controller Design's Three Basic Decisions 83
4.3.1 Control Configuration or Structure 83
4.3.2 Control Method or Technique 85
4.3.3 Controller 85
4.4 The Feedback Control System - Revisited 86
4.5 Hypothesis I: Intelligent Systems Design Methodology 87
4.6 Distributed Intelligence Systems Proposed Design Methodology . . 87
4.7 Feasibility and Initial Assessment 91

4.7.1 State Reachability Analysis 94
4.7.2 Inverse Dynamics Analysis 95
4.7.3 Stability Requirements Analysis 96
4.7.4 Capability Potential Analysis 98
4.8 Summary 98
V INTELLIGENCE AND CONTROL ARCHITECTURE METAMODEL 100
5.1 Introduction 100
5.2 Control Systems Taxonomy 101
5.2.1 Traditional Control Systems Taxonomy 103
5.2.2 Inadequacy for Conceptual Design Phase 105
5.2.3 Novel Control Systems Taxonomy Approach 108
5.2.4 Novel Control Configurations Taxonomy 110
5.2.5 Novel Control Methods Taxonomy 111
5.3 Control Architectures Meta-model 113
5.4 Physical Subsystems vs. Control Framework Decomposition 114
5.5 Intelligent Systems Features - Revisited 115
5.6 Proposed Control Configuration Metamodel 116
5.6.1 Spatial Federation Framework 118
5.6.2 Functional Federation Framework 119
5.7 Proposed Abstract Control Method 119
5.7.1 Essential Modules 121
5.7.2 Non Essential Modules 122
5.8 Hypothesis II.b 124
5.8.1 Experiment II.b.1 - PID Control 125
5.8.2 Experiment II.b.2 - Intelligent Decentralized PID / Adaptive Control 125
VI BENCHMARK PROBLEM: MECHANISM CONTROL ARCHITECTURE 127
6.1 Controlled Robotic Mechanism Design Approach 127
6.1.1 Traditional Approach 129

6.1.2 Physical Design A - Four Link Design 130
6.1.3 Physical Design B - Two Link Design 133
6.2 Control Architecture Analysis Suite 135
6.2.1 State Reachability Analysis 135
6.2.2 Inverse Dynamics Analysis 135
6.2.3 Stability Requirements Analysis 148
6.2.4 Capability Potential Analysis 151
6.2.5 Actuator Placement 152
6.2.6 Interim Summary 154
6.3 Control Architecture Metamodel Description 154
6.4 Mechanism Control Architecture Metamodel 156
6.4.1 Centralized Control Architecture 156
6.4.2 Hierarchical Control Architecture 161
6.4.3 Distributed Control Architecture 166
6.4.4 Decentralized Control Architecture 171
6.5 Control Architecture Demonstration 175
6.5.1 Controller Design Overview 176
6.5.2 Controller Operation - Sensitivity Analysis 177
VII APPLICATION: CONTROL ARCHITECTURE DESIGN OF A MULTI-NETWORK COMPLEX SYSTEM 187
7.1 Introduction 187
7.1.1 Behavioral model 188
7.2 System Behavioral Model Description 189
7.2.1 Fluid Network 189
7.2.2 Electric Network 194
7.2.3 Thermal System 196
7.3 Naval Vessel Control Architecture Analysis Suite 208
7.3.1 State Reachability 210
7.3.2 Inverse Dynamics 211

7.3.3 Stability Considerations 213
7.3.4 Capability and Performance Potential 215
7.4 Naval Vessel Control Architecture Meta-model 216
7.4.1 Control Node Complexity 216
7.4.2 Control Architecture Configuration 219
VIII CONCLUDING REMARKS 242
8.1 Research Objectives Review 242
8.1.1 Objective I: Interaction Between the Physical System and the Control Architecture 244
8.1.2 Objective II: Control Architecture Meta-model 246
8.2 Methodology Limitations 247
8.3 Summary of Contributions 248
8.3.1 Intelligent Systems Hybrid Design Process 248
8.3.2 Control Architecture Meta-model 249
8.4 Recommendations for Future Research 250
APPENDIX A MECHANISM DESIGN PROBLEM COMPUTER CODE . 252
APPENDIX B AMDAHL'S LAW 282
APPENDIX C CYCLOMATIC COMPLEXITY 283
REFERENCES 285

LIST OF TABLES

1 Basic Comparison of UML and AADL 60
2 Typical Controller Matrix of Alternatives 102
3 Mechanisms A & B Energy Consumption 147

LIST OF FIGURES

1 Control Architecture Selection 2
2 Proposed Intelligent Systems General Design Process 4
3 Intelligent Systems Design Methodology Drivers 5
4 First Research Driver Task Breakdown 7
5 Second Research Driver Task Breakdown 9
6 Intelligent Systems Design Process 16
7 Current Control Architecture Design Efforts 18
8 Challenges of Intelligent Systems Design 19
9 The Smart Grid 30
10 Generic Autonomous System 34
11 Hierarchical Control Configuration 37
12 Decentralized Control Configuration 38
13 Distributed Control Configuration 39
14 Model and Simulation Environment 44
15 Traditional Modeling Infrastructure 46
16 Intelligent Control Domain 51
17 Structure of a UAV Hierarchical Controller 52
18 Supervisory Ship Damage Control Architecture 54
19 Single Agent Architecture 57
20 Multi-Agent System Architecture 58
21 Simplified Submarine Maneuvering System 62
22 The F-16 Fly-by-wire Flight Control System 66
23 The F-16 Fly-by-wire Redundancy 68
24 The F-16 Angle of Attack / Load Factor Limiter 71
25 The F-16 Aileron Rudder Interconnect 71
26 The F-16 Rudder Fadeout 73
27 Typical Conceptual Design Steps 78

28 Traditional Controller Design Steps 81
29 Control Architecture Requires Three Major Decisions 84
30 Feedback Control System with Three Basic Decisions 86
31 Distributed Intelligence Systems Proposed Design Methodology . . . 90
32 Feasibility and Initial Evaluation Tests 93
33 Three Major Control Architecture Decisions in Conceptual Design . . 109
34 Novel Control Configuration Taxonomy 110
35 Novel Control Method Taxonomy 113
36 Spatial Federation Framework 119
37 Functional Federation Framework 120
38 Proposed Generalized Control Node 121
39 PID Control Method Representation 125
40 PID/Adaptive Intelligent Hierarchical Control Method Representation 126
41 Design A Diagram - Four Link Mechanism 131
42 Design B Diagram - Two Link Mechanism 134
43 Design A Nominal Configuration - Reachability 136
44 Design B Nominal Configuration - Reachability 137
45 Mechanisms A & B Required Path 138
46 Design A Inverse Dynamics Model - Top Level 139
47 Design A Inverse Dynamics Model - Internal Structure 140
48 Design B Inverse Dynamics Model - Top Level 141
49 Design B Inverse Dynamics Model - Internal Structure 142
50 Design A Inverse Dynamics Analysis - Performance - State History . 144
51 Design A Inverse Dynamics Analysis - Performance - Path 145
52 Design B Inverse Dynamics Analysis - Performance - State History . . 146
53 Design B Inverse Dynamics Analysis - Performance - Path 147
54 Design A & B Inverse Dynamics Analysis - Efficient Operation 148
55 Design A Constraints 150
56 Design A Capability Potential - Accuracy 153

57 Design B Capability Potential - Accuracy 153
58 Centralized Control Architecture Block Diagram 157
59 Centralized Control Architecture Configuration 157
60 Centralized Control Architecture - Configuration Complexity 158
61 Centralized Control Architecture - Node Complexity 159
62 Hierarchical Control Architecture Block Diagram 162
63 Hierarchical Control Architecture Configuration 163
64 Hierarchical Control Architecture - Configuration Complexity 164
65 Hierarchical Control Architecture - Node Complexity 165
66 Distributed Control Architecture Block Diagram 167
67 Distributed Control Architecture Configuration 168
68 Distributed Control Architecture - Configuration Complexity 168
69 Distributed Control Architecture - Node Complexity 169
70 Decentralized Control Architecture Block Diagram 172
71 Decentralized Control Architecture Configuration 172
72 Decentralized Control Architecture - Configuration Complexity 173
73 Decentralized Control Architecture - Node Complexity 173
74 Centralized Control Architecture Implementation 177
75 Trajectory - Segment Defined by Two Boundary Waypoints 178
76 Bar Angles - Segment Defined by Two Boundary Waypoints 179
77 Position Vector Error - Segment Defined by Two Boundary Waypoints 180
78 Trajectory - Segment Defined by Four Internal Waypoints 181
79 Bar Angles - Segment Defined by Four Internal Waypoints 182
80 Position Vector Error - Segment Defined by Four Internal Waypoints 183
81 Trajectory - Segment Defined by Nine Internal Waypoints 184
82 Bar Angles - Segment Defined by Nine Internal Waypoints 184
83 Position Vector Error - Segment Defined by Nine Internal Waypoints 185
84 Trajectory - Segment Defined by Nineteen Internal Waypoints 185
85 Bar Angles - Segment Defined by Nineteen Internal Waypoints 186

86 Position Vector Error - Segment Defined by Nineteen Internal Waypoints 186
87 Fluid Network 190
88 Fluid Network Plant 191
89 Pump Electric Power Characterisitics 192
90 Fluid Network Service (Load) 193
91 Electric Network - High Level 194
92 Electric Junction 195
93 Electric Network - Relay Logic 196
94 Electric Network - Power Panel Assemblies Details 197
95 Compact Heat Exchanger 200
96 Thermal Service Simulation Diagram 201
97 NTU Vs Effectiveness - Cross Flow Single Pass - Both Fluids Unmixed 203
98 Heat Exchanger Simulink Model 205
99 Service Cavity Simulation Model 207
100 Air Circulating Fan Model 208
101 Air Circulating Fan Controller 210
121 Fluid Network Service Metamodel 218
102 Service Temperature Response - 50W Heat Load 227
103 Service Temperature Response - 100W Heat Load 228
104 Service Temperature Response - 150W Heat Load 228
105 Service Temperature Response - 200W Heat Load 229
106 Service Temperature Response - 300W Heat Load 229
107 Fan Power Vs. Energy Consumption 230
108 Fan Controller Switching Cases - $10W$ Fan Power 230
109 Fan Power Vs. Energy Consumption - Stability 231
110 Stabilizing Fan Controller Feasibility Region - $10W$ Fan Power 231
111 Desired Performance Region - $10W$ Fan Power 232
112 Rise Time[s] - $50W$ Heat Load 233
113 Rise Time[s] - $100W$ Heat Load 233

114 Rise Time[s] - 150W Heat Load 234
115 Rise Time[s] - 2000W Heat Load 234
116 Settling Time[s] - 50W Heat Load 235
117 Settling Time[s] - 100W Heat Load 235
118 Settling Time[s] - 150W Heat Load 236
119 Settling Time[s] - 2000W Heat Load 236
120 Fluid Network Isolation Valve Metamodel 237
122 Fluid Network Plant Metamodel 237
123 Electric Network Power Panel Assembly Metamodel 237
124 Thermal Network Service Metamodel 238
125 System Functional Federation Control Architecture 238
126 Strict Functional Federation Control Architecture 239
127 Hybrid Functional - Spatial Federation Control Architecture 240
128 Variant Hybrid Functional - Spatial Federation Control Architecture . 241

SUMMARY

Large-scale intelligent systems constitute an important category within the systems of systems field. Such systems cover a vast range of applications. Vehicles (aircraft, naval vessels, spacecraft, etc.), networked systems (electric grid, fluid networks, wireless networks, etc.), hybrid systems (composed of dissimilar entities with a common goal) are some examples among a multitude of applications.

However, there exists no standard process for the design of such systems. It is a common practice within the intelligent systems field to design the physical system first, without paying attention to the intelligent behavior requirement. Once the physical system is in the detailed design phase (and in some cases after the detailed design is finalized), the intelligent control architecture design starts. This means that the current state of the art is a serial design process, which confines the control architecture design space. The serial design process also results in the design of a control architecture that is limited in performance by the physical system design.

This research presents a design methodology focusing on distributed control architectures while concurrently considering the systems design process in the conceptual design phase.

The methodology is enabled by two major drivers. The first driver relates to the interaction between the physical system and the control architecture. It introduces a hybrid design process, based on the infusion of the control architecture and conceptual system design processes. The control architecture design approach is analyzed and broken down into its elementary components. The resulting control architecture design process is laid side by side with a general complex systems conceptual design process. Points of information exchange between the two processes are identified.

The result is an internal design loop involving both the control architecture and the physical system.

A suite of analyses that can be used to evaluate the matching between the physical system and the control architecture over the design loop is proposed. The analyses rely on having a behavioral dynamic model of the system. A detailed model is not needed; what is important is modeling the general behavior of the system and the interaction between the physical system and the control architecture. The suite comprises four categories of analyses: 1) reachability analysis, 2) stability analysis, 3) inverse dynamics analysis, and 4) capability potential analysis. In addition, actuator and sensor placement is addressed on case by case basis.

The second major driver exclusively relates to the control architecture. It is directed towards developing a control architecture meta-model that standardizes the representation of control architectures. Control architectures are selected by making three main decisions: control configuration, control method, and controller parameter selection. It is shown that the first two decisions may be addressed in the conceptual design phase. However it is essential that a standard classification of controllers is used to span the design space of configurations and methods.

The standard controller classification approach in the literature does not support such task. Control architectures are assembled in a novel classification or taxonomy tree. Hence, the standard representation of a wide range of control architecture frameworks is feasible. Software metrics can be applied to the control architecture meta-model, enabling consistent and fair comparisons of different architectures.

The hybrid design process is applied to the problem of designing a "'two degrees of freedom mechanism"'. Two physical designs are proposed and evaluated from a control architecture perspective. The analyses suite is used to compare both designs, shedding light on the interaction between the mechanism and the control architecture.

Four control architectures are proposed and represented using the standard meta-model. Two software metrics are evaluated for the different architectures and weighed against the qualitative properties of each.

A large-scale system representing several networks on a naval vessel is used to demonstrate the control architecture design methodology. The system is composed of three main networks: an electric network, a fluid network, and a thermal network. The objective of the control architecture is to maintain cooling fluids to six service loads. Each service load has its own compact heat exchanger. The analysis suite is used in conjunction with the control architecture meta-model to obtain more information about the behavior of the system and the complexity of the control architecture. This information is shown to be valuable in later stages of the design at which the control architecture parameters are set.

CHAPTER I

INTRODUCTION AND MOTIVATION

Engineering systems are becoming larger, more complex, and more interdependent with other systems. A large-scale complex system may consist of multiple, highly coupled networks, from different disciplines. Together with the increase in complexity, systems are required to be more intelligent, possessing a control architecture that can drive the system during normal operation, avoid failure when faults occur, and gracefully degrade while maintaining a minimum capability after damage. To accomplish such capabilities, a control architecture has to be designed and matched to the physical large-scale system.

The objective of this research endeavor is to present a methodology for such a design process. The methodology has to answer several design questions:

- Which control or intelligence architecture works better?
- How is it applied to the system? Is it based on a functional decomposition of the system or a spatial decomposition?
- What are the components in the architecture and how do they interact?

As demonstrated by the example in Figure 1, a large-scale complex system consists of three different networks: thermal, fluid, and electrical. Thus the questions that impose themselves here are: What is a suitable control architecture that can be applied to the system such that certain capabilities are achieved? How is this architecture structured, is it centralized, distributed, hierarchical? Is there a difference in capability, or more importantly, are all architectures feasible?

To begin with, some basic concepts will be covered. Then, the complex systems

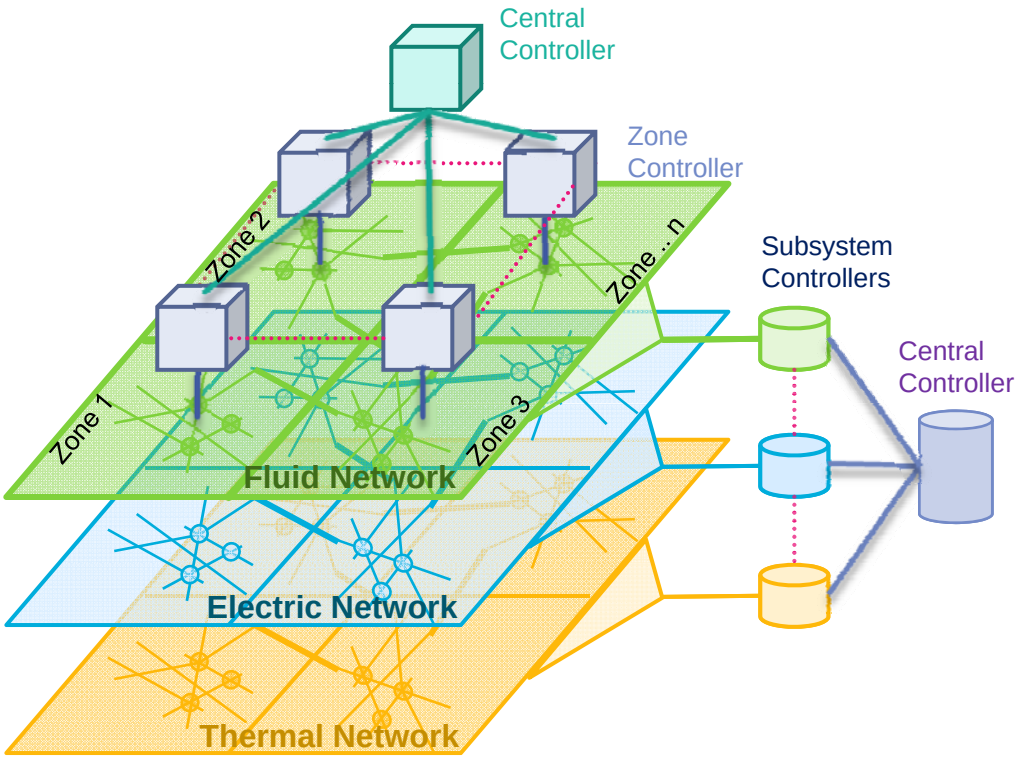


Figure 1: Control Architecture Selection

design practices are presented, followed by the implications of the intelligence requirement on complex systems design. Afterwards, the intelligent systems design practice is explained in detail, together with some of its challenging issues. In addition, an analogy between software engineering and intelligent systems control architectures is then drawn, and finally, a summary of the motivation behind this research is given.

1.1 Problem Statement

Given the major task of designing a prospective large-scale complex intelligent system, such that the system is still in the conceptual design phase, propose and implement a methodology by which this task can be accomplished. The methodology has to include all the tests, analyses, tools, assumptions, and evaluation metrics that are needed for intelligent system design. It is important that this methodology is set in a process form, so as to be clear on what steps to take to fulfill the design process task.

As explained later, intelligent systems are designed in serial process. For example, the physical system is designed first, followed by the design of the control or intelligence architecture. The methodology that is required in this dissertation should assume that the physical system's design is not complete. Hence, a concurrent design methodology of the physical system and the control architecture is sought in this research. Note that the control architecture refers to all the control algorithms, the control components (including actuators and sensors), the controllers on different levels, and the communication between them. The physical system, on the other hand, is everything else in the system including all the hardware, networks, layout, and structure.

1.2 Proposed Method

The methodology is a list of tools that are employed to accomplish a certain task. It lays the foundation for developing a practical design process of intelligent large-scale systems. Figure 2 depicts the intelligent systems design process presented in this research. It is assumed that the physical system is still in the conceptual design phase. Hence, there exists no prototype for the system nor is the complete emergent behavior of the system known. It only assumes that the general behavioral characteristics of the system can be understood and investigated through the solution of the system's dynamic equations. Another assumption is that the physical system's constraints, that either originate from the systems inherent physical characteristics or from the operational limits, are documented and fully comprehended.

The entry point to the process is the characteristics system's behavior and constraints. Analysis of the required functionality of the system is combined with its behavior and constraints in the context of the control architecture. The result of this step is a proposed control architecture structure or configuration. The structure of a control architecture refers to its components, their functionality, their connectivity,

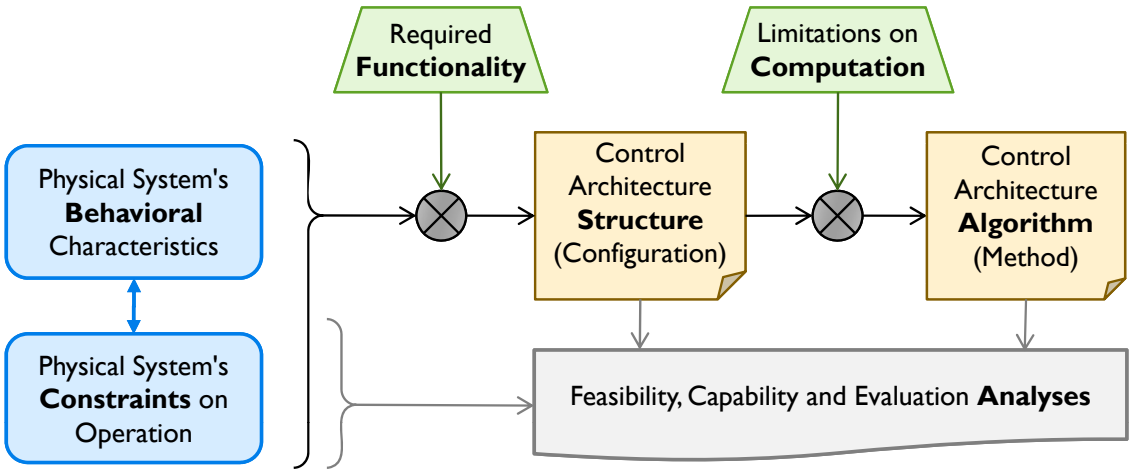


Figure 2: Proposed Intelligent Systems General Design Process

their layout, and information exchanged among them. At this point, several control architectures can accomplish the required functionality of the system.

Since modern control architectures are heavily based on software implemented algorithms, computation issues becomes a very influential factor in control architecture design. The limitations on computation together with the proposed control structure are used in the selection of feasible control algorithm. The control algorithm or method is the general computational approach that the control architecture employs. For example, the control method may conduct full optimization, or simple feedback; it may use a model predictive control approach or a neural network learning approach.

The control architecture structure and the control architecture algorithm, coupled with the system's constraints on operation, are used in feasibility analyses of their combined physical system / control architecture proposed design. It is a necessity to evaluate whether this combination is feasible or not. Another set of analyses are directed towards the capability potential of the proposed system: how well the system can perform and the limits on its performance. The third key set of analyses that needs to be conducted in the design proposed process is the evaluation analyses set. This analyses set is more focused on evaluation and comparison of different

combinations of physical systems / control architectures. Feasibility, capability, and evaluation analyses need not to be mutually exclusive. Some experiments conducted on different intelligent systems can produce information on any of the above three.

1.3 Intelligent Systems Design Methodology Drivers

The target design methodology is a hybrid between the intelligent control architectures design methodologies and standard physical systems design methodologies with all its variations. It is directed towards large-scale systems. The development of an intelligent system design methodology is driven by two main drivers, as shown in Figure 3. The first driver relates to the interaction between the physical system of the control architecture while the second focuses on a consistent control architecture representation and comparison framework.

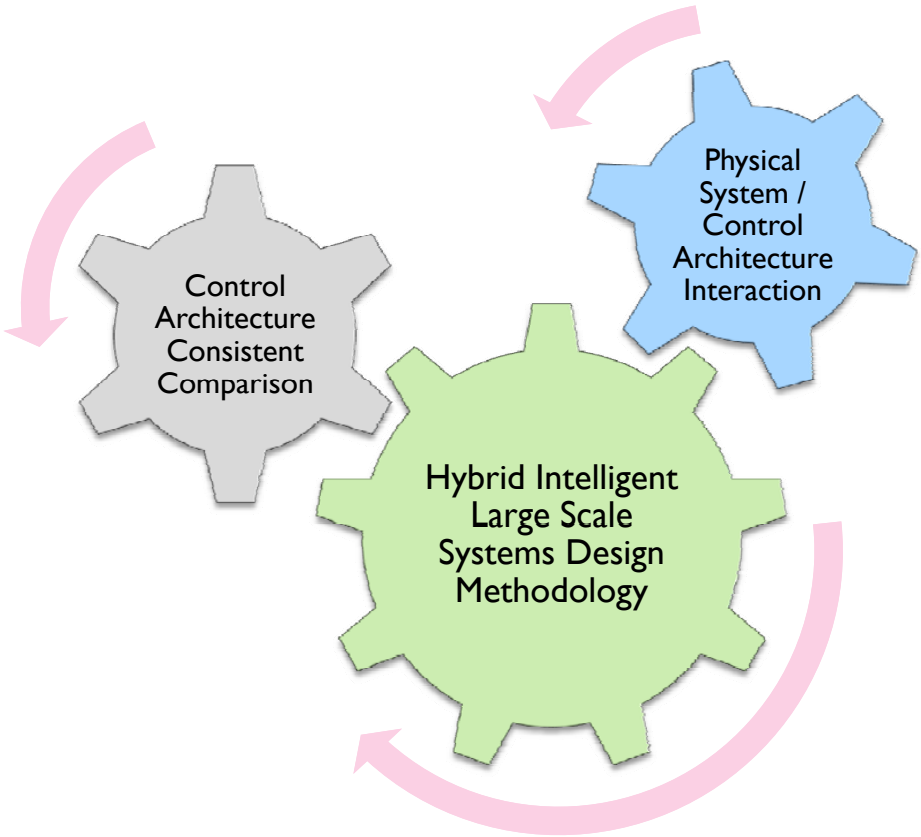


Figure 3: Intelligent Systems Design Methodology Drivers

It is not necessary that the all information is obtained at this level of conceptual design. There are many loose ends that can only be tied in the detailed design phase. However, the more information acquired in the conceptual design phase, the more likely that future problems can be avoided in later steps of the design process. The more complex, less traditional, highly emergent a system is, the less detailed information is determined using this methodology.

1.3.1 Physical System / Control Architecture Interaction

The first driver is the need to identify, investigate, and assess the interaction between the physical system and the control architecture. The effect of the physical system on the control architecture structure and feasible algorithms, such as the satisfaction of system's requirements, has to be determined. For example, a system with a survivability requirement excludes centralized control architectures from the feasible ones available for this system. In addition, the performance limits that the control architecture places on the physical system have to be roughly established. As an example, the particular choice of actuators in a control architecture dictates a certain accuracy of the physical system. If more accuracy is acquired, either the physical system configuration needs to be modified or the control architecture actuators have to be substituted with more accurate ones.

Figure 4 presents a breakdown of the first intelligent system's design methodology driver. A hybrid design process between standard system design processes and control system design process is developed by merging the two types of processes. This is done by the identification of the points of information exchange between the two processes, leading to the satisfaction of the first research objective.

A control system functional decomposition as well as a physical system functional and spatial decomposition are necessary in investigating the interaction between the

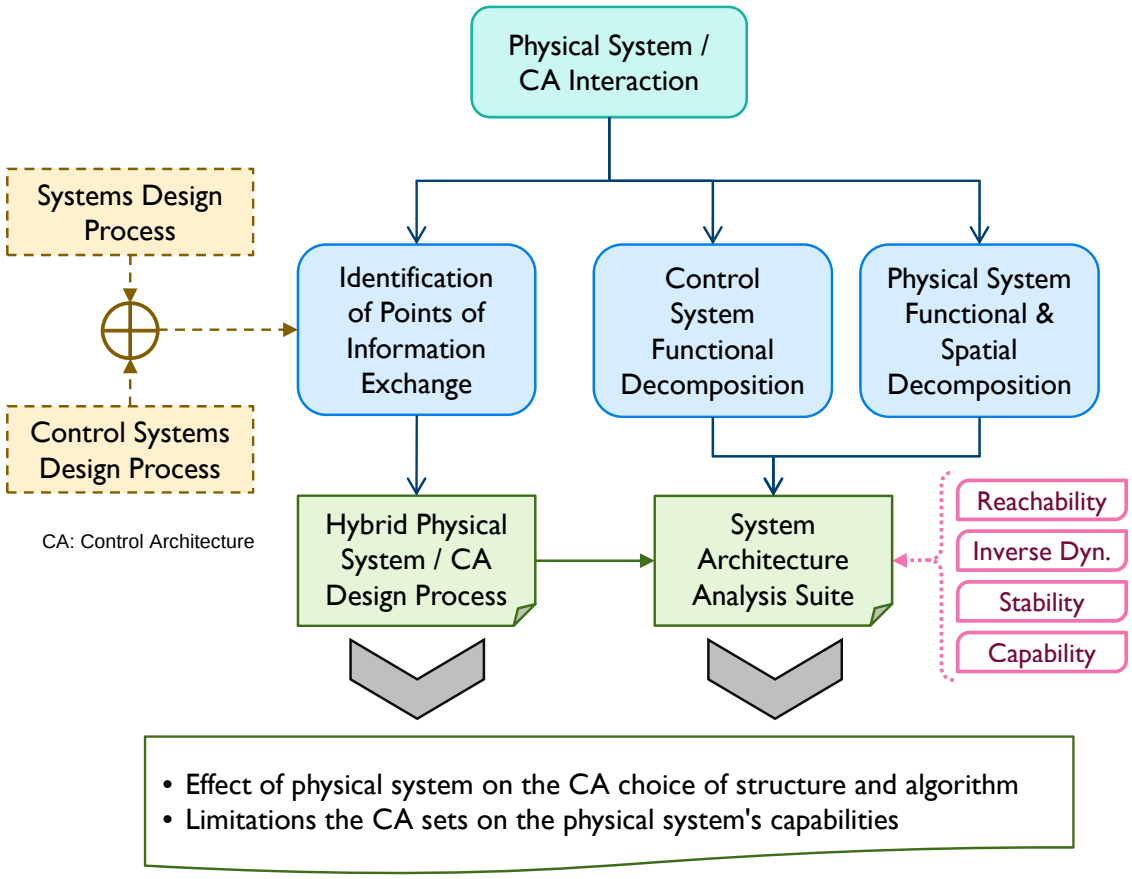


Figure 4: First Research Driver Task Breakdown

physical system and control architecture. This leads to the development of an intelligent system architecture analysis suite that is applicable in the conceptual design phase. The analysis suite may contain any type of analysis of which four are proposed in this research, namely: reachability, inverse dynamics, stability, and performance capability analyses.

The two products from this group of tasks are the hybrid physical system / control architecture design process and the intelligent system architecture analysis suite. Both products provide valuable information about the effect of the physical system on the control architecture and limitations that the control architecture places on the system's capability, old into conceptual design phase.

1.3.2 Control Architecture Consistent Comparison Framework

The second driver is developing a consistent comparison framework for control architectures. Control architectures are seldom compared based on their structure or algorithm. They are usually compared based on the final performance of the system when the control architecture is applied to. Although this might produce many problems as discussed later, it works well for traditional types of systems and for systems that do not require intelligence to be one of the inherent properties. The reason is that the control architecture design is addressed after the physical system design is almost finalized. Hence, a proper control architecture can be built for the physical system, either for the hardware or a simulation model representation. However, the need here is for an intelligent systems design methodology that is applicable in the conceptual design phase. Therefore, a control architecture consistent comparison framework is needed, such that it applies to control architectures beyond evaluating their performance on the physical systems. The main assumption behind this driver is that control engineers, during the detailed design phase, can successfully design and implement a well-chosen control architecture (chosen in the conceptual design phase) to achieve the required system performance. The breakdown of the second methodology driver is shown in Figure 5.

The control architecture consistent comparison driver has two main thrusts. The first thrust is developing a standard representation framework applicable to control architectures. However, developing such a framework requires the investigation of how control systems are classified. It will be shown that control systems are classified in a way that does not facilitate standard representation. Instead, a novel taxonomy or classification of control systems is proposed and is taken as a basis for a control architecture metamodel. Basically, the metamodel abstracts the concepts, components, structure, and algorithms of the control architecture into a standard representation

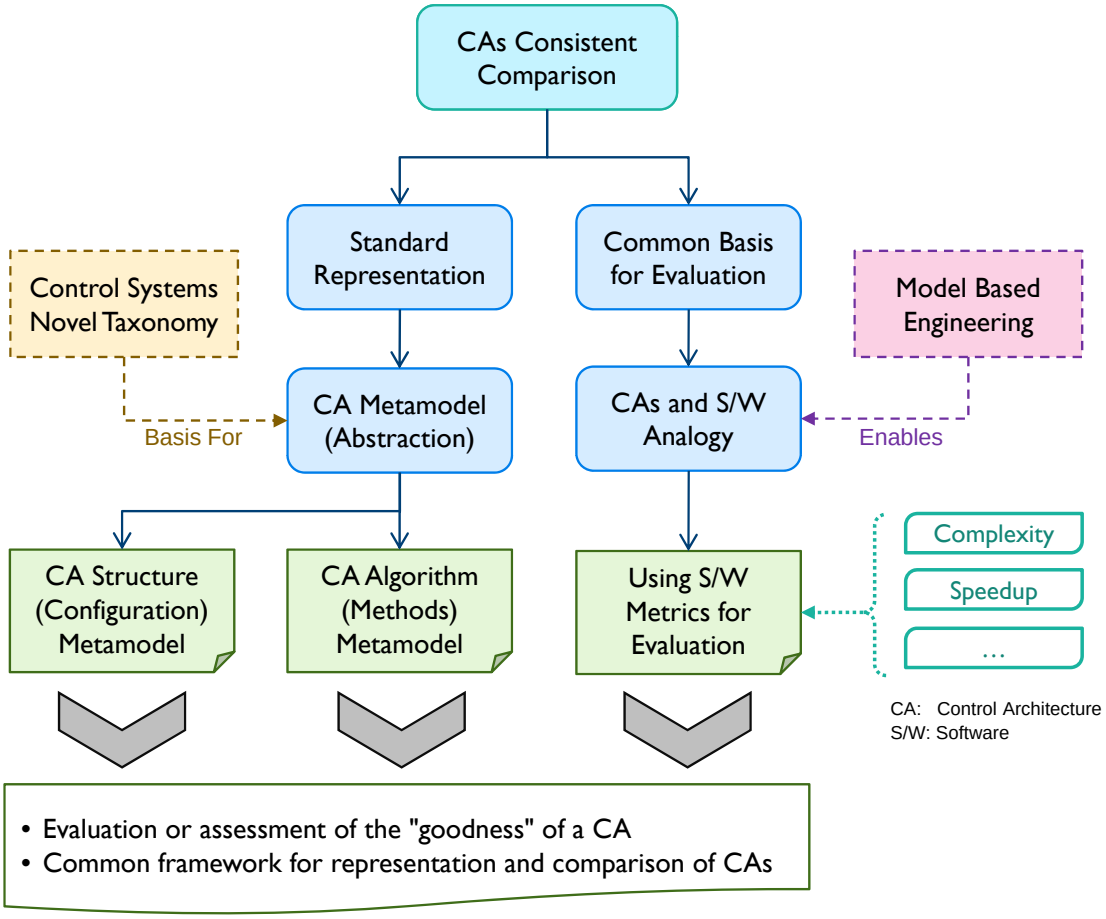


Figure 5: Second Research Driver Task Breakdown

that is applicable across different control architectures. The control architecture configuration and methods are distinguished into their own metamodels as part of the main control architecture metamodel.

The second thrust is developing a common basis of evaluation applicable to control architectures. Currently, the only basis of comparison of controllers is based on traditional system performance measures, such as rise time, steady-state error, ...etc. Traditional measures require the physical system to be designed, which is not the case here. An analogy is made between control architectures and software architectures, enabled by model-based engineering practices. The result is the applicability of many software metrics to control architectures.

The three products from this research driver column: the control architecture structure metamodel, the control architecture algorithm metamodel, and the utilization software metrics for control architecture evaluation provides the designer with more knowledge during the conceptual design phase. An evaluation or assessment of how well the control architecture is designed as well as a common framework to present and compare competing control architectures for the same system are both examples of such knowledge.

1.4 Basic Concepts

1.4.1 Large Scale System of Systems

A large-scale system is defined as a system that can be decomposed into subsystems, such that each subsystem is self-contained but cannot function independently [71], [73]. Large-scale systems can also be defined as systems whose output information can be distributed. They are ever increasing in components, interconnections, and modes of operation.

A System of systems is a large scale concurrent and distributed system composed of complex systems (see [33], [70], and [81]). It all or the majority of these five characteristics are: operational independence, managerial independence, geographic distribution, emergent behavior, and evolutionary development [115]. A System of systems can be described as a method of pursuing development, integration, interoperability, and optimization of systems to enhance performance in future battlefield scenarios [104].

1.4.2 Complex Emergent Systems

Emergent Systems are systems in which a large number of simple components, member entities, and member processes or agents interact in a given environment so as to collectively produce more complex behavior patterns. Emergent behavior occurs in

systems due to interdependency (or interconnectivity). This emergent complex behavior usually cannot be predicted from the consideration of the simple logic inherent to the member agents. Due to the large number of agents, the number of interdependency connections among these agents increases combinatorilly, resulting in an unmanageable set of possible behavior patterns. Yet, the interdependency among the agents need to be considerable and effective. Weak or negligible interdependency does not produce an emergent system. A good example of an emergent system is a colony of ants or a school of fish.

According to an emergent perspective, intelligence emerges from the connections between neurons. From this perspective, it is not necessary to propose a "soul" to account for the fact that brains can be intelligent, even though the individual neurons of which they are made are not.

1.4.3 Intelligent Systems

A collection of software modules that work co-operatively and intelligently towards achieving a common goal defines an intelligent system [92]. Intelligence of such a system is defined as the mechanism that can generate the most advantageous behavior and improves its ability to act effectively and choose wisely between alternative behaviors for achieving the defined goal [92]. Each module or, as is generally termed, a sub-system contributes its share of work towards achieving this goal. Intelligent systems are not built to work in isolation, but are expected to work - autonomously and cooperatively - to adapt themselves to their rapidly changing environment, without the need for repeating a time-consuming learning cycle [123].

1.4.4 Relevant Complex Systems Dissertation Scope

The focus of this dissertation is to address the concurrent design of complex systems and control architectures in the conceptual design phase. Control architectures are addressed from a systems engineer's structure and configuration perspective, rather

than a controls engineer controller design perspective.Conceptual design is a description of how a new product will work and meet its performance requirements. By the end of the conceptual design phase, the design team should be able to answer the following:

- What the system looks like and how is the system configured
- Which components are present in the system
- Means by which these components interact and exchange information
- How much (roughly) the system will cost
- How does the chosen system design excel over other discarded designs
- How does the system accomplish its intended function
- What are the limitations on the system's performance

The concurrent control architecture and physical system conceptual design proves to be a daunting task, since the above information is needed for a controller that controls a system that does not exist yet. The methods developed in this research does not require the full knowledge of the physical system. As a matter of fact, it assumes that only the general behavior of the system (together with its constraints) are known, in one form or another.

Such a research objective cannot be accomplished unless simulation based engineering is utilized, which enables the designer to build a "behavioral model" of the physical system. Mass or energy transfer complex systems are systems whose behavior characteristics is well known. Dynamics, inverse dynamics, or approximate models of any kind can be built for such systems. However, not all large-scale complex systems can be designed using the methods presented later. There are two main types of systems that cannot be used in conjuction with the the concurrent methodology.

Since the methodology relies on behavioral models, the first type of systems that cannot be addressed are systems whose behavior is not fully characterized, extremely stochastic, or not well understood. In addition, the constraints on the physical system have to be fully defined. Also, signal transfer complex systems such as cell phone wireless networks, or queuing systems, or social networks, or medical applications, all have a very different dynamics representation forms that cannot be used with the current methods.

1.5 Complex System Design Practice

Complex systems can be broadly classified in different ways, one of which is classification into two main categories: preexisting and brand-new complex systems. Preexisting complex systems are current updated, extended or modified in any way. They also include systems that have been previously designed and partially implemented. Complex systems that have not been built yet, but have existing implemented identical (or close to identical) instances also fall in this category. On the other hand, brand-new complex systems explore uncharted domain. Usually most of the system within this domain are revolutionary systems without prior design and implementation history of their integrated setup. As with many other system of systems concepts, there is no sharp line that divides both categories. Instead, a gray area exists where some researchers regard particular systems as preexisting and others as brand new.

The design of complex systems is usually done in several steps. Major subsystems are designed first, independently, then loosely integrated together to form the complex system backbone. The gaps between the subsystems are then bridged with appropriate software, hardware, or other smaller subsystems. This is pretty straight forward when it comes to clearly defined preexisting complex systems. Brand new complex systems introduce the additional challenges of determining what subsystems need to be included in the whole integrated final product, how they interact, and how

they accomplish the overall function(s). The additional requirement layer of system intelligence does not help solve the matter, but adds several degrees of freedom to the design process.

Therefore, for both complex system categories, integration of subsystems is not the issue as much as whether this collection of integrated subsystem accomplish the required overall system objectives. The literature describes how this can be determined using methods from Model based engineering and such, but almost all methods boil down to: design representation, modeling and simulation.

The implications of the intelligence requirement on complex systems are briefly discussed next. This will be followed with a detail break down of the the three major building blocks of the design process (representation, modeling and simulation), and how these three building blocks affect the final outcome of the process.

1.6 Implication of the Intelligence Requirement on Complex Systems

The system intelligence requirement complicates the issue even more. Intelligence is a loosely defined requirement. In fact, autonomous behavior does not necessarily mean intelligence. Also, a system can achieve the intelligence requirement only through the integration, interaction and interdependency of several components and/or approaches and/or algorithms. As an example, the presence of high speed computational resources (such as advanced computers or servers) on a ship does not necessarily lead to an intelligent naval vessel design. Neither does the deployment of sophisticated algorithms on these computational resources without an adequate communication network. It is the interaction between the integrated complex system's components (the ship's components in this case) that achieve the intelligence requirement.

Hence, an intelligent system cannot be evaluated, and not even ensured, simply by making sure its components function properly as designed. In fact, there is no guarantee that these components or subsystems will not work against each other

bringing the system down. And since there are nearly unlimited possibilities of how such a system can be put together (resulting in a spectrum of designs), then exploring the full design space is a tedious task. Therefore developing a systematic approach to the process of design space exploration is needed for intelligent complex systems.

Although systems engineering literature is rich with developmental task description, a review of the current state of the art for "intelligent" systems design reveals an extreme lack of specific design methodologies [64]. This further plagued by the fact that many systems that have the intelligence requirement are large scale systems.

1.7 Intelligent Systems Design Practices

Intelligent systems have been designed since computing resources became more available, together with the advent of techniques such as fuzzy logic, neural networks, expert systems and genetic algorithms. However, no standard process exists by which intelligent systems can be designed. Every artificial intelligence algorithm used in intelligent systems has its own ad hoc design process, and almost always requires a physical system to be designed. For example in your network type artificial intelligence is designed after the physical system is designed so that data from the physical system can be used in training a neural network. Neural network training is well understood process in the literature; design of the physical system that the neural network controls is also a standard process; but the designing both the neural network and a physical system is seldom addressed in any literature.

Figure 6 shows the general approach of control architecture design, including intelligent architectures. The physical system is designed first using standard system of systems design processes. System design starts with problem decomposition and requirements definition, after which the conceptual design loop starts. Once the physical system concept has been reached, the conceptual design is finalized. The process repeats for the preliminary design and the detailed design faces until a final

physical system design is reached.

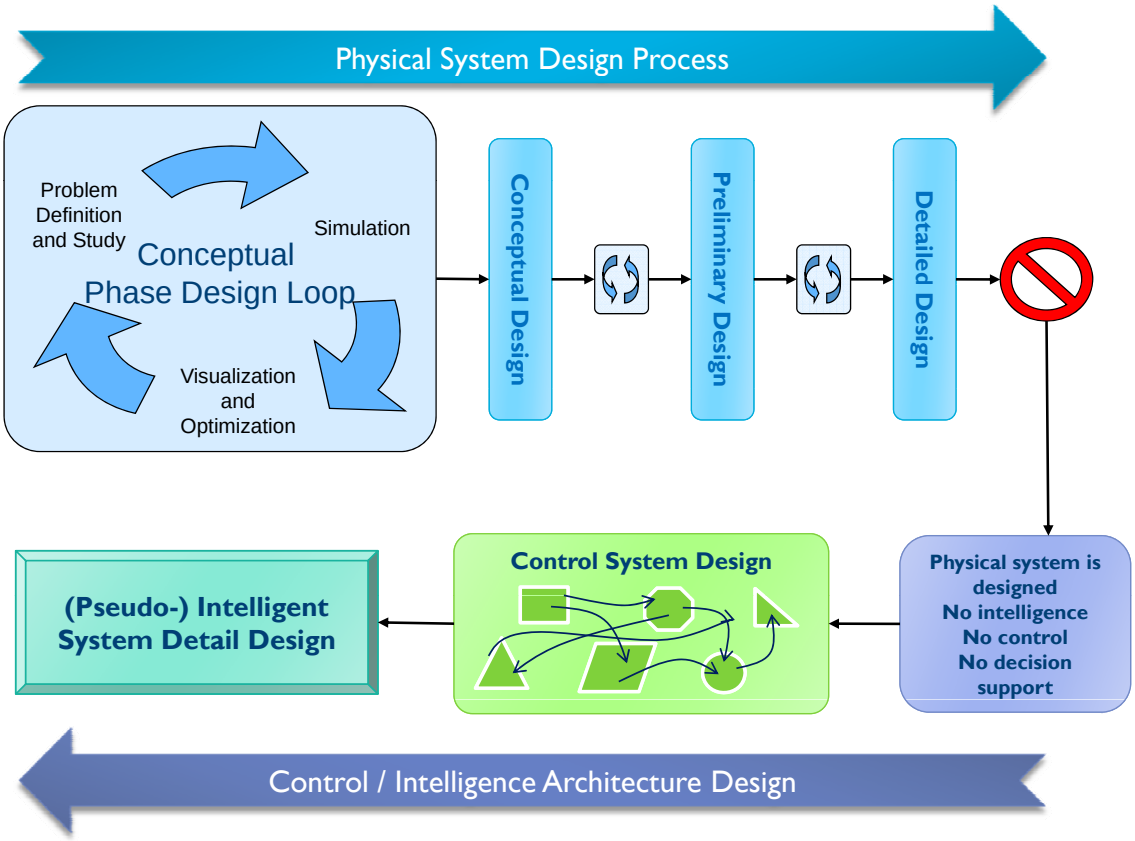


Figure 6: Intelligent Systems Design Process

The control architecture design starts after the physical system design is over. In fact, in many cases the control architecture is addressed even after the physical system is already in its manufacturing phase. Controller of specific subsystem is specifically designed to control this subsystems assuming no interaction with other subsystems. The set of controllers for all subsystems are then "stitched" together in one control architecture. The configuration of the control architecture (the components, the interconnection structure, and their functionality of each component) is such that it satisfies the controllers designed earlier. control methods within the controller where already chosen prior to the controller design.

This method works well for large-scale autonomous systems. Automatic controllers are designed for the subsystems as mentioned, then connected. In some cases

intermediate controllers are added to ensure or improved stability or performance. Minimizing interactions and interdependencies between subsystems helps design better controllers. Also, such systems are well understood, and have been designed numerous times.

However, the systems under consideration are required to be intelligent systems. This means that intelligence is an inherent property of the required systems. The current state of the art in systems design does not allow for intelligence architectures to be taken into consideration during the conceptual design phase. It does not make sense to design an intelligent system by designing the physical system first, followed by designing the intelligence as an add-on. There is no guarantee that the system can function optimally with its will intelligence potential if the intelligence is an add-on.

Control architecture design has three main elements (which will be discussed in more detail later in this dissertation): control configuration, control method and controller, as shown in Figure 7. The control configuration describes the components in the architecture, their functionality and their connectivity. The control method describes the mathematical algorithms used within these components. The controller is the actual parameters that are used by the control method, and the numerical implementation of it. Most of time and the resources are spent in the controller design, not in the configuration choice or the control method selection.

1.7.1 Modeling

A model is a simplified representation of a real physical system that may or may not be implemented. It facilitates the understanding of the behavior of the system, its properties and its boundaries. In some cases, models can be thought of as mathematical abstractions of the physical system (e.g. models of dynamic systems), while in other cases, models are a qualitative description of a system's behavior (e.g. creation and evolution models, business models). The scope of this research falls under

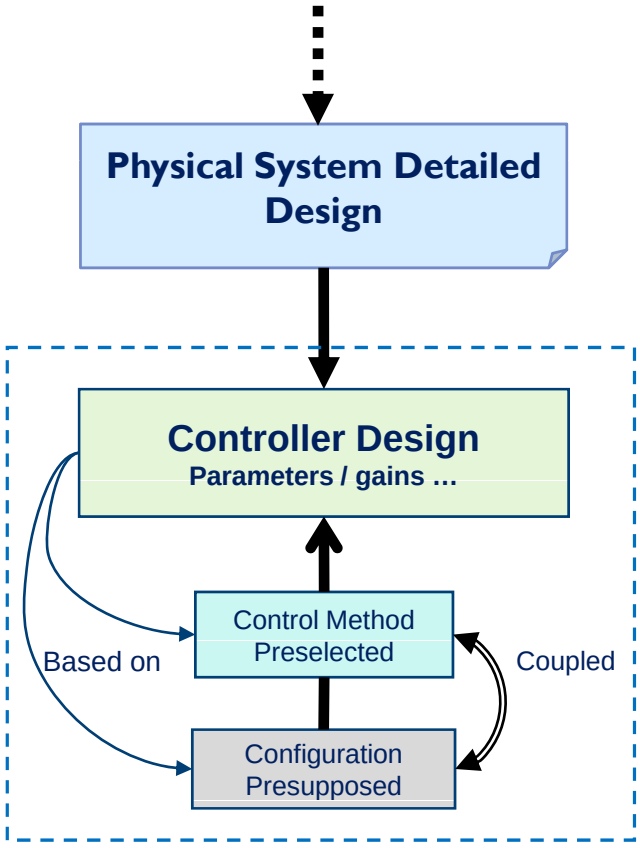


Figure 7: Current Control Architecture Design Efforts

mathematical models of intelligent complex systems such that the systems are always represented by a set of mathematical / logical relationships among its components.

Modeling is an important enabler in complex systems design throughout all levels of the design cycle. Hence, the way the designer builds, validates, and uses a complex system's model affects the final design outcome. Models are not created the same way, there are no standard processes for their validation. Even when assuming one model for a particular system, various designers might utilize them differently. This leads to the immediate conclusion that solely relying on a specific model to defend a certain design decision might be unwise. Engineering sense, experience, professional practices, and procedures are among other factors that can help void such shortcomings.

Yet no designer undermines the value of a well built and adequately employed

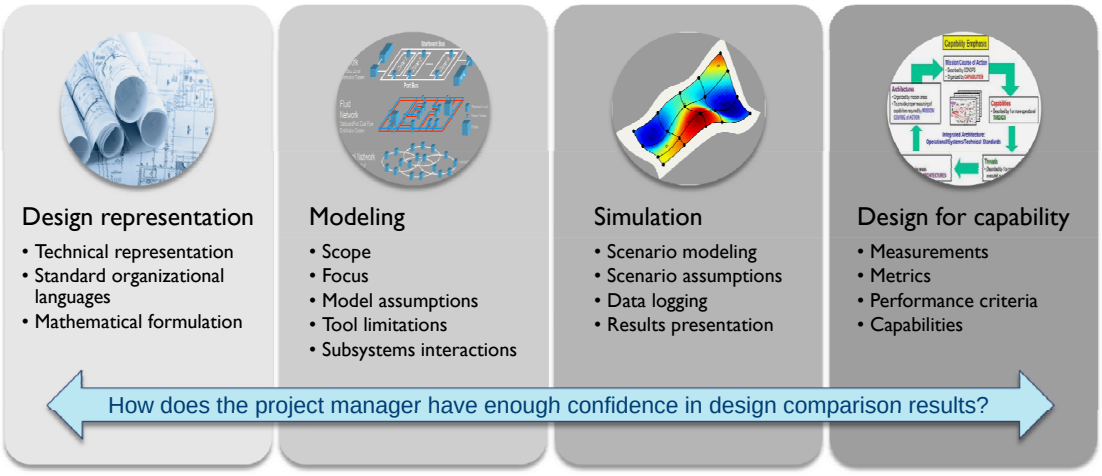


Figure 8: Challenges of Intelligent Systems Design

model in reaching a final design product. Issues related to the model scope, focus, assumptions, simplifications, limitation of tools and scalability need not to be overlooked. The breakdown of these issues is not a standard one; several other points of view may exist. For example, model focus and scope may be merged while tool limitations can be broken further into subcategories. The current research will adopt the afore mentioned issues' classification because it allows for better understanding of future introduced concepts. Many designers instinctively consider and assess these issues when comparing results from dissimilar models, but not necessarily in a systematic process. To better explain these issues, an example of an intelligent complex system of systems, namely an intelligent electric ship model, will be used as a clarification.

1.7.1.1 Model Scope

Model scope is the extents of the model. It is what a model of the physical system bounded by the problem statement under consideration. The model scope simply refers to "what the designer needs to model so as to reach a design decision".

For example, a model can be built to study the behavior and performance of the

electric network on a ship. The DC electric network on a naval vessel can be divided into zones such that each zone is supplied by two buses. Some designers might design each zone independently while others prefer designing the whole system. This decision affects the model scope, will the model represent a single zone, or will it model the behavior of all zones, or both? As another example, a model that represents the electric power flow on the electric network is substantially dissimilar to another that models the DC (or AC) dynamics.

1.7.1.2 Model Focus

Model focus is slightly different from model scope. While the scope identifies "what is being modeled and within which limits", model focus refers to "what output(s) needs to be considered". Note that although scope and focus are two distinct issues, there is no clear cut on where the scope concept ends and the focus begins. Model focus is more concerned with the final requirements from the model.

Building on the previous electric ship network example, a model can be focused on tracing the power consumption across the ship' different subsystems. Another model may focus on exploring the transitional dynamic behavior and stability of the electric network These two model focus approaches result in two very different models. In other cases, a single model can focus on and cover different design team requirements. Also, it might be the case that the design team decides on several points which the model focuses on. This is a part of the design team's tasks: "what to model and within which limits, and type of results needed".

Note that model scope and model focus greatly affect other issues such as tool limitations and model scalability.

Also, note that depending on the design team's decision regarding model focus and scope (and other issues as will be shown later), a final model (or several models) are built. If several design teams are competing, with each producing a final model

(or several models) resulting in different conceptual designs, the project manager has to decide on which design is more suitable (optimal or "wins"). And here is where the main problem lies: the project manager chooses a design based on comparing the results from different models, not fully taking into consideration that the models have different focuses, scopes,...etc. Results are simply not "fully comparable".

1.7.1.3 Modeling Assumptions and Simplifications

As an example, a model representing the electric network on a ship might have the simplification of steady state dynamics rather than modeling the transition dynamics. A certain design chosen based on a simple model might not be adequate for off design ranges at which the simplification assumptions do not hold.

1.7.1.4 Interactions Among Subsystems

Subsystem interdependency contributes to the complexity and partial unpredictability of emergent behavior. When it comes to modeling, interdependency is the highly coupled relations set (e.g. equations, logical rules, ...etc.) that governs the behavior of the system. Interdependency can be among major subsystems or in between components in the same subsystem. The degree of coupling between the relations determines the degree of interdependence across various levels. Less coupled relation sets can be grouped in different subsystems or sub-models to simplify the modeling, simulation, and validation tasks. On the other hand, if highly coupled relations are overlooked (or ignored), the model can produce erroneous or misleading results. A decision during the modeling task regarding the level of interdependency modeling greatly affects the validity and usability of these final results.

As an example, assume that the electric network discussed earlier contains a substantial number of heat producing loads, such that their performance is temperature dependent. In addition, assume a cooling network layer separate from electric network layer. The electric loads are cooled by the cooling network, and the pumps in

the cooling network are run by power delivered over the electric network. Hence, the electric and cooling networks are highly interdependent, such that a major fault in either will result in a cascading failure in the other. Some design teams or modeling teams might choose to approximate this interdependency with proper functions, with other models, with historic data, or by some other means, while some other teams might choose to neglect this interdependency all together. Assumptions and simplifications clear documentation is of ultimate importance to the project manager because it helps compare different designs.

1.7.1.5 Tool Limitations

Modeling tools are diverse and span a wide range of applications and research domains. The category of modeling tools more relevant to this section is software and programming languages. Some tools are better than others in modeling certain complex systems, and considerably vary in their level of sophistication, detail, accuracy, ease of use, and export/import capabilities. The choice of the modeling tool is crucial in accomplishing the model scope and focus objectives and in maintaining an appropriate level of detail so as to produce valid results. Limitations of certain tools may result in areas in the design space not being explored, or off design points not properly investigated.

Tool limitations also cover technical capabilities offered by the tool, such as flexibility of input/output, state access, integration, and compatibility with other tools, stand alone execution, batch execution, flexibility and learning curve, library support among other factors. Yet most of these limitations have less impact on the decision made by the design team/manager.

1.7.2 Simulation

Simulation is the manipulation of the model so as to generate data describing how the physical system behaves in reality. Simulation (ideally) compresses real time.

Simulation enables designers study the behavior, performance, limitations, emergent behavior, and interdependency of a particular system (and its embedded subsystems) without having to build it. Running a simulation requires three major components: a model, inputs, and outputs. Inputs are better referred to as scenarios while outputs are usually a subset of the data pool within the simulation. Issues with modeling were discussed earlier, so this section is concerned with the other two.

1.7.2.1 Scenario Modeling

The choice of the inputs to the model and how these inputs vary are crucial factors in studying the behavior of the system. Inputs are better referred to as scenarios because they comprise, but not limited to, time dependent input data, system's parameters settings, time history of the system's states, and in some cases, external disturbances affecting the system.

Guidelines for scenario modeling are found either explicitly or implicitly within the complex system's definition and requirements documents. Yet these guidelines are usually very broad, and allow for wide variations among different modelers and system designers. In addition, some designers, according to their discretion, might choose to emphasize specific scenarios while ignoring others. This leads to the fact that comparison of simulation results from different teams or based on different models is a partially inconsistent comparison.

1.7.2.2 Scenario Assumptions

The usefulness of the level of detail of scenarios greatly depends on the level of sophistication of the models used in simulation. An elaborate scenario render itself an overkill if the model used in design is over simplified. Along the other extreme, a scenario that is over simplified might not push the model enough to produce reliable design data. For example, a scenario that only takes into consideration specific discrete system states might be useless in studying the effects of transition between

these states on the system. This becomes a profound and complex design problem when transitional states might lead to system instability. AC Electric networks are a common example.

Again, similar to scenario modeling, scenario assumptions are generally not consistent among teams or competing designs and may result in unfair comparisons or less informed decisions.

1.7.2.3 Data Handling and Logging

As a common example, if a model is received in the form of a compiled library or stand alone application, with access to only the inputs and outputs specified by the model builder, there is no way to track particular states within the system because they count as internal variables in a compiled code. Object based modeling tools offer a better choice when it comes to exploring the internal states of a model. Yet, these capabilities come with a computational overhead.

Another issue with data logging is the amount of data generated. Even with having the capability of recording large amounts of data, extracting useful information from this data set is a challenge. In addition, some computational setups cannot handle large amounts of data. Also, if several models are run simultaneously, a proper form of data sharing and synchronization among those models need to be put in place.

Both data handling and data logging offer a challenge to the design team in comparing different designs.

1.7.2.4 Results Presentation

Typically, design teams will present their best result sets. Malpractice is not assumed in this context, but usually weaknesses in a certain design are not on top of the priority list of the items to be presented. It is usually up to the project manager or reviewer to point those out, either by requesting more information or by inference from results presented. In other cases, by requesting a more rigorous scenario modeling resulting

in a wider scope of results.

1.7.3 Design for Capability

A *Measurement* is a physical quantity directly obtained from a sensor, or indirectly from a sub model, that represents a physical state in the complex system. Temperature, rotational speed, pressure ...etc. are common quantities. A particular component might be designed to withstand a certain physical quantity (or more) but seldom is a subsystem designed based on these lower level states.

Measurements aggregate to form *Metrics*, either through mathematical relationships, lookup tables, historic data or some other means. Metrics calculation brings meaning to a group of measurements from a systems engineering and design perspective. For example, an aircraft velocity vector may be calculated from a combination of GPS data, inertial navigation system data, aerodynamic data and the aircraft dynamic computer model's results, resulting in the metric of aircraft true air speed, or relative wind. Although metrics are important in complex systems design process, most are not explicitly specified in the top level system requirements. In the conceptual design phase, very little is known about their specifics.

On the other hand, complex system's *Performance* is usually part of the top level design requirements. For instance, a performance requirement on a military aircraft might be simultaneously engaged in targets in a specified flight condition, with an x probability of surviving the engagement. In this case, performance is not a measured physical quantity, nor is it a metric aggregation of measurements. It is a collection of metrics that describe how the system's top level requirements are accomplished. Engineering systems are typically designed based on performance requirements.

Capability is the ability to undergo a specific action, course of action, or to be affected by a certain treatment. The required increase in capability for intelligent

systems results in larger more complex systems, with more components, interconnections, and modes of operation [9]. Capability-based design and acquisition is a fairly new branch in systems engineering. The U.S. Armed Forces doctrine states that it is basing its military goals on system capability [5]. The U.S. Department of Defense as well is pushing forward more capable systems in its vision for 2020 (see [126] and [56]), followed by publishing the DoDAF standard [40].

Intelligent complex systems are more concerned with capabilities and performance criteria rather than specific metrics. For example, a military vehicle may be designed for lethality or survivability (capabilities), and the amount of fire power it can deliver (performance). A naval vessel might be required to support littoral navy operations, but how can a designer assess such a capability? And even if a standard definition is set and agreed upon, what does this capability mean in terms of specific system design parameters?

This imposes a challenge to researchers since capability metrics are defined arbitrarily. Hence, depending on one's mathematical definition of a certain capability, a system can satisfy this capability's design requirement and be accepted while another might not. Changing the mathematical definition of such capability can result in changing this result. Note that there is no sharp distinction between capability and performance, a gray region exists.

1.8 Software Engineering - Control / Decision Making Architecture Relationship

In general, software projects have a high risk, and in a way, unpredictable outcomes [44]. A NIST (National Institute of Standards and Technology) study in 2002 that found software errors cost the U.S. economy $59.5 billion annually [136]. Also, the report states that 80% of the money spent on software development goes into correcting defects. Software is provided to customers with a high level of errors [136].

Control architectures have long been based on software platforms. For example

the software enabled control developed on an open control platform in [120] is applied to unmanned air vehicles. A hierarchical control structure approach is used, such that mission planning and situation awareness are on the highest level while flight control is at the lowest level. The control architecture was implemented on an actual hardware platform as described in [74].

The fact that control and decision making of complex systems relies heavily on software architectures complicates the matter even more. Users of these complex systems cannot tolerate such error levels. For instance, a user may accept an update to his personal computer's operating system because of a defect in the API architecture. But the navy will definitely not accept a defect in it. damage control software-side architecture on a multi-billion dollar naval vessel. The higher the investment and the risk to human life, in addition to the increased complexity of a system, make users less tolerable to errors. Hence, standard software engineering design practices cannot be directly used in the design of complex systems. Extra steps have to be taken to ensure an adequate optimal design.

1.9 Motivation

The bottom line: if there are multiple ways of representing a certain design, if models differ widely, if simulation is an art in itself, if there are no standard definitions of top level capabilities, how can the project manager have enough confidence in a certain design as compared to another? This problem is amplified when considering intelligent large scale complex systems.

The reasons for this can be summed as follows:

1. Design of the control and decision making architecture always start close to the end of the physical system's design cycle. Control and decision making architectures are designed almost always after the detailed physical system‘s design is finalized and all the parameters are set. Sometime, this takes place

between the completion of building the physical system and the beginning of the testing and accreditation (if any). In fact, the only factor that keeps system owners (such as the navy in the ship example) from conducting full scale system wide tests is that the control and decision making architecture is at the heart of an intelligent complex system. Therefore, the question is: How can the a subsystem vital to the existence of it parent system get the lower priority in its design cycle, receiving a follower status rather than the influence of a design driver?

2. Control systems design steps are not standardized

3. No standard process exists for the design of distributed intelligent complex systems. However, there exist several complex system design processes and methodologies, but none is specific to the complex system type. Decision making architectures and control systems have no standardized design processes. Only experience based or general rules of thumb define the design procedures of such systems.

4. Controllers are not classified in a way that enables rigorous exploration of the control architecture design space.

5. Control architectures have no standard representation framework to enable consistent comparison of different designs.

The solution to this problem is simple: Consider the intelligent system control architecture concurrently with the physical system architecture during the conceptual design phase.

The issues discussed earlier with modeling simulation capability and design the presentation are essential to intelligent systems. However, these issues tend to hinder many of the intelligent design processes that are being developed. The lack of standardization, the multiple definitions of the same concept, the multitude of standards

applicable to different disciplines, the diversity of standards regarding modeling practices in terms of scope focus and assumptions, all help drive intelligent system design processes towards an ad hoc path.

In addition to the above issues, intelligent architectures are not addressed until the controlled physical system is completely designed. The lack of a standard process and a standard representation framework leads to the fact that many designers produce various architectures based on different physical systems with representations relying on incompatible standards, producing dissimilar results.

Hence the need for a standard process for intelligent control architectures design is obvious. If this standard process is included in the physical system's conceptual design phase, adopting common design tools, and extending current design paradigms to concurrently address intelligence capabilities as little as physical system requirements, then a more consistent intelligent system can be built.

There will be two main thrusts included in this research. The first proposes a hybrid design process that addresses the concurrent conceptual physical system design and the control architecture design. The second thrust relies on the development of a control architecture metamodel, enabling the standardization of representation and analysis among various architectures.

1.10 Need for Intelligent Large-Scale Systems

As engineering systems increase in scale and complexity, requiring the achievement of multiple, and in many cases contradicting, capabilities, traditional design processes fall short.

Large-scale systems as the number of integrated subsystems rises,

There are many engineering fields in need for intelligent systems design.

1.10.1 The Smart Grid

The electric grid has often been referred to as the greatest, most complex machine ever built, consisting of a host of component types, many subnetworks, multiple subsystems, all operating interdependently. In actuality, the electric grid is (or used to be in the near past) mostly built of mechanical systems and components, save some modest use of sensors, electronic communication with almost no electronic control. Within the past 25 years, many utilities companies have started modernizing their electric grids with the use of intelligent sensors, communications and computational capabilities [57].

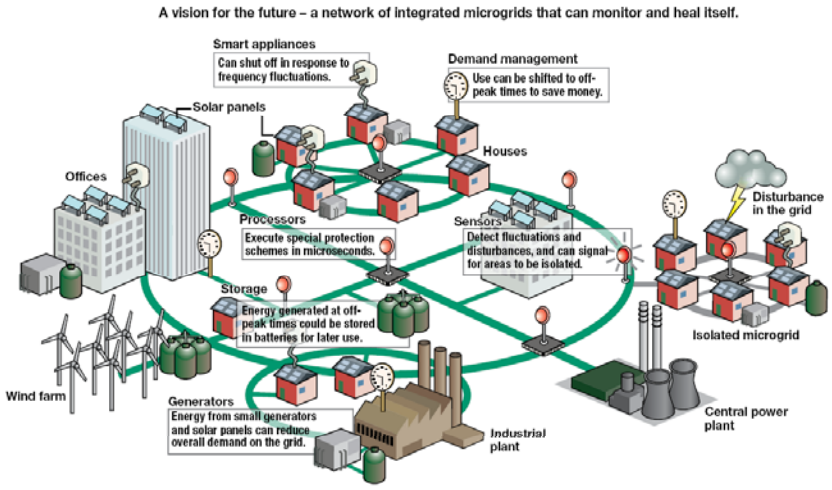


Figure 9: The Smart Grid [89]

The collective use of intelligent sensors, together with computational equipment running intelligent algorithms, in addition to advanced two-way communications on multiple levels, are referred to as the smart grid. One of the main defining features of the smart grid is the ability to use subsystem (or micro grid) feedback to continually adjust grid parameters for optimal performance through sending back control commands to different grid equipment. Other important smart grid defining features include: the ability to integrate, optimally manage distributed energy sources (traditional as well as renewable generation and storage), improvement of the current

electricity infrastructure with regards to smart metering, and improved automation at both control centers and substations levels [6].

The vision for the future of the smart grid is presented in a report by the U.S. Department of Energy. It envisions a smart grid that: self-heals, motivates and empowers consumers, resists attacks, provides power quality suitable for modern needs, accommodates all storage and generation options, enables the integration of markets, and operates efficiently while optimizing assets [96].

However, these characteristics require a control and automation architecture, that does not only manage the electric network on the grid level, but also facilitates the distributed control of various member microgrids. Electric utilities are going to have to expand monitoring and control throughout their distribution grids all the way to the customers side using smart meters [37]. The U.S. Department of Energy vision clearly states [96]:

"These seven characteristics describe a vision for the Modern Grid that is generally more resilient and distributed, more intelligent, more controllable and better protected than todays grid."

The key enabler for the smart grid is the intelligence, not only on a central level, but in the distributed sense. A standard control architecture for the smart grid (or even a process do design one) is not yet available. However, there are initiatives for the intelligent grid such as the *Integrated Energy and Communications Systems Architecture* of the Electric Power Research Institute (EPRI), more commonly known as the *Intelligrid*. EPRI issued a technical report [68] documenting the methods utility companies use to integrate emerging technologies in the smart grid. Open standards is emphasized as a way of integration, in addition to systems engineering methods approach. Use cases are extensively employed in the analyses methods presented, in which the role of an intelligent control architecture is evident. Hence, the intelligent control architecture is an inherent self defining feature of the smart grid.

1.10.2 Next Generation Naval Vessels

The primary aim of the design of a ship board electric power system is survivability and continuity (reliability) of the electrical power supply. This task has to be accomplished in almost all scenarios a naval vessel faces. The vision for the next generation naval vessels is to: *"Produce affordable power solutions for future surface combatants, submarines, expeditionary warfare ships, combat logistic ships, maritime prepositioning force ships, and support vessels."*[46]

The Next Generation Integrated Power System (NGIPS) is an enterprise approach to develop and provide smaller, more affordable, and simpler power systems for all Navy ships with more capabilities. NGIPS is both a business and technical approach to define standards and interfaces and efficiently use installed electric power. The NGIPS technical approach utilizes common elements such as Zonal Electrical Distribution Systems (ZEDS), power conversion modules, and electric power control modules as enablers along an evolutionary development path [49].

Two of the major electric power system's architecture components are the Zonal Electric Distribution Subsystem (ZEDS) and the Power CONtrol subsystem (PCON) [51], [48]. The ZEDS may contain a controllable bus transfer module which receives commands from the control architecture to switch power supply to certain loads. It may also contain an automatic bus control module which has the same function but acts autonomously using its embedded algorithms without commands from the control architecture. Hence, the ZEDS requires a control architecture of some kind, either embedded within the zone or an over arching architecture across all zones. The PCON counts as the over arching control architecture. It consists of all the software necessary to coordinate the behavior of other modules, such that all the naval vessel capabilities are achieved during normal operation, maintaining quality of service during faults [50] and ensure survival during system wide failures [47].

However, the control architecture function extends beyond the power system. It

also encompasses other ship systems, such as the cooling system, combat system, fire suppression system, environmental system, ... etc.. Moreover, the modern view of naval vessels is that each is a player in what is referred to as "distributed networked forces" [31], i.e. a team of different military assets set on accomplishing one objective. The control architecture has to communicate and coordinate with other control architectures of other players, and execute its part of the overall system of systems plan. Therefore, the next generation naval vessel is another type of a large scale system requiring an intelligent control architecture.

CHAPTER II

FUNDAMENTAL CONCEPTS

2.1 Control System Principles

Autonomous systems function with minimal external inputs. These systems are set for a desired final performance and should reach this performance setting (or close to it) within a specified time limit. A generic autonomous control system is shown in Figure 10. The free running uncontrolled system (referred to as the plant) is the physical system itself. For example, a plant can represent a simple DC motor, or it can represent a naval ship. What is important is that this plant has to produce a flow of energy (or mass), money, information or any other desirable quantity. Almost all plants have a nonlinear nature to their behavior. Yet, in many cases, this nonlinear behavior can be overlooked if the designers focus on an operation window rather than a full spectrum of operation.

The controller is the part that regulates the flow of energy (or mass), money, information or any other desirable quantity [27]. A controller can be a physical entity, such as a circuit of resistors and capacitors that regulate the voltage to a DC motor, or it can be a non physical entity, such as a software that controls certain

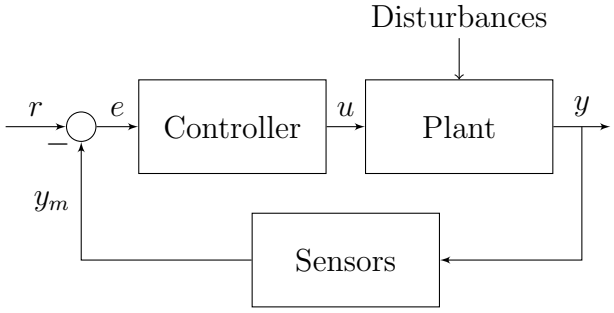


Figure 10: Generic Autonomous System

subsystems on a naval ship. A controller requires an external plant set point (i.e. a required final state or performance of the plant) and is responsible for driving the plant to this final state. Controllers vary in structure and complexity. Controller design is based in the area of *Controls Theory*. Extensive research exists in this area.

Some controllers receive signals representing the desired plant operation set point *r(t)* and manipulate this signal producing a *control input* $u(t)$ to the plant. This is referred to as *open loop control* because signals (or states) flow in one direction from the user input to the controller to the plant. In many cases, open loop control is not enough to adequately drive a plant to a certain required final state, or even to stabilize a plant. *Closed loop control* has to be used, instead. In closed loop control several output states ($y(t)$) of the plant are fed back, pass through a sensor model to produce different outputs $y_m(t)$, which in turn is used to calculate the discrepancy between the required final states and the feedback states ($e(t)$). Closed loop control is a necessary in controlling inherently unstable systems such as missiles.

The first objective of a control system is to achieve and maintain closed loop stability. The second objective, if any, is to achieve the required final states and performance. Controllers, similar to plants, can either be linear or non linear. The former set of controllers has well established and well tested design and analysis methods, usually rely on pole / zero placement techniques. The latter, however, is more difficult to achieve, specially if the plant is also nonlinear. Usually design methods for nonlinear plants / controllers are based on the Aleksandr Lyapunov theory. Controller design complexity can range between the simplest PID control to very sophisticated software optimized control techniques.

2.2 Control Architecture Configuration

Simply stated, the *Control Configuration* is the arrangement, connectivity and functionality of all control architecture components. Control components include subcontrollers, actuators, sensors and communication links. The arrangement of components refers to the logical links between these components. In some cases, logical links coincide with physical (spatial) links, but not necessarily. The components connectivity identifies the data, signals, or information exchanged between components. The functionality of each component needs to be defined for a proper control architecture configuration specification. Functionality is what task the component accomplishes, but not necessarily how it does it.

2.2.1 Centralized Control

Centralized control refers to the presence on one controller that controls all the plant independent variables by processing all the sensor signals and issuing all the actuator commands. Centralized control is common in small scale systems and linear systems. Centralized control has several advantages. It is simple to design if the system governing equations are known, or if a system simulation engine exists. All interactions between independent system variables are taken into consideration when designing the controller. Most of the centralized controller design approaches for linear systems are well understood and documented in the literature. However, as the size of the system gets larger and as the level of interdependence between system states increase, centralized controllers become harder to design. In addition, no matter how reliable a centralized controller is, it remains as a single vulnerability for complete system failure, since the failure of the controller almost always leads to the loss of system capability.

2.2.2 Hierarchical Control

Hierarchical control configuration, as the name implies, relies on a set of independent controllers arranged in a hierarchy. This is possible if the physical system of systems can be characterized by a finite set of subsystems with clear and unchanging interface between them. Each controller can be separately designed and optimized [72], hence reducing the complexity of controller design process. Optimizing the performance of the whole system requires all controllers to be designed and interactions modeled accurately. Several issues need to be addressed in hierarchical control design, such as data transmission between sub-controllers [65], data latency, and data consistency among others. An example of hierarchical control is shown in Figure 11.

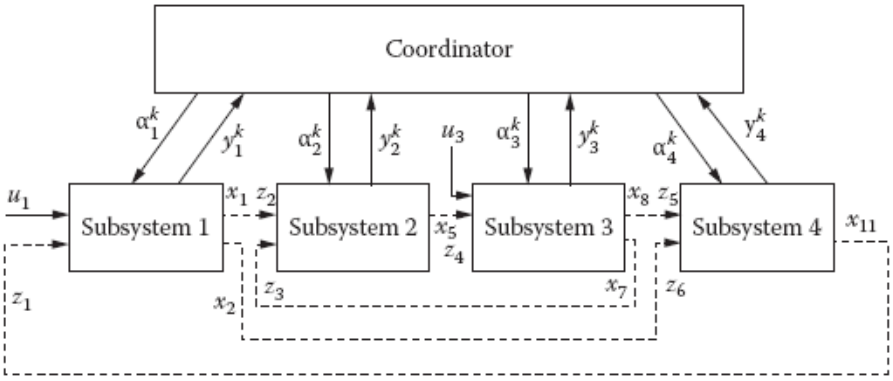


Figure 11: Hierarchical Control Configuration [72]

2.2.3 Decentralized Control

Decentralized control configurations are possible when the collection of all input and output variables in a system can be grouped in distinct subsets. A local controller performs local control action on each subset of variables, so as to control its local domain. In general, the less the coupling (shared inputs/outputs) between local sub-controller, the easier it is to design the system. Instability regions can be avoided easier since they usually originate from couple variables. Decentralized control is well suited for large scale systems and system of systems. There are several texts that

tackle decentralized control, such as [129] which has a more rigorous mathematical approach. The paper [14] presents an overview of decentralized control, decentralization (as related to information structure), decomposition of subsystems (such as disjoint, overlapping or hierarchical subsystems), and robustness. Communication is a key concept in decentralized control since stability and performance of the system rely on the communication channels carrying the overlapping variables [13]. Regulatory controllers may be placed between subsystems' controllers to ensure stability (see Figure 12). Decentralized control, as treated in this dissertation, requires that no central sub-controller communicates with all other sub-controllers, such that all sub-controllers are treated on the same level. However, inter-communication between controllers is allowed.

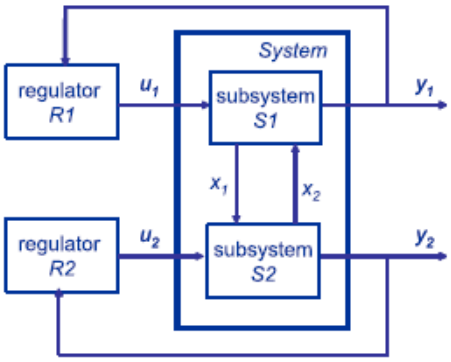


Figure 12: Decentralized Control Configuration [118]

2.2.4 Distributed Control

Distributed control configuration is the intermediate concept between hierarchical, centralized, and decentralized control configurations. The controller components are distributed throughout the system, either functionally or spatially, but not embedded in the same module. Although counter intuitive, centralized control can be of a distributed control type control. An example is given in [29], in which the control architecture is distributed among four layers (digital, analog, network communication,

and device interface), but all layers are controlling the same process. The paper [18] shows how distributed control can be of a hierarchical type. It outlines a hierarchical control architecture for improving sensor management for distributed military surveillance operations. Distributed control suffers from the same issues as hierarchical and decentralized control, namely the profound effect of communication functions on the overall control architecture performance.

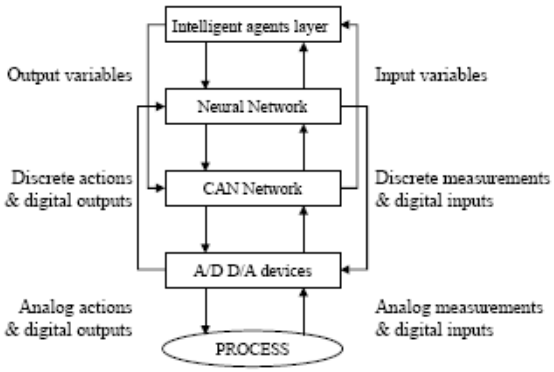


Figure 13: Distributed Control Configuration [29]

2.3 Control Architecture Methods

Control Methods refer to the the type of mathematical algorithms used to process the measured signals from sensors, process the received human input (or other higher level controller input), evaluate them, and compute a control action to be passed to actuators or to other controllers. There exists no comprehensive list of control methods. However, some of the more common methods in industry are presented next. It is important to note that, sometimes, there is no clear distinction between different methods. For instance, cooperative control, network control, distributed control and multi-agent control are related and overlapping control techniques [125]. In addition, control methods and control configurations are highly correlated in the literature. Distributed control configuration is sometimes treated as a control method. Model Predictive Control has distributed and centralized versions; among many examples.

However, it is the intention of this dissertation to introduce the conceptual difference between the control architecture configurations and control architecture methods.

2.3.1 Classical and Gain Control Methods

The simplest control method is the multiplication of a feedback signal by a constant number (a gain). Sometimes, this is extended to the derivative and integral of the feedback signal, hence Proportional Integral Differential (PID) control. Other methods, such as adaptive control, changes the gains depending on the range of operation of the control variables or system states. Linear Quadratic Control (LQR) is similar to proportional control in structure but differ in how the gain is calculated. The gain is computed to optimize a linear quadratic equation which has depends on the system states. All control configurations can include this type of control method. With classical methods, no intelligence is involved.

2.3.2 Model Predictive Control

Model predictive controls aims at optimizing forecasts of process or system behavior over the manipulatable inputs [107]. This is accomplished by including a software model of the system in the controller. The controller, using the software system model, attempts at changing the inputs to produce desired or optimal system response. Once a set of inputs are calculated, they are applied to the real system. An accurate, but not necessarily perfect, model is essential for model predictive control [108]. Similar to classical control, model predictive control can be applied within all control configurations, making it one of the more commonly used control methods. Decentralized control, distributed control, hierarchical control for coordination and for multilayer systems, and coordinated (cooperative) control are all addressed in [118], in a model predictive control method context.

2.3.3 Cooperative Control

Cooperative control method assumes that a system can be decomposed into a set of subsystems or agents with a common goal, such that the agents all work together "in cooperation" to achieve this goal. Cooperative control can be applied to multi-agent sensor networks [141], in which several sensors track targets and share information, as well as in [34]. Other applications include speed control and damage avoidance for several agents [58], distributed cooperating agents search a range with obstacle avoidance [106], vehicle formations [102] and [109] among other applications. Stability of cooperative control architectures has also been a subject of research, such as in [98]. Cooperative control can be categorized, as presented by [110], into leader-follower, behavioral, virtual structure/leader, and others.

2.3.4 Network Control

A network control system is a control system that includes network path on one or more of its feedback or feedforward paths. The main concerns with network control systems are latency and network bandwidth. Both can result in instability of the whole system.

2.4 System Stability

When it comes to complex system controls, there are two major trends that tend to undermine the well established principles of design and stability of controllers [134]. The first trend is the almost blind trust of control systems specially with open loop unstable physical systems. As the physical system becomes more complex with multitude of components and increasing interdependency between these components, the sense of stability becomes less obvious. Adding this to the fact that many physical subsystems around us are open loop unstable, this trend gets more profound and with potentially dire and dangerous consequences. The second trend is the loss of

the "control and decision making physical sense" to the emphasis on abstract mathematical constructs or models with many underlying assumptions. An example of such elaborate mathematical model for stability is given in [25], Given plant $g(s)$, compensator $k(s)$, and

$$gk(s) = \frac{n(s)}{d(s)}; \tag{1}$$

$$s(s) = \frac{1}{1 + gk(s)} \tag{2}$$

$$Z = \{z | n(z) = 0, Re(z) \geq 0\} \tag{3}$$

$$Z = \{p | d(p) = 0, Re(p) \geq 0\} \tag{4}$$

Then the system is stable if and only if

$$s(s) < \infty \quad \forall \ \mathrm{Re}(s) \geq 0 \tag{5}$$

$$s(z) = 1 \quad \forall \ z \in Z \tag{6}$$

$$s(p) = 0 \quad \forall \ p \in P \tag{7}$$

The above mathematical model is widely accepted by the control community, yet, it applies to the design of the control system of the SAAB Gripen JAS-39 airplane, which crashed on landing in 1989 in one of its first test flights. The point is, sometimes, elaborate mathematical models can cloak the inherent dangers of the open loop system, and designers are deceived by these models into thinking that their design is safe. As another example, the same theorem applies to the Chernobyl nuclear Plant disaster in 1986 [134].

The current stability tests, as borrowed from the rich and diverse control systems theory realm, require the physical system to be designed beforehand, hence all its parameters are available. This is a very restrictive assumption when it comes to

conceptual design and is one of the main reasons why controllers are not designed until the physical system is in its final design phases.

Three facts remain important when it comes to unstable systems [134]: 1)Open loop unstable systems are more difficult to control compared to open loop stable systems. For example, even with one of the simplest system configurations, a regular pendulum is far easier to stabilize and control than an inverted pendulum. 2)Controllers for unstable systems are operationally critical. 3)Closed loop systems with unstable components are only locally stable.

However, it is possible to identify stability issues, in a general sense, during the conceptual design phase. This is accomplished, not through detailed mathematical analysis of the system governing equations, but rather through inspecting system characteristics. The system's behavior revealed through simulation of a well chosen set of scenarios (refer to Section 2.5), together with a study of the system's operational and inherent constraints, characterize the stability properties of the system. Hence, such an approach will be adopted in this research.

2.5 Simulation and Modeling Concepts

Dynamic systems are systems that are governed by fixed rules (usually represented by ordinary or partial differential equations, or difference equations), and in which points (states) in the system propagate in time based on these fixed rules. Dynamic systems vary from a simple pendulum, to a fluid flow in a fluid network, to a fleet of ships at see, and even to a logistics battlefield management system. These fixed rules that govern a dynamic system are referred to as the system's set of equations. Solving these equations results in the time trajectories of the system's variables and parameters, referred to as the system states.

The set of equations that govern the system can have several types. The most

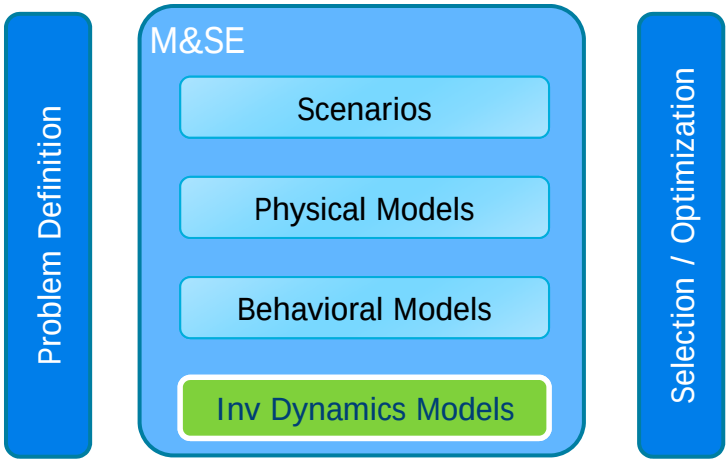


Figure 14: Model and Simulation Environment

common type for simple systems is a set of simultaneous continuous ordinary differential equations. There are many ways of solving a set of ordinary differential equations . Unfortunately, most systems exhibit varying degrees of nonlinear behavior, hence a set of nonlinear equations. Nonlinear differential equations have very limited analytical methods of solution. Hence, most complex systems' design methods rely on numerical techniques to solve the set of equations. The system's set of governing equations are not necessarily comprised of nonlinear equations, hybrid systems are systems having a nonlinear subset of equations in addition a subset of discrete difference equations. In the general case, there is no closed form solution to a hybrid system's state trajectories. Therefore, simulation-based engineering is an important enabler in complex systems design.

2.5.1 Simulation Based Engineering

"Simulation refers to the application of computational models to the study and prediction of physical events or the behavior of engineered systems" [100]. Simulation-based engineering together with advanced computational science are essential in presenting solutions to complex problems [19]. Analytical methods are, at many times, almost futile when it comes to large scale intelligent systems. There is no other way to design

an intelligence architecture but to use well designed and adequate software models in conjunction with realistic scenarios, hence simulation based engineering science. This discipline is strongly emerging as the main enabler in complex multidisciplinary systems design [100]. Simulation-based engineering combines traditional engineering disciplines, a such as electrical,mechanical, civil, chemical, aerospace, nuclear, biomedical, ... etc. with the computer science, mathematics. Simulation based engineering starts with studying the actual design problem definition and requirements (Figure 14). Simulation models are then created to appropriately represent the system to be designed in terms of accuracy and scale. Results from applying multiple scenarios to the software model are used to evaluate design options.

2.5.2 Model Driven Engineering

Software researchers and developers created many computer languages over the past years, raising the level of abstraction. For example, programming in C/C++ screens the programmer from machine code; using Unix shielded the users from programming the hardware directly. However, the abstraction is almost always within the computational domain (computer languages), not within the problem space (physical system to be designed) [119]. Model-driven engineering aims at moving the focus towards domain specific modeling approaches. The result is that the simulation model becomes the central design tool, test-bed, and evaluation platform. Many modeling tools are currently being used at an increasing rate, including UML, AADL, Simulink, Matrixx, LabVeiw among others. All such tools rely on the basic concept of modern driven engineering. Model based design is more accepted in the fields of software systems and control engineering [99], i.e the fields most relevant to this dissertation.

The *object Management Group* [60] develops integration standards for a wide range of technologies, one of which is model driven architectures. [10] presents the model driven architecture (Figure 15) as four layers. The focus of this research is the M_1

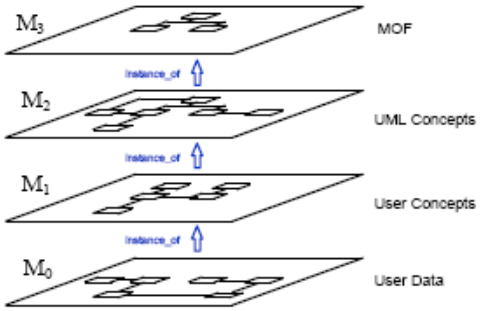


Figure 15: Traditional Modeling Infrastructure [10]

layer, i.e. the user concepts layer. This will be done through meta models.

2.5.3 Metamodels

Matamodeling is the analysis, construction, and development of the frames, rules, constraints, models, and theories applicable and useful for modeling a predefined class of problems. It is an essential foundation for model driven development. A metamodel is a precise definition of the constructs and rules needed for creating semantic models [2]. Metamodels heavily rely on abstraction of layers, components and even concepts.

Software computational modeling is one of the first steps in simulation based engineering design. There are several approaches to modeling dynamic systems, and not all are adequate for design. The software model should be able to satisfy several requirements including adequate accuracy, speed of execution, capturing required behavior to be studied, and in some cases scalability. The following modeling approaches represent different concepts in designing software models, and satisfy the afore mentioned requirements. Yet, they are not necessarily mutually exclusive. For example, a behavioral model could also be classified as a dynamic mode. Models used in the current presented research satisfy the following modeling concepts although on different levels of the design cycle.

2.5.4 Behavioral Models

Behavioral modeling is a modeling approach that results in a model which reproduces the real behavior of a system under consideration. There has to be a one to one relationship between the original system states and the simulated states. Usually, behavioral models are built for a specific degree of accuracy. For example, a model built to track the transient behavior on a fluid network will be very different from another that models the steady state operation. Behavior modeling is more concerned with the mapping of inputs (from scenarios) to outputs (states of the system) and less with how this mapping mathematically takes place. It is essential though to properly represent the physics to produce correct simulation results [150]. Hence, a behavior model can be as simple as a lookup table of such mappings or a neural network, or as complicated as solving a set or partial differential equations.

2.5.5 Forward Dynamic Models

Forward dynamics models are models that map generalized forces to system states. For example, in the field of kinetics, a forward dynamic model will take the forces and moments applied to a group of rigid bodies as inputs, and produce the motion states of individual members as outputs, hence the term forward dynamics. This is not limited to kinetic systems, but can be extended to all dynamic systems. Another way of viewing this approach is the actuators' system effects on the sensors' measurements.

2.5.6 Inverse Dynamics Models

Inverse dynamics is opposite to forward dynamics. It simply answers the questions of what generalized forces are needed to produce the required states. Building on the previous kinetics example, inverse dynamics asks for the torques and forces that result in a prescribed motion of a group of rigid bodies. Inverse dynamics play a more important role in conceptual design, more than the role of forward dynamic systems. The reasons are two fold. Behavioral models (essential to conceptual design) are

easier implemented through inverse dynamics models. The second reason is that during conceptual design, the known inputs are the required behavior states of the system, and the unknowns are the what external generalized forces are needed to reach this behavior, hence inverse dynamics.

2.5.7 Dissertation Modeling and Simulation Approach

As mentioned before, simulation-based engineering combines a multitude of engineering disciplines as well as computer and software modeling practices. Model driven engineering is one of such practices, that may use a meta-modeling approach to represent complex systems. Although originally developed for software, the same approach is intended to be used in the current research. Complex large scale physical systems, in conjuction with the proposed control architectures, are abstracted from a problem domain perspective, modeled, and their behavior is studied and evaluated.

CHAPTER III

LITERATURE REVIEW

3.1 Decentralized and Distributed Control Architectures

3.1.1 Centralized vs. Distributed Control

"Put all your eggs in one basket and then watch that basket." - Mark Twain

The research effort presented in [15] and [16] focuses on distributed control of aircraft propulsion systems. In the process, the research presents a comparison between the pros and cons of centralized and decentralized control architectures. Centralized control architectures are simpler to program since all the information and algorithms are in one place. This may result in simpler software configuration. In addition, the control architecture components, with the exception of actuators and sensors, are co-located, making protection against the elements easier. Needless to say, centralized control methods are widely studied and applied. This makes centralized control architecture a very viable option for many design cases.

However, centralized controllers are generally not scalable. Adding a subsystem to an existing system and including its control in the control architecture almost always results in redesigning the full centralized controller. Also, centralized systems are limited by the number of input/output connections. Their components are usually expensive.

On the other hand, decentralized control architectures have several appealing characteristics. These include, but not limited to:

- Modular, enabling integration of subsystems.
- Flexible, using heterogeneous components.

- Scalable, such that they scale up/down as the system is upgraded.
- Better ability to locate and isolate faults.

In spite of the stated features, distributed control architectures need all aspects of the system (software and hardware) to support distribution. Designing distributed architectures requires better tools, more skills and sophisticated design methodologies. Bandwidth constraints on the communication network may limit the controller performance. Coordination controllers are often needed in between subsystem controllers to ensure stability and performance.

3.2 Intelligent Control

Intelligent control is an enhancement of tradition and controlled to include the ability to sense and decent about the environment with incomplete and in exact a priority knowledge, and executes commands and controls in an adaptivity, flexible and robust way. Intelligent control is intersection of the feast of artificial intelligence and automatic control (traditional control) [137].

Intelligent controlled methods are kept classified in the following categories[137]:

- *Model-based Methods* assume the knowledge of a mathematical model of the system under control. Intelligent control is realized using learning, estimation, and adaptation of algorithms [11].
- *Knowledge-based intelligent Methods* enjoy symbolic (nonnumerical or linguistic) knowledge and models of the system under control. They are structured like expert systems, they include infusions engines or the knowledge base.
- *Neural Network Control Methods* rely on your networks. They require training using realistic data.

- *Fuzzy Logic Control Methods* are sometimes referred to as linguistic controllers. They rely on simple intuitive if-then rules without the need for a mathematically model of the system. They are very common among intelligent controllers.

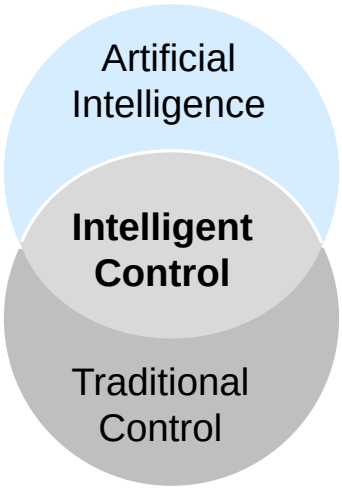


Figure 16: Intelligent Control Domain

Intelligent control has numerous published research applications. A four layer intelligent control architecture for Unmanned Aerial Vehicles (UAV) is presented in [24]. The first and lowest level layer is the inner loop control layer which directly interacts with the UAV's sensors and actuators. One level higher is the project origination may which is responsible to fit a feasible trajectory through desired waypoints. The third layer is the path planning layer which generates the waypoints. The last and topmost layer is the autonomous decision-making layer. It makes mission level decisions and assesses available control authority after failures. The multi-layer control architecture is shown in Figure 17. Even though the control architecture is composed of four layers it counts as a centralized controller since older control layers are present in the same location. Yet, it is always possible to build intelligent controllers for distributed systems.

The research efforts presented in [18] describes an intelligent control architecture

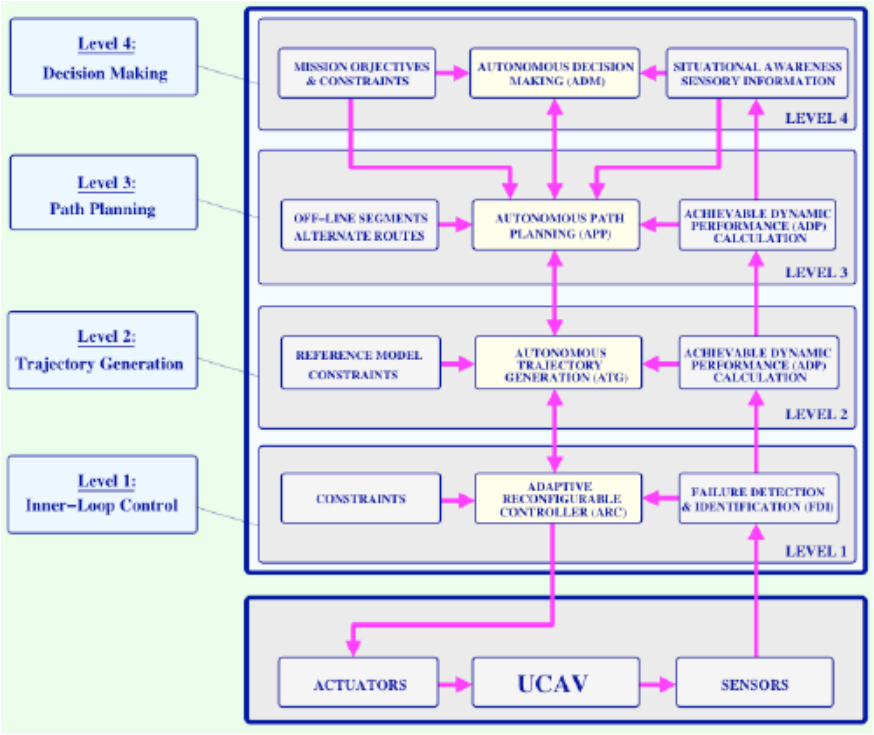


Figure 17: Structure of a UAV Hierarchical Controller (courtesy of Scientific Systems Company, Inc.)[24]

for a tactical sensor management used in distributed military surveillance operations. The architecture has three levels: sensor, platform, and group.

3.2.1 Supervisory Control

Supervisory control is a branch in controls engineering that encompasses concepts from both controls and artificial intelligence [117]. It is more focused on the high level decision making in knowledge base modules and inference engines. Such decisions are then propagated and transformed to lower level control actions executed by local controllers.

The heart of intelligent control is the knowledge base, which focuses on the problem solving tasks of a specific application. Knowledge can be represented in the form of graphs, objects, if-then rules, neural networks, computer models, ... The if-then rules technique is most often used in intelligent control. Conventional control

differs from intelligent control in that the latter separates between problem-solving knowledge, the inference engine and the algorithms used [144].

Supervisory control systems are very much suited for large scale complex systems that are required to operate as a certain level of intelligent autonomy. Of notable application to supervisory control is the research conducted at the Naval Research Laboratory for ship damage control [149], presented next as a sample application.

3.2.2 Supervisory Control Architecture Example - Ship Damage Control

The Supervisory Control System for Ship Damage Control is described in this section [149]. The objective of the supervisory ship damage control architecture is to automate the decision making task for damage control management of fire, smoke, flooding pipe rupture, and stability aboard a naval ship. It assumes the availability of intelligent sensors and actuators, with a level of autonomy higher than what was commonly the case in 2001. This is not a restricting assumption since ship design, construction, testing, and deployment programs take in excess of 10 years. The main research was directed towards studying knowledge based expert systems, machine learning in noisy environments, and assessment of supervisory human-computer interfaces.

It is composed of four subsystems: The first is the human / ship interface subsystem which includes GUI interfaces to the ship's supervisory control, scenario specification, sensors, and actuators among other functions. The physical ship simulation subsystem is the second subsystem in the control architecture. It is responsible of modeling and simulation of damage to the ship. The third subsystem is the total ship representation subsystem which contains the knowledge base about the ship. The last subsystem is the intelligent reasoning subsystem. Its task is to conduct non-numerical reasoning needed of supervisory control.

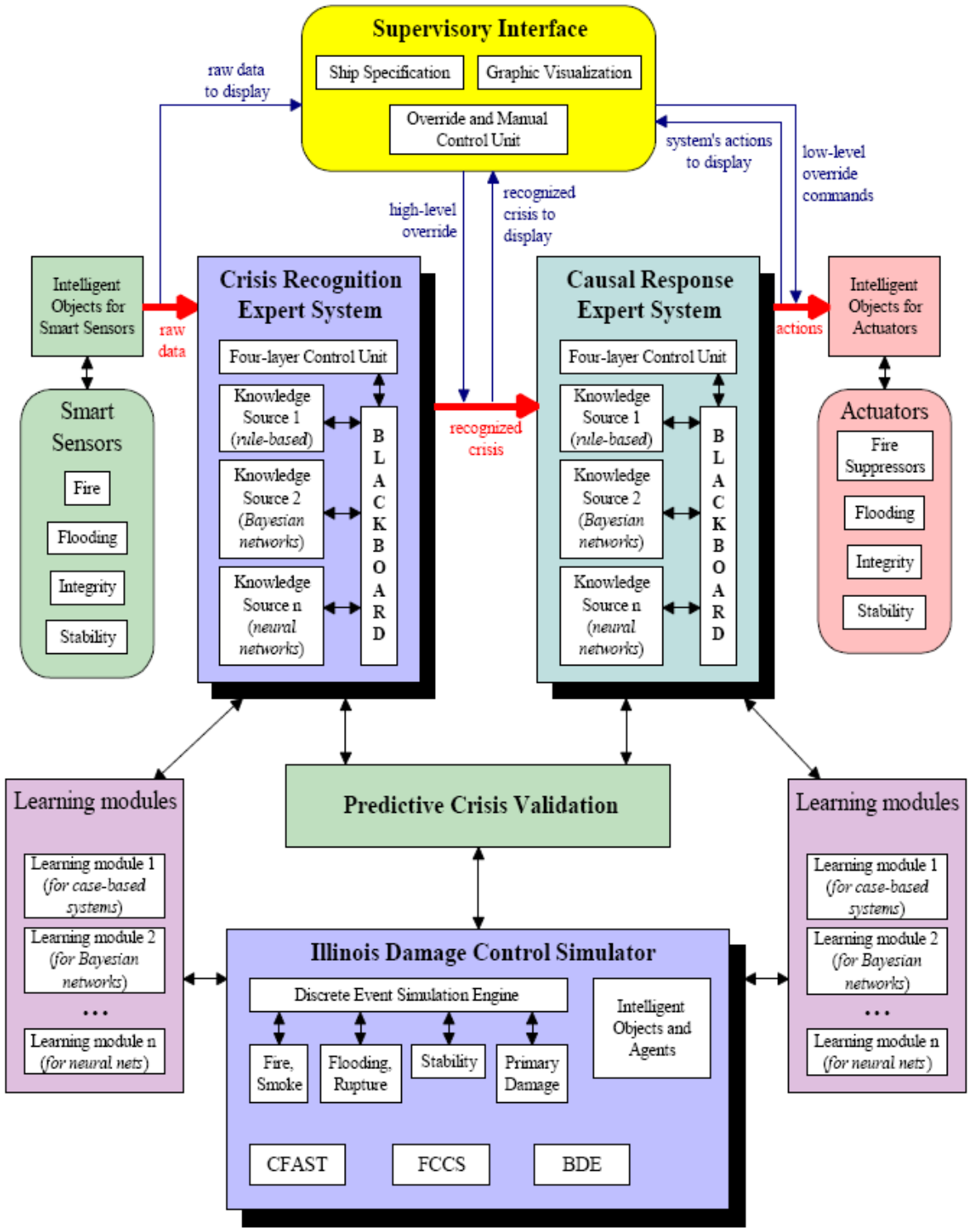


Figure 18: Functional Overview of Supervisory Ship Damage Control Architecture[148]

A functional overview of the supervisory control architecture is shown in Figure 18. It is more of a centralized, layered architecture, such that all control decisions (qualitative and quantitative) are made in one central location. However, this task is accomplished through layers within the architecture. The architecture encompasses several types of modules. A supervisory interface module that interfaces with the human user; smart sensors module (where all sensors are grouped); smart actuators module (where all actuators are also grouped); learning modules; predictive module; crisis recognition module (which functions more like post sensor processing or inference engines); and finally a casualty response system (which is responsible of making the actual control decisions).

Intelligent reasoning subsystem is addressed in [59]. It is required that the subsystem performs data acquisition, casualty detection, classification, and casualty response functions. In addition, the control architecture has to: enable situation awareness, manage limited resources, initiate preemptive actions to prevent damage, and finally conduct and monitor damage control actions in case of casualty. Situation awareness experiments are performed on the ship simulation / control architecture and presented in [140].

All intelligent control systems, and supervisory control systems included, heavily rely on software modules. Ship damage control is no exception. It is primarily composed of four main software modules performing the functions of intelligent reasoning and critiquing learning, graphic visualization, human computer interface, and simulation with scenario generation [62].

The human machine interface is described in [30] emphasizing the role of human users in supervisory control architectures. The subsystem is designed to present relevant situational awareness information as well as pass commands to the supervisory controller.

A separate effort of this research project is dedicated to scenario modeling and

generation. Multiple simulations using a wide range of scenarios can be run and result saved to a database [127]. This effort also corresponds to part of the control architecture.

3.2.3 Agent-Based Control

Multi-agent systems are built on intelligent agents. Multi-agent systems posses are proactive, reactive, and posses social properties. Agents interact in multi-agent systems to accomplish the systems task. Multi-agent systems are not necessarily used for controls but span wide variety of fields.

Multi-agent systems can be used both for system control and system monitoring. A multi-agent control architecture can help in solving distributed control problems using cooperative methods as well as negotiate control methods [76], [80], [75], [61]. In addition, multi-agent systems can provide adequate monitoring and asset management by diagnostics and protection against faults [66], [90].

A Multi-Agent System based Intelligent Control System is described in [64] to control a large-scale steam power plant. This research paper proposes a single agents architecture design (shown in Figure 19) which acts as the main building block for the control architecture. Although generic, this agent is developed with the application problem - control of a large scale power plant - in mind. The agent has a perception and an effector modules to act and react with the external environment, i.e. the power plant. In between those modules, lies the decision making module. It compares the sensed data (through the perception module) to predefined scenarios, and determines an objective and, consequently, picks a plan of action. The objective is communicated to other agents to ensure it is the best course of action to achieve a general system goal. Hence, a plan to achieve the objective is chosen, and consequently, an algorithm to execute the plan, which is then effected to the environment. Note the four main components: the decision making, the perception, effector, and

communication modules.

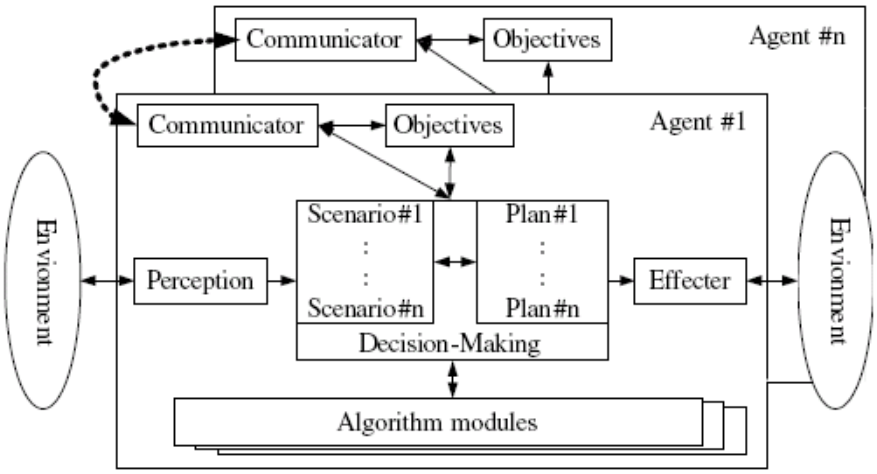


Figure 19: Single Agent Architecture [64]

Figure 20 shows an organization for the multi-agent system as the basis of the intelligent control architecture [64]. The low level intelligent agents perform the local control or local objective achievement tasks. It is not clear from this research, but it seems that the low level agents are directly associated with hardware components. They communicate with each other and with higher levels. The middle level layer contain agents that either act as mediators between low and high level agents, or monitor the performance of low level agents, or both. They seem more like functional agents rather than associated with specific subsystems. The high level agents are responsible for overall task delegation, or top level courses of action. Since the high level agents are task based, then they are organized in a functional hierarchy (each agent performing a certain high level task), rather than a component or local subsystem hierarchy.

3.2.4 Artificial Intelligence

Artificial intelligence is the automation of activities that we associate with human thinking, activities such as decision making, problem solving and learning [17]. It is the art of creating machines that perform functions that require intelligence when

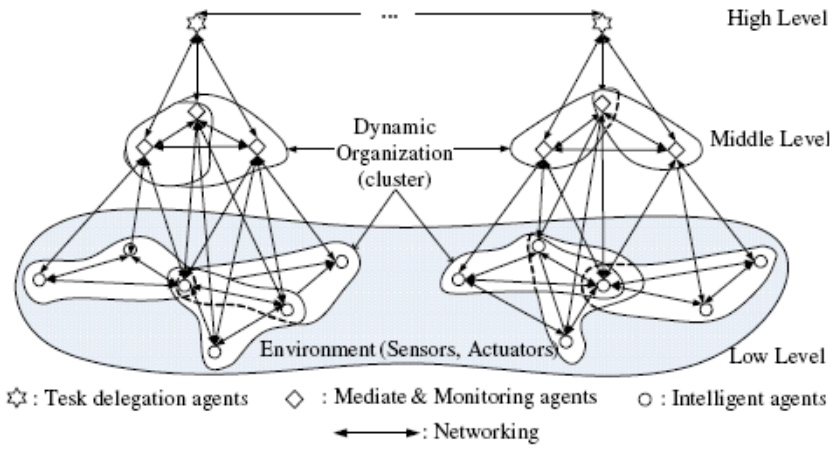


Figure 20: Multi-Agent System Architecture [64]

performed by people [82], things at which, at the moment, people are better [111]. Artificial intelligence relies on intelligent agents (an entity or module that perceives its environment and takes appropriate actions), such that these agents take the best possible action in a given situation [114].

Artificial intelligence can be implemented using several methods. Artificial neural networks, fuzzy systems or fuzzy logic, evolutionary computing, and swarm intelligence are among some. The fuzzy systems method is very appealing since it is intuitive and requires no mathematical models.

The application of artificial intelligence in control architectures lies mainly in decision making. A localization method for mobile robots is given in [97]. A robust method is proposed that uses ceiling landmarks, through a vision system, to localize mobile robots. The robot looks up towards the ceiling, acquires a series of images, detect ceiling features from this images, hence locating its current position and making decisions of translation and rotation to reach a required final position.

3.2.5 Resource Allocation

These software modules enable the autonomous, emergent, and learning behavior generation in intelligent systems. [123] uses a supervisory control architecture in

addition to resource allocation techniques for intelligent behavior generation.

3.3 Network Control

A major concern for the networked control architecture of a large-scale system is the complexity due to the number of components and their interaction patterns and communication delays [64]

3.4 Model-Based Architecture Representation

Currently, model-based approaches to representation of different system designs are widespread and successful. Model based design has different methodologies [69]. One such methodology is platform based design [28]. The platform acts as an abstraction layer that can hide underneath several practical implementations (designs) geared towards different applications. Platform-based design is a half-way process where successive practical application of specifications meet with abstractions of potential designs [43]. Another methodology in model based design is actor based design [**?**]. Actors are concurrent components that communicate through ports and interact according to a common patterns of interaction. Primarily, actor oriented design allows designers to specify interaction between components separately from the definition of component behavior [84]. Actor oriented design in software engineering is analogous with agent-based design in systems engineering. A Simulink® model implementation is a clear example of an actor based methodology.

3.4.1 Unified Modeling Language (UML)

UML provides a way of graphically depicting software structure. It can be extended to depict organizational architectures and similar applications. One of the main advantages of UML is the ability to tackle complex structures making it easier on practitioners to address the design problems of such structures. Nevertheless, UML cannot fully define relationships between components or diagrams [44].

Table 1: Basic Comparison of UML and AADL [44]

	UML	**AADL**
Origin	Diagrams tradition	Language tradition
Purpose	Depict functional structures	Define runtime behavior
Representation	Diagrams; graphic	Textual and graphic
Verification	—	Automated Analysis
Current Domains of Use	Software, business processes and similar	Embedded and real-time software systems

3.4.2 Architecture Analysis and Design Language (AADL)

Unlike UML, AADL has its roots in a computer language tradition rather than a graphical interface. It was developed as a programming language to represent software architectures in a text form. More importantly, it allows for the formal definition of the syntax and semantics. Yet, AADL gives the software designer the option to graphically depict the system [44]. By having its basis in computer languages, AADL architecture representations (even if presented graphically) avoids ambiguity. This is another advantage over UML. A summary of the basic comparison between UML and AADL is adapted from [44] and presented in Table 1.

It is worth mentioning that AADL can handle automated analysis for verification. This means AADL models can be executed, producing results relevant to the system verification, i.e. the system represented by the AADL model. UML only provides a functional breakdown with no real automated verification capability.

3.4.3 Domain Specific Modeling Languages (DSML) Semantics

Domain specific modeling languages are used describe systems using domain specific abstractions in the form of abstract system models. They are also used in the structural analysis of such models [69]. (Note: this does not necessarily refer to the structures realm as in frames or beams, but to the broader perspective of how components within a system are connected and built, and how they interact and affect each other). DSML were first developed within the software design arena.

3.5 The Reference Control System

The RCS reference model is used as a basis for control architectures in various control applications. A submarine maneuvering system demonstration of the RCS is presented in [67]. Figure 21 depicts a simplified submarine maneuvering control architecture. Several observations are clear from the figure.

1. The control architecture is hierarchical. High level controllers function as top level decision modules. For example, the "Course" controller request a submarine heading while the "Maneuver" controller computes the details of the submarine trajectory. Mid level controllers use the higher level commands to compute general subsystem actions. For example, the "Propulsion" controller determines the amount of power required. Information flows to the lower level control which decides on specific control commands to actuators, such as a commanded shaft speed issued by the "Turbine" controller to the gas turbine.

2. Controllers can communicate with other controllers, and not necessarily with physical components. A controller can communicate with a parent controller and several child controllers. In addition, controllers can communicate with others on the same level.

3. The control architecture is layered, with each layer issuing specific system level decisions within its scope.

4. The controllers are distributed based on a functional hierarchy on the top levels (maneuver, propulsion, depth, helm). On lower levels, controllers are specific to local components (gas turbine, main ballast, sail plane, stem plane, rudder), hence a spatial hierarchy.

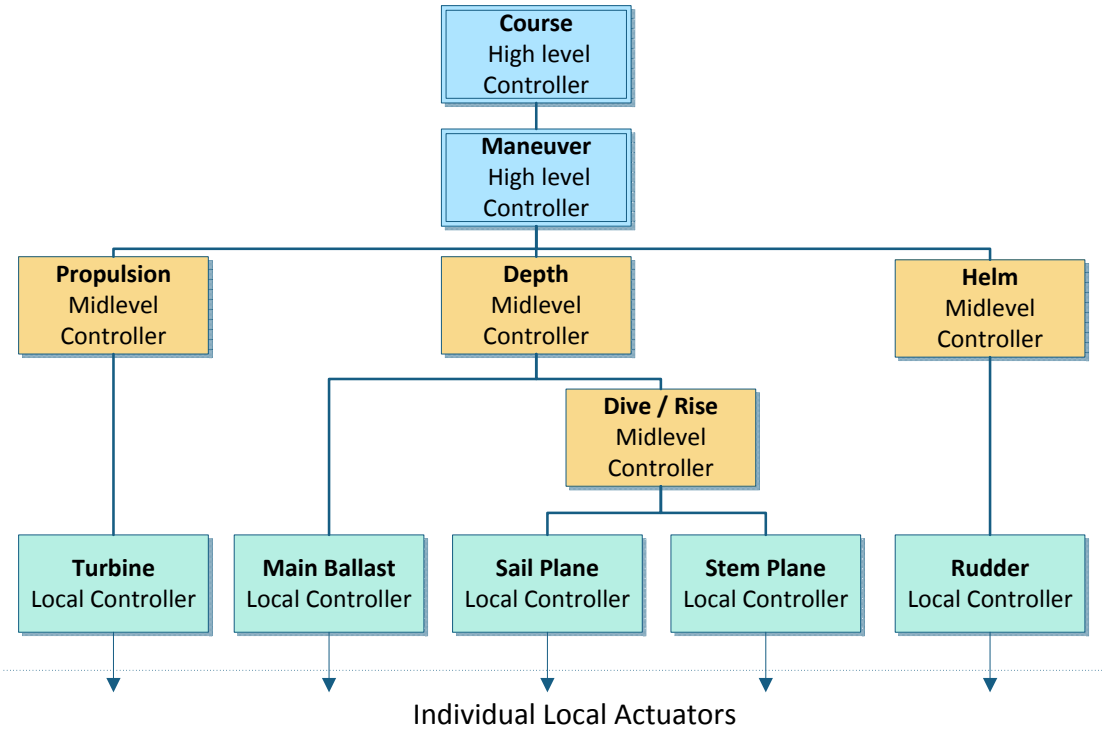


Figure 21: Simplified Submarine Maneuvering System [67]

3.6 Case Study: The F-16 Flight Control System

This section is based on the AIAA professional study series case study presented in [32]. The objective is to draw some conclusions about control architecture design, and how the physical system affects the control architecture structure and modules. In addition, this section gives insight on the control architecture design process for complex systems.

The F-16 is a light weight fighter and had been an extremely successful aircraft along the years, serving since 1974. Its design is based on its predecessor: the YF-16 prototype. All the physical systems of the F-16 are either copied or modified YF-16 systems, with the exception of the fly-by-wire flight control system discussed later. The fact is that all aircraft systems went through the conceptual and the preliminary design cycles before the control architecture design was tackled. The YF-16 systems design was slightly modified in the detail design level of the F-16. For example, the

fuselage was extended by about 10 inches.

The YF-16 flight control architecture was more on the conventional side, with mechanical linkage between the pilot commands and the control surfaces. The F-16 control architecture, represented by the fly-by-wire flight control system, was completely designed from scratch, replacing the YF-16 conventional system. It is important that this was done based on the - already designed - YF-16 physical design.

The next section presents the fly-by-wire control system, together with its advantages over the conventional system. Also, the thought process driving the design tasks are outlined. Important characteristics of the fly-by-wire system are the multiple redundancy of its components and the online manipulation of pilot inputs to keep the aircraft within its safe flight envelope. Both characteristics are discussed later in the context of control architectures.

3.6.1 The F-16 Fly-By-Wire System

The fly-by-wire control system is one major feature that distinguishes the F-16 from its predecessor the YF-16. Initially, a case for a complete fly-by-wire control system could not be made. General Dynamics attempted at maximizing maneuverability. This was accomplished by a variable wing camber, so as to maximize the lift to drag ratio. A leading edge flap is programmed as a function of angle of attack and Mach number while a trailing edge flaperon position varies as a function of Mach number and landing gear position. A control architecture set these aerodynamic surfaces' angles depending on the mission phase. This requirement was in favor of a fly-by-wire control system as compared to a fully mechanical system.

General Dynamics had developed F-111 with a triple redundant, large authority stability, and command augmentation system. Hence, the company was uniquely qualified to develop a fly-by-wire control system. Almost all the significant issues of the fly-by-wire had already been addressed in the F-111 program. For the small size

of a lightweight fighter, the F-16, a fly-by-wire control system could not show a weight advantage over a mechanical system. The reason is that the added components of the electric power system (independent power sources, batteries, etc.) increase the overall weight of the system. In addition, the fly-by-wire control system did not show an advantage in cost since it adds complexity to the system due to the redundancy requirements.

Aerodynamic instability and control engineers finally settled the debate between the fly-by-wire control system and a mechanical system. The F-16 took advantage of the relaxed static stability concept (in which the aircraft is slightly unstable in the longitudinal direction) to improve its maneuverability and performance. This requires constantly adjusting flight controls by the pilot to maintain static stability, which is impractical, or an automated controller that does the same task without pilot intervention. Such concept is referred to as the stability augmentation system and can only be accomplished through a highly reliable electrical control system. Hence, one of the main factors that tipped the scale in favor of the fly-by-wire system was the relaxed static stability requirement. The pilot's command signal can now be augmented with the stability system command signals with no penalty. A mechanical system is incapable of such function; hence, the mechanical linkage between the pilot and the aerodynamic control surfaces is not needed.

In addition, the F-16 is a highly maneuverable aircraft. The pilot might fly the fighter into unstable regions within the flight envelope. Stall, spin, as well as undesired aircraft attitude behavior are some of the flight incidents that need to be avoided. The major benefit of fly-by-wire is the ability to tailor the system's characteristics at each point in the aircraft's flight envelope [55]. The introduction of digital computers allows more complex algorithms to be implemented; however, the F-16 relies on an analog system. The fly-by-wire control system has several benefits, such as care free handling, optimized handling qualities, aircraft agility, aircraft performance, thrust vectoring

utilization, lower drag due to optimized trim settings, and aircraft reconfiguration [55]. The subject of constraints on pilot inputs or control signals is discussed in more details in a later section.

Hence, the aircraft design requirements drove the aircraft physical systems design as well as the control architecture concept. A mechanical flight control system could not have satisfied the design requirements, neither on the stability level nor on the performance level. Yet, this result was reached after the aircraft systems were designed, manufactured, and tested on the YF-16. It is important to emphasize that the state of the art design practices treated the control architecture design (the fly-by-wire flight control system) as a "post design task" (after the YF-16 was designed) even though the control architecture is a main system requirement.

A simplified schematic of the fly-by-wire flight control system is shown in Figure 22. The original F-16 fly-by-wire architecture has no specific structure; all controllers and sub controllers are connected on the same level. There is no hierarchy, nor supervisory control nodes. In fact, centralized control is used on the F-16. However, close examination of the architecture yields the perspective presented in Figure 22. The controllers are categorized into three types: controllers applied to overall aircraft functions, miscellaneous controllers, and controllers directly linked to actuators.

The pitch, roll, and yaw are the first type of controllers, each controlling a specific mode of aircraft attitude. The pitch controller is the most complex since the aircraft is unstable in the longitudinal direction. The roll controller passes the commanded roll rate to the yaw controller in order to coordinate turns, especially at high angles of attack.

The second type of controllers are more of the coordination controllers. The gun compensation controller makes sure the aircraft keeps its attitude during firing the gun; hence, it sends control signals to the roll and yaw controllers. The trailing edge flaps sends commands to the flaperons to act as flaps (and ailerons in case of

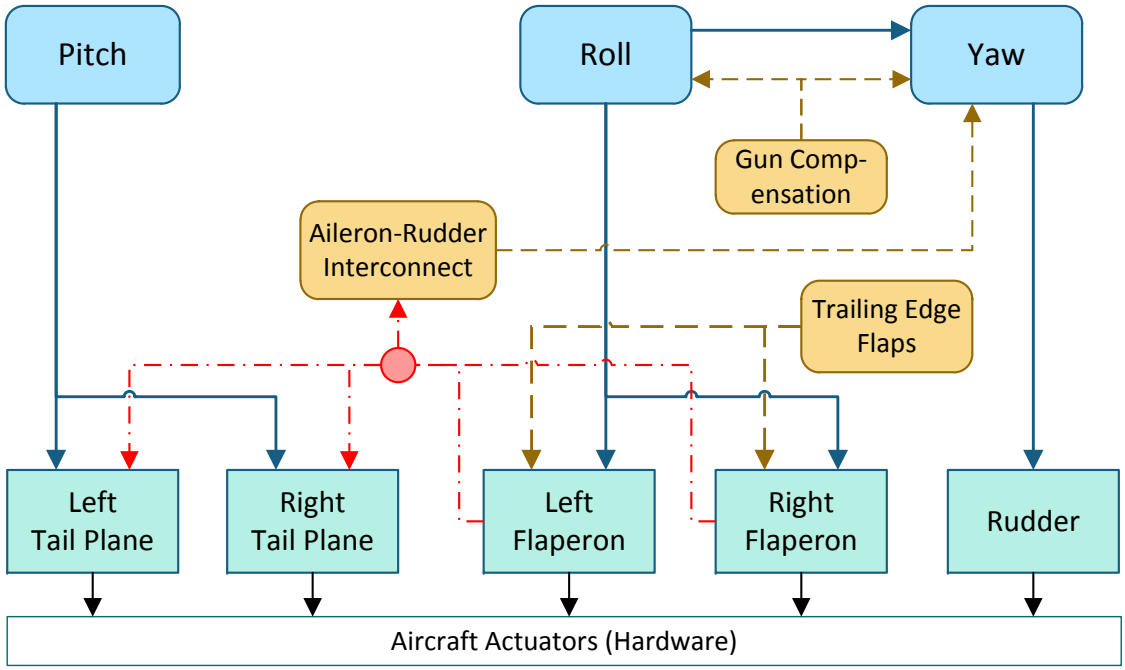


Figure 22: The F-16 Fly-by-wire Flight Control System [32]

turning) during landing. The aileron rudder interconnection controller makes sure the aircraft's performance is safe and acceptable during high angle of attack maneuvers. Note that in the absence of a fly-by-wire system (a mechanical or hydraulic system instead), the pilot is responsible for carrying out these tasks. The fly-by-wire enables the aircraft's care free handling.

Controllers linked directly to the actuators constitute the third type. Since the flaperons act as flaps, ailerons, and air brakes, they have the most complex structure, receiving inputs from the roll controller and the flaps controller. They also initiate correction signals to the aileron rudder interconnection controller and to the tail plane controllers.

The F-16 control architecture has no standard structure. It seems that all control tasks are listed, a controller designed for each, receiving appropriate inputs from other controllers, followed by connecting all controllers in one architecture. No formal functional, or spatial, decomposition is set in place. This results in a very limited

control architecture scalability and upgradability potential. The only modifications that can be applied to the control architecture are tweaking the controller gains or changing the ranges on signal limiters. Adding a controller for an additional subsystem most likely involves redesigning the full control architecture.

3.6.2 Redundancy Study

Redundancy is a term used in failure tolerant systems to indicate the level of protection against failure. The level of redundancy (or protection against failure) is determined by the number of similar failures that can take place within the system without loss in capability. For example, a two-fail operational system can sustain two consecutive similar failures while maintaining its main capability. Fail safe systems, on the other hand, suffer degradation (in capability) on failure, without preventing safe operation.

For the F-16 case, it was found that redundancy comes at a cost beyond monetary cost. Although redundancy reduces the chance of catastrophic system failure (the aircraft not returning safely to base), it decreases reliability resulting in an increase in the probability of mission aborts.

The F-16 fly-by-wire system redundancy ranges from none to quadruple, while the failure protection varies between none and two-fail operative/fail-safe. Figure 23 shows the redundancy and failure protection levels on the F-16.

Note that the control architecture is not applicable to all modes of faults/failures. For example, if one component fails in a triple redundancy subsystem (or branch, component, etc.), the control architecture reconfigures the system maintaining the same capability. Failure of two components, however, switches the control architecture to the fail-safe mode, i.e. from maintaining capability to degradation of service ensuring pilot/system safe return. No control architecture is designed to "do it all". In certain mission or failure scenarios, a control architecture designed for normal

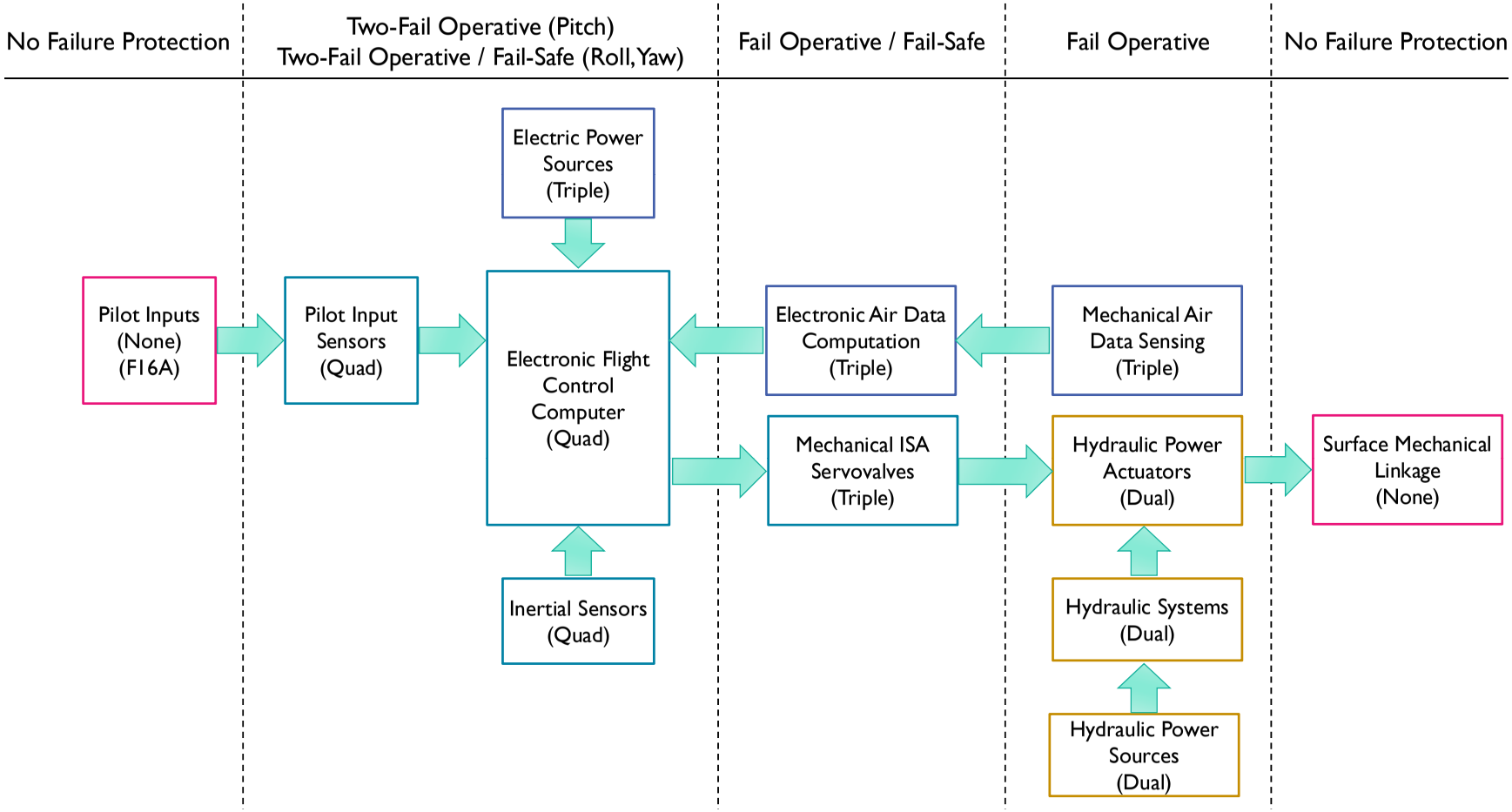


Figure 23: The F-16 Fly-by-wire Redundancy [32]

operation has to be abandoned for another specific to abnormal conditions.

3.6.2.1 Signal Data Branches Redundancy

Each component receives control signals from a selector module connected to the four F-16 electronic data branches. The choice between different branches is made using a predefined algorithm. During normal operation, the selection is made between branches A, B, and C, such that the algebraically middle signal is selected, while branch D is held in standby mode. If one branch fails, it is replaced by branch D, and selection is as before. In case two failures take place, the branch carrying the middle value is chosen until the failed branch is identified, and then the branch with the smaller signal absolute value is selected. If three branches fail, the fourth is selected (after the failed ones are identified). The algorithm is hardware implemented using analog and logic circuitry (which was referred to as "modern" in the 1970s).

The algorithm is designed based on previous experience with the F111 in the 1960s. However, no other algorithms were tested. This is understandable since testing new algorithms (in the 1970s) means risking a F-16 prototypes and pilots lives, together with substantial increase in cost. However, computational resources and simulation approaches have come a long way since. The preselection of control architecture algorithms need not to be the case nowadays. In addition, algorithms can be implemented using software enabled control techniques, making swapping of algorithms an easier and more feasible task than what was available in the 1970s.

3.6.2.2 Electric Power Redundancy

The fly-by-wire of the F-16 must have a redundant source of power that can provide uninterrupted power with at least the same reliability as the control system. Hence, the F-16 was designed with three main power sources: the engine, the emergency power unit, and the batteries. Further redundancy was achieved by adding another generator in the emergency power unit and with multiple batteries.

However, the part of the control architecture responsible for switching between power sources was a significant problem with the electrical system. The design was heavily biased towards a fail operative battery operation, to the extent that maintenance ground crews would cause batteries to be engaged during normal ground operations. Many pre-flight self-tests found the batteries completely drained, resulting in delayed the flights.

Modifications to the electric system control architecture eliminated this problem. Yet, this was done after the F-16 was manufactured, and the problem became persistent. Software enabled engineering design, simulation, and proper architecture representation can detect such problems early in the conceptual design phase, resulting in considerable resource savings.

3.6.3 Performance Limits and Constraints

The F-16 fly-by-wire control system objective is to truly achieve "eyes out of the cockpit" maneuvering in combat arena. This means that the pilot should be less concerned with the instruments and more with flying the aircraft, thereby achieving maximum combat effectiveness. Hence, it was a necessity that the control architecture constrains pilot inputs (and the corresponding aircraft states) to within safe limits of operation.

3.6.3.1 Angle of Attack / Load Factor Limits

The F-16 was designed to handle 9g limit normal load factor throughout the operating envelope. However, the F-16 had a tendency for the to nose rise at the termination of a high angle of attack rolling maneuver. The F-16 angle of attack versus the load factor limits is shown in Figure 24. It shows that the maximum effect of the roll rate input is a 4 degrees decrease in the allowed command angle of attack during rolls.

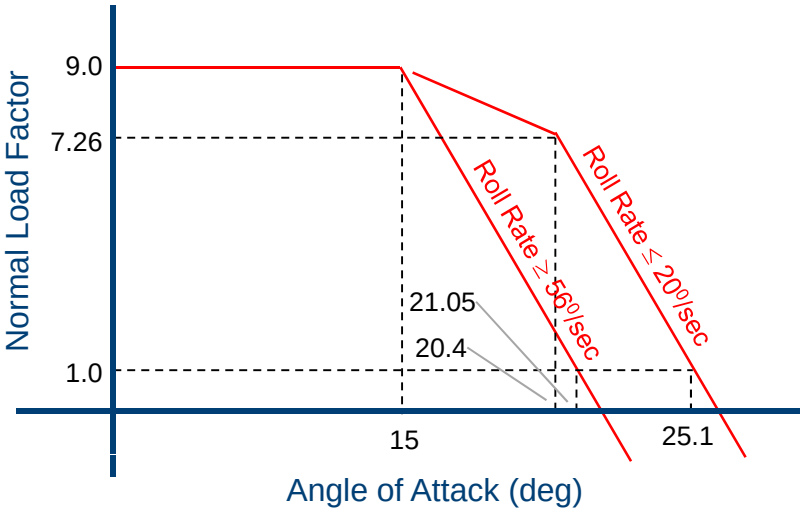


Figure 24: The F-16 Angle of Attack / Load Factor Limiter [32]

3.6.3.2 High Angle of Attack / Rolling Coordination

The F-16 is capable of well coordinated rolling maneuvers at maximum angle of attack without pilot coordination, using only lateral control. This function is accomplished using: an aileron-rudder interconnection module and a roll rate to rudder cross feed. The aileron-rudder interconnection scheduled is shown in Figure 25 for low Mach numbers. The control architecture has to take this constraint into consideration.

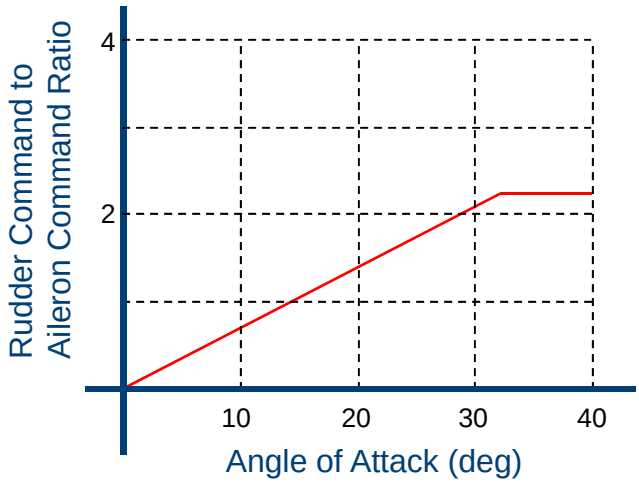


Figure 25: The F-16 Aileron Rudder Interconnect [32]

3.6.3.3 Roll Rate Limiter

The F-16 has a requirement to be able to perform maximum command rolling maneuvers at load factors from 0 to 0.8 normal load factor, at air speeds above 100 knots, through 360 degrees. Analysis as well as flight testing showed that departures could occur above approximately 20 degrees of angle of attack. The simple solution of reduction of the allowed roll rate command as a function of angle of attack yielded unsatisfactory results. Another approach was to reduce the maximum roll rate command as a function of impact pressure, which worked, but also resulted in penalizing the high angle of attack roll performance.

The solution was to add a function that uses the impact pressure, angle of attack, and the horizontal tail position, with appropriate weights on each, to calculate the maximum roll rate command. This is another module that had to be added to the control architecture.

3.6.3.4 Rudder Fadeout

At sideslip angles of approximately greater than 10 degrees, the F-16 exhibits directional instability. The rudder power available for the F-16 is sufficient to drive it into the unstable region. In order to always operate in the stable region, the rudder authority available to the pilot was reduced as a function of angle of attack. To allow the pilot to make rudder assisted the rolls, the rudder fadeout schedule was furthermore defined to include a roll rate function. The final schedule is shown in Figure 26.

3.6.3.5 Yaw Rate Limiter

Wind tunnel tests of the 1/25 scale F-16 model showed that it might exhibit a fast flat spin mode from which recoveries might be unacceptable. The F-16 enters the region of unacceptable recovery at the spin rates greater than 0.35 revolutions per second. Hence, a yaw rate limiter was developed. The way it was implemented was to replace the normal commands to the ailerons and rudder with anti spin commands

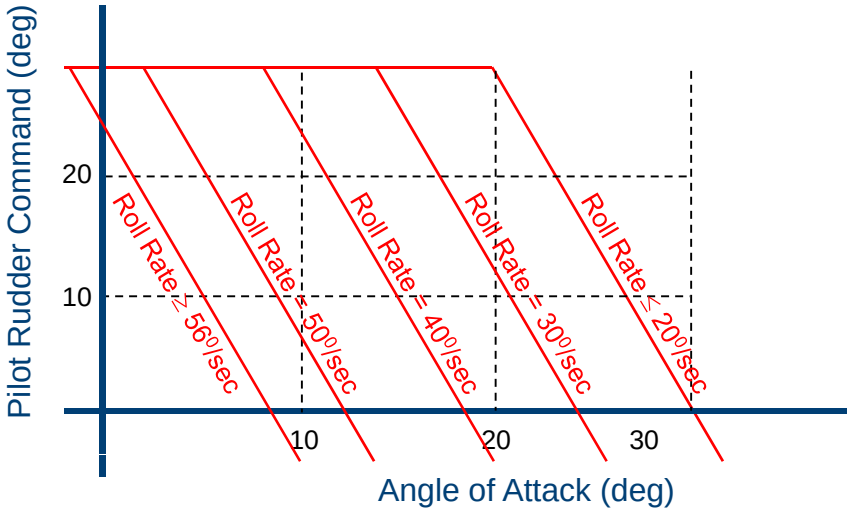


Figure 26: The F-16 Rudder Fadeout [32]

outside the acceptable boundaries of operation. The yaw rate limiter is activated at angle attacks higher than 29 degrees, at which the normal commands to the ailerons and rudder are replaced with commands proportional to the yaw rate.

This means that certain logic has to be implemented in the control architecture as well as the limiting constraints described in this section and previous sections. The control architecture in the F-16 case is mainly built of analog and logic components. Modern control architectures are heavily based in software engineering, hence more complicated logic, rules, and intelligence can be embedded more easily.

3.7 General Observations

The various literature search resulted in several observations. The observations are categorized in three main categories. The first relates to the design of intelligent systems. The second addresses the structure of cultural architectures, i.e. the components, their connectivity, and their arrangement. The third category relates to the standards by which control architectures are represented.

3.7.1 Intelligent Systems Design

- In many cases it is clear that the control architecture is designed in an ad hoc fashion. Components are just put together to achieve a specific functionality after the physical system is already designed. There is a huge void of control architecture design processes and methods in the literature. The only standard design approach is for traditional control systems. But as far as intelligent controlled architectures are concerned, no method or process exists that systematically takes a physical system and transforms it into an intelligent system by adding an intelligence architecture.

- All control architectures studied are designed for a particular physical system. Their physical system parameters, configuration (or structure), components, connectivity, and function are always given. Hence the physical system is beyond the conceptual design phase in its design cycle.

3.7.2 Control Architecture Structure

- Some intelligence control methods required inference engines; some require a software model of the physical system; others require a database. However, all intelligent control architectures require a decision-making module. The decision-making module is implemented using different algorithms, depending on the application, the function and the control architecture overall structure.

- Control architectures are found in layers, with the three layer structure being the most common. The top-level layer usually functions as the decision-making layer, generating general decisions and policies for the whole physical system. The mid-level layer takes the top level decisions and translates them into general actions that subsystems need to take. The lower-level layer directly interacts with the hardware.

- Almost the only intelligent control architectures are distributed.
- All intelligent control architectures are based on software.
- All distributed control architectures have separate communication modules.
- Control architecture structure built of nodes, which represent sub-controllers.
- Since many control architectures are layered architectures, communication between nodes takes place both across layers and within the same layer.
- The hierarchical control structure is a very common control architecture used with many physical systems.
- Distributed control architecture is also a common occurrence. However, due to limitations on stability, it is less common than the hierarchical control architecture.
- There has been several systems that combine hierarchical structure and distributed structure in one hybrid control architecture.

3.7.3 Control Architecture Standards

- Control architectures have no standard form of representation in the literature. Some attempts at this effort are AADL and UML. However, AADL is more concerned with the computing hardware or processors and with communication, and less concerned with the function of each module within the control architecture.
- There is a huge overlap between control methods. For example, operative control can be agent-based or intelligent. Agent-based control can be fuzzy, distributed, or hierarchical; distributed control can be supervisory control. No clear distinction is being made between what the control architecture does, how it is structured, how it accomplishes its function, and what algorithms it uses.

- It seems like there is a multidimensional continuum of control methods and control architecture structures.
- Attempts at standardization are almost always developed within a specific discipline or domain. A standard representation of a general control architecture is put together with an application in mind rather than for a general purpose intelligent system.

CHAPTER IV

INTELLIGENT SYSTEMS DESIGN METHODOLOGY

4.1 Traditional Systems Design Process Methodology

The system of systems design process has several variations across the literature, refer to examples in [147], [103], [78] and [116]. The scope of systems and engineering design range from manufactured products development [45] to software and information systems development [83] to industrial systems [139] among other areas.

All system design methods share common steps. Differences among the processes are usually domain specific although few basic differences exist. However, there does not exist a standard process for the design of advanced concept systems, such as emergent system, resilient systems, and distributed intelligence systems ... etc. A representative general design process is shown in Figure 27. The process applies to the physical system design, not to the intelligence / control / decision making architecture design.

In brief, the process starts with the problem definition step which results in the problem statement, benchmarking against current state of the art, identifying the constrains, and considering conceptual trade offs. The next step is defining the objective and success criteria that render a particular design superior to another. The tests or experimentation plan should be proposed at this step. Note that these two steps are not standard. Some designer rename and rearrange them and their components in the form of problem requirements and system analysis. Regardless of the approach, the output from these steps is what is important at this point: the definition of the system, its requirements, its criteria of success and how it compares to state of the art.

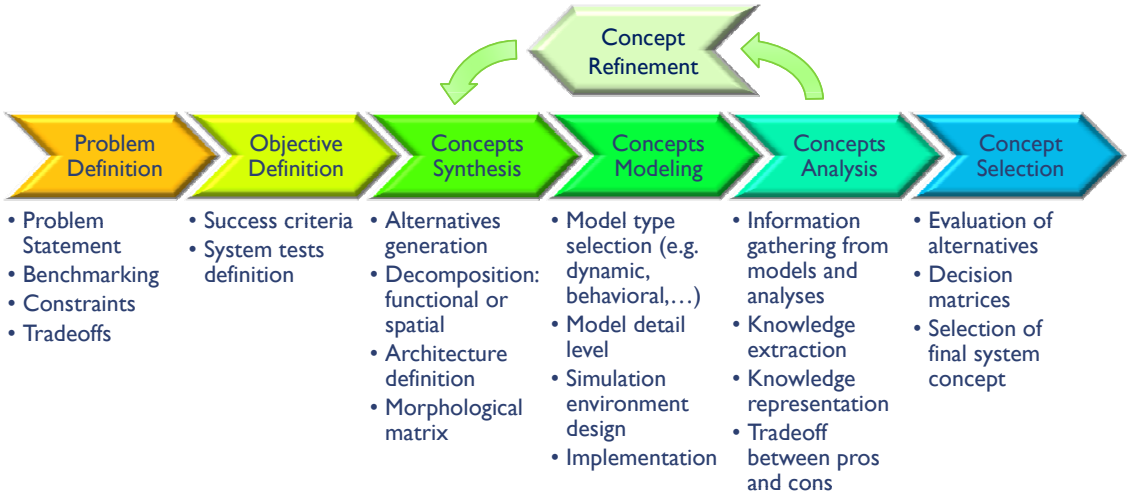


Figure 27: Typical Conceptual Design Steps

Concept synthesis follows. It is putting together a concept solution for the physical systems. Automated or manual concept synthesis can be employed to generate alternatives. The system is decomposed functionally or spatially. The architecture has to be defined for the candidate concept. A morphological matrix helps collect the alternatives and rule out some of the immediately obvious and infeasible ones.

The concept is then modeled. All modeling practices apply at this step (refer to Section 1.7.1). The simulation environment is also set up at this step. The model and simulation environment together present the main enabler for concept analysis.

4.2 Traditional Control System Design Steps

Control systems design is not a standard process. Yet, from a practical perspective, almost all control engineers will cross several given milestones before a controller is realized. A traditional (and general) process is presented in this context gathered from the common control systems practices but should not be taken as the standard process. Some control engineers will choose to overlook some steps or add others. However, there are several items in this traditional process that are common among any control system design exercise, namely the input to the process and the output

from the process. The process input is the physical system to be controlled. The process output is the controller architecture which can be summed up in the control structure, the control technique and the controller itself.

The traditional process starts with the physical system. Almost all control and intelligence design methods rely on an existing system, either on paper as in detailed design, or a physically realized manufactured system. For example, a control architecture is designed for a naval ship after the ship has been designed, and in some cases, built. Note such an input implies that all the physical system's architecture and configuration decisions have already been finalized. In fact, traditional controller design requires all the physical system's parameters be be set, finalized, and depending on the control technique, may only allow a prescribed limited variance in these parameters.

The first task a control engineer addresses in the design process is modeling. Although a wide variety of modeling approaches are available, the control designer will usually use continuous time models dynamic systems. Decision making systems can use discrete models, agent based models, or some other approach. Modeling involves mathematically representing the physical system, applying any simplification assumptions, and above all, the task of linearization. There are no controller design methods for nonlinear systems except in certain restrictive design cases. Therefore, usually control engineers will linearize the system by applying linearization assumptions and choosing one (or more) operating point(s). Limits of validity of the system model should also be determined.

Once the software model is realized, analysis is performed. The control objectives are determined, i.e. what is being controlled. Also, although the model was linearized earlier, the implications of this simplification on the behavior of the model, and how it represents the real physical system, should be studied. Constraints of operation are either concluded or set at this point, which allows for a reachability study.

Next decisions regarding the control configuration are made. The structure of the controller is set at this point. This step is the core of the control and decision making architecture. It is to be emphasized here that because the physical system's design is finalized (or almost finalized), the control engineer has very limited freedom in his choice of the control or intelligence structure. The reason is that certain physical system's configurations dictate the control structure. The control inputs for the physical system have to be chosen, and so does the locations and types of sensors and actuators. Other components in the control architecture are also chosen and their functionality and behavior are clearly specified. Intelligent systems require a communication network (and in some cases a distributed data storage). Hence, latency and connectivity issues need to be resolved too. Example of controller configurations include, but not limited to, hierarchical control, distributed control, centralized control ... etc.

The controller technique step follows. The general control objective implementation method is the main output from this step. It is at this step that the controller class, subclass, and family is set, as given in Section 5.2.1. If any optimization is required, the optimized states have to be decided upon, together with the numerical algorithm used for execution. It is to be noted that similar to the previous step, there is not much freedom in choosing among a wide spectrum of controller techniques or methods. The reasons are that the physical system is already designed and set, and the control configuration is also set from the previous step. Another important reason is that control engineers usually specialize in one or two controller classes (refer to Section 5.2.1), hence there is an underlying bias in the choice of a particular control technique over another by a specific designer. This is in addition to the fact that different models and simulation results cannot be directly compared.

So far, the control engineer spent considerable effort building a simplified enough model of the physical system, so as to be used in the controller design. Not much

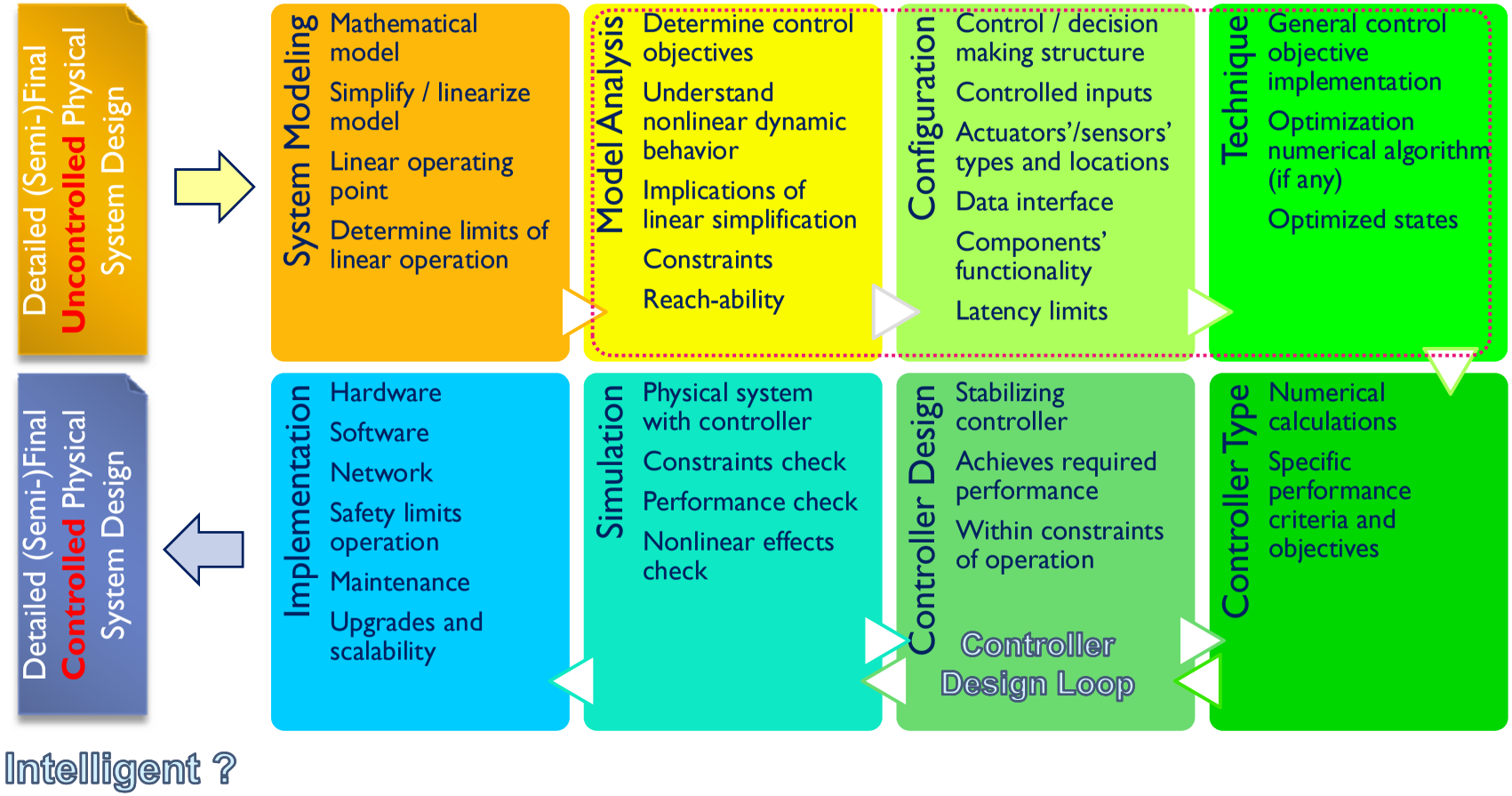


Figure 28: Traditional Controller Design Steps

time was spent on choosing the actual control configuration, nor on the control technique. These were just picked either from experience or from what the physical system dictates. The next three steps are referred to in control engineering literature as controller design. Controller design is one of the most addressed topics in the discipline. It is here where the control engineer will spend most of his work in the controller design loop.

Firstly, the controller type is chosen. This refers to how the calculations will be done, and what specific performance criteria or final objectives are implemented. Specific parameters for the controller are proposed, so as to perform two main tasks: maintain stability and achieve performance metrics, all within the range of the operation of the system. This controller is then implemented in software model simulation, or applied to the mathematical model to check for constraints, performance, and stability. If the results are not acceptable, another iteration should be performed, and this continues in a design loop until acceptable controller performance is reached. The output of this step is an actual controller that can be directly implemented on a physical system.

It is to be noted and stressed here that the controller design loop only involves choosing "numerical values" for the controller, based on a picked control configuration and technique. The design loop does not include the more system wide influencing tasks of control configuration and method (technique).

The final step in traditional control system design is the implementation of the controller of the physical system. This includes resolving all hardware issues, software issues, networks, limits of operation, maintenance plans, upgrades, and scalability potential, and so on.

4.3 The Controller Design's Three Basic Decisions

There are three basic decisions a control engineer will have to make during the design of any control system, regardless of intelligence, namely: the control configuration, the control method, and finally the controller parameters. This section describes each and its implications on the design process of intelligent emergent systems.

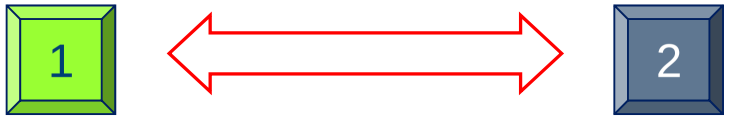
4.3.1 Control Configuration or Structure

Control structure, sometimes referred to as control architecture, is the top level decision. It answers questions related to how the control/intelligence/decision making subsystem is setup. Three main topics are addressed by the configuration or architecture, namely the components included in the system, their connectivity, and arrangement or hierarchy.

A full definition of the components found in the control architecture is essential in defining the configuration. The component's type, function, behavior, inputs, and outputs are all included in the component definition, together with the location and replication of the component.

Currently, the state of the art in control systems design is to simply *presuppose* or *pick* the control configuration, basing this decision on experience and on the physical system configuration (refer to Section 5.2.1). The fact that control architectures are tackled after the physical system design is done allows for this practice. In case of intelligent emergent systems that have no precedence, the current practice will not be feasible since the physical system is still in the conceptual design phase. It is proposed that the control configuration be accomplished through functional breakdown analyses, spatial breakdown analyses, or any other method borrowed from the systems engineering design process.

Three Decisions in Control Systems Design



Configuration

- Independent variables
- Control system components and interconnections
- Hierarchy
- Communication
- Actuators / sensors
- Hybrid / digital / continuous …etc.
- State of the art
  - Presupposed based on experience
  - Follows the detailed physical system design

Method

- PID / LQR / MPC / NN / cooperative …etc.
- Placement in the chosen architecture
- Choice affected by:
  - Required accuracy and range
  - Computational constraints /
  - …etc.
- State of the art
  - Also presupposed
  - Usually merged with step 1

Controller

- Tweaking of controller parameters
- Gains / time constants / poles and zeros
- Sampling time
- Gain calculation
- Based on control approach
- State of the art
  - Sophisticated numerical techniques
  - Sometimes only possible through iteration or simulation

Figure 29: Control Architecture Requires Three Major Decisions

4.3.2 Control Method or Technique

The control method is the approach the control architecture (as implemented in the control structure) will use, so as to calculate a suitable control law. For example, a control method might be the optimization of a certain objective function of states and inputs (such as in optimal control), or another control method might be based in artificial intelligence through rule-based control. Control methods correspond to classes, subclasses and families in Table 2. The choice of the control method is affected by the required accuracy and the computational constraints.

Similar to control configuration, the control method is chosen from experience. Usually, no analysis is done to substantiate the choice of a control method over another. In many cases, if the physical system's design is finalized and the control configuration is chosen, very limited choices remain for the control method. In fact, the choice of the control method and control configuration are highly coupled to the extent that some control engineers will not address them as two distinct decisions.

The emphasis here is that no systematic process exists for evaluating different control configurations and methods against each other. At best, these are chosen based on experience. The existence of the physical system in the final design phase greatly limits the control configuration and methods' feasible choices.

4.3.3 Controller

Once the control configuration and control method are chosen, and the physical system's design is finalized, control engineers will spend most of their effort choosing numeric values for the control architecture's parameters (e.g. gains, sampling time, ...etc). Mathematical representation of the controller is fully defined at this stage, and full dynamic simulations with higher fidelity models are possible. The control engineer design loop starts at this stage and ends with the decision of what particular values in the controller to use. Refer to Section 4.2. Sophisticated numerical

techniques exist for a full spectrum of controller types.

Note that the traditional controller design cycle covers only the choice of numerical values for the controller parameters. The design cycle assumes a presupposed control configuration and method.

4.4 The Feedback Control System - Revisited

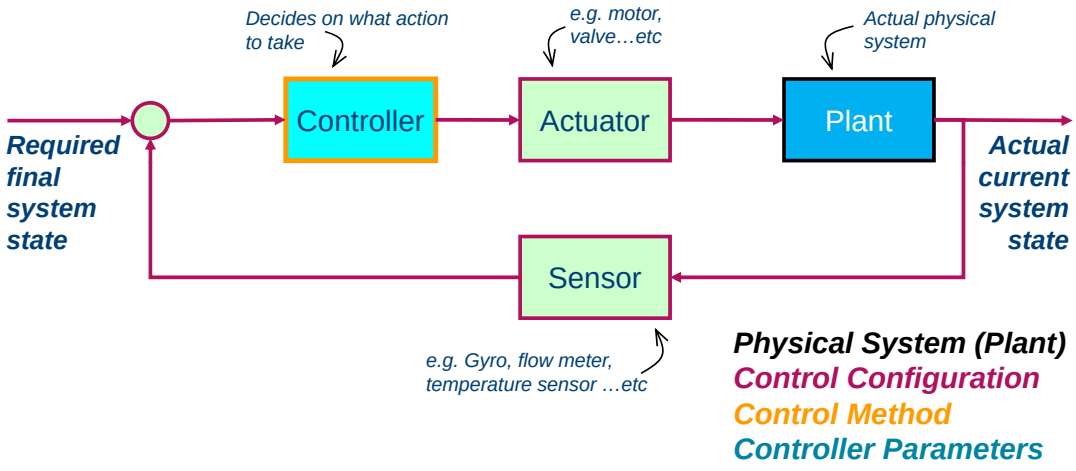


Figure 30: Feedback Control System with Three Basic Decisions

The feedback control system discussed in Section 2.1 is revisited here. Figure 30 shows the same plant / controller pair. As before, the sensor measures the plant's output, and passes this measurement through the feedback path to the control system. The control system then compares the required final system state with measurements from the sensor and issues a command action to the actuator, which in turn drives the plant.

Figure 30 also shows how the three control basic decisions are implemented in practice. The control configuration is the choice of actuators, sensors, feedback states, commanded control signals, plant controlled states, and finally the comparison criteria with the external required state. This is shown by the red line.

The control method is the orange frame around the controller. It represents

the algorithm the used to process the feedback signals and the external inputs, and compute an appropriate control command signal.

The controller is represented by the blue box. It is here where the parameters needed to define the control method are set. The traditional control design practices concentrate most of the efforts, time, and computational resources on computing the controller parameters, i.e. the numbers that go inside this box.

4.5 Hypothesis I: Intelligent Systems Design Methodology

Hypothesis I: Integrating a selection of steps from the Standard Control System Design Methodology early on, in the design cycle,and during the conceptual design phase of the Standard Systems Design Methodology enables:

Hypothesis I.a: More knowledge about the final control / decision making architecture and configuration to be learned by the end of the conceptual design phase

Hypothesis I.b: Insight into the limits on controller / decision making architecture and configuration and limits on performance in the final design.

Hypothesis I.c: Recommendations to the overall physical system design from an intelligence / control / decision making architecture perspective.

4.6 Distributed Intelligence Systems Proposed Design Methodology

The main objective behind developing a new design methodology (specific to distributed intelligence emergent systems) is to bring the intelligence or control architecture design upstream within the conceptual design phase. Section 4.1 introduced one of many variations on the conceptual design process. Section 4.2 presented a general all encompassing process for control architecture design, even though this full process is seldom used. The task in this section is combining both design processes into one hybrid distributed intelligence system design process. The idea is that relevant steps are included from both processes while exchanging key information at

predefined milestones.

The hybrid design process reveals information in three major areas:

- The effect of the control architecture on the physical system layout, configuration and components.
- The constraints which the physical system imposes on the control architecture with respect to performance and capability.
- Potential improvements in the matched pair of control architecture and physical system designs.

Starting with the conceptual design process (see Figure 31), the problem definition and objective definition steps are done for the overall system; physical system and control architecture. The design process temporarily splits into two paths at this point.

On one path, the physical system's conceptual design is proposed or synthesized, then modeled. Behavioral or inverse dynamics models are used at this stage. Along a second path, the physical system is further analyzed from a control architecture perspective and based on the earlier problem and objective definition. The physical system concept alternative is set at this point with enough top level details that enables the partial initial choice of a control architecture.

Next, the control architecture is addressed. The control configuration is proposed, followed by a control method. Appropriate analyses of both of these steps are done as shown in Section 4.2. A control architecture design loop is defined here, such that the output is a compatible pair of a control configuration choice and a control method choice. Note that this is not a fully defined control architecture (the actual controller design remains), but for the purposes of this section, the pair of control configuration and method will be referred to as such.

The control architecture choice is fedback to the systems design process. A concept analysis is done parallel to a control architecture analysis. In fact, this analysis is done on the same software behavioral model, which contains a layer defining the distributed intelligence (embodied in the control architecture choice) and other layers defining the rest of the physical system. This hybrid step is referred to as the feasibility and initial evaluation step. It will be covered in details in the following section.

If the model analysis yields unacceptable results, another iteration along the interconnecting design loop is done. It is possible to go one step lower and go through one or more iteration along the control architecture design loop. The output of the design process, so far, is **one** conceptual design alternative that is either acceptable or unacceptable based on some criteria.

The outer most loop is the conceptual design loop. It is the loop that spans the extents of the design space and produces different feasible design alternatives. The above steps are repeated for a different conceptual design alternative proposed or synthesized in the concept synthesis step.

Eventually, several feasible design alternatives are handed to the final concept selection step. This involves final evaluation of design alternatives based on capabilities of the overall system, as well as other criteria.

Note that the design methodology involved three loops:

1. Internal control architecture loop. Output is one control architecture alternative.

2. Mid concept analysis and feasibility loop. Output is a feasible design alternative compatible with the chosen control architecture.

3. Top concept generation loop. Output is a set of feasible designs with varying degrees of initial assessment values.

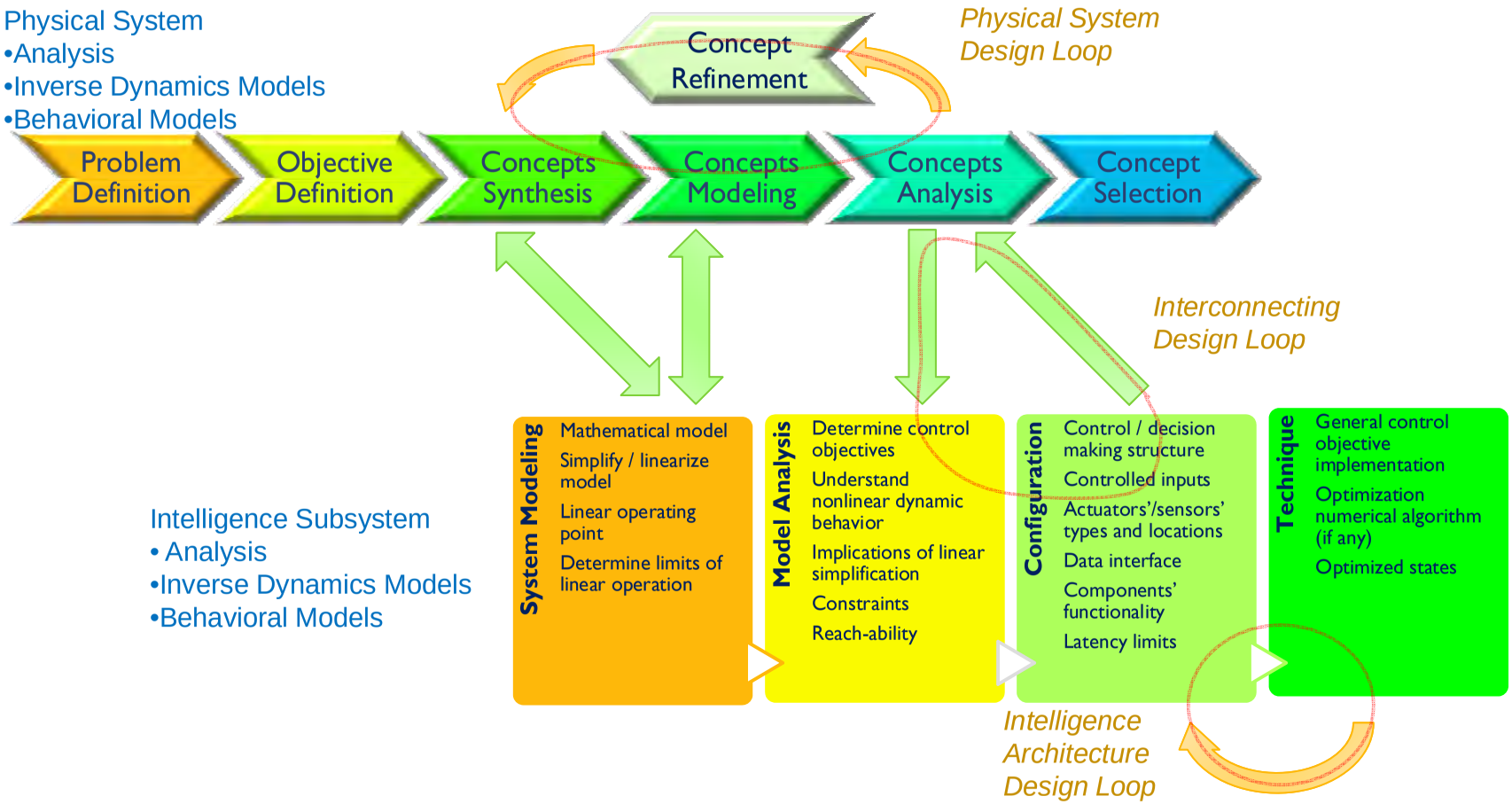


Figure 31: Distributed Intelligence Systems Proposed Design Methodology

4.7 Feasibility and Initial Assessment

The generation of a conceptual design does not mean that this design is feasible. This section proposes a feasibility and initial assessment approach for intelligent systems.

The feasibility analysis heavily depends on simulation based engineering. A model of the physical system is built, such that the behavior is modeled as accurately as possible. It is not important whether this model is a detailed dynamics focused model or a less accurate large scale model. It depends on the application and the type of information required. What's important is that it models the "behavior" of the system in a way that captures state interdependency, emergent behavior, vulnerabilities, constraints, i.e. the inherent system characteristics, characteristics that cannot be predicted with mathematical analysis alone. Therefore behavioral, inverse dynamics models are most suitable for this application.

Initial assessment also relies on the computer model. However, the information sought is different. Initial assessment of a feasible control architecture aims at determining the level (qualitatively or quantitatively) of satisfaction of system requirements. For example, how well can the control architecture / physical system combination able to achieve the system design requirements.

The function of the control architecture is to drive the system within the design requirements *range*, using a *finite resources*, avoiding any *instabilities*, and maintaining a standard level of *capability*. It accomplishes this function by processing inputs from a *sensor network* and applying control commands to an *actuator network*. The control architecture combined with the physical architecture form the system architecture. Hence there are four main functions any control architecture needs to carryout on the system architecture. Four different analyses are proposed, one in place of each function.

Figure 32 outlines the integrated design methodology, i.e.the control architecture and the physical system design process. It starts by taking the potential system

architecture as its input. System spatial and functional requirements are considered first. For example, will all the subsystems fit in the available assigned space for the overall system, and / or will the system weight be acceptable? This step is a feasibility step, meaning it is a pass / no pass step.

The second step is composed of four analyses conducted on the potential architecture corresponding to the control architecture functions discussed earlier. The four analyses exchange information with the two last steps, namely the actuator placement and sensor placement. Actuator and sensors positions are either handed over from the potential architecture definition or proposed within these two steps. If proposed, the actuator and sensor architecture has to be compatible with the potential design. The four analyses attempt at answering the following design questions:

1. Can the system reach the required states or capabilities? → *State Reachability*

2. What is required of the actuators, such that the system reaches its required final states? → *Inverse Dynamics*

3. Along the path to the final states, will the system become unstable? Can it be stabilized? → *Stability Requirements*

4. Along the path to the final states, will the system perform adequately? → *Capability Potential*

The system has to be represented by a physics based behavioral model for conducting any of the above analyses. The more detailed and accurate the behavioral model is, the more information delivered by these analyses. Moreover, not all analyses can be conducted for all types of systems, and not all analyses present the same level of information accuracy for the same system. The exception is the state reachability analysis which can be (and needs to be) conducted for any physical system.

The following sections consider each type of analysis in detail.

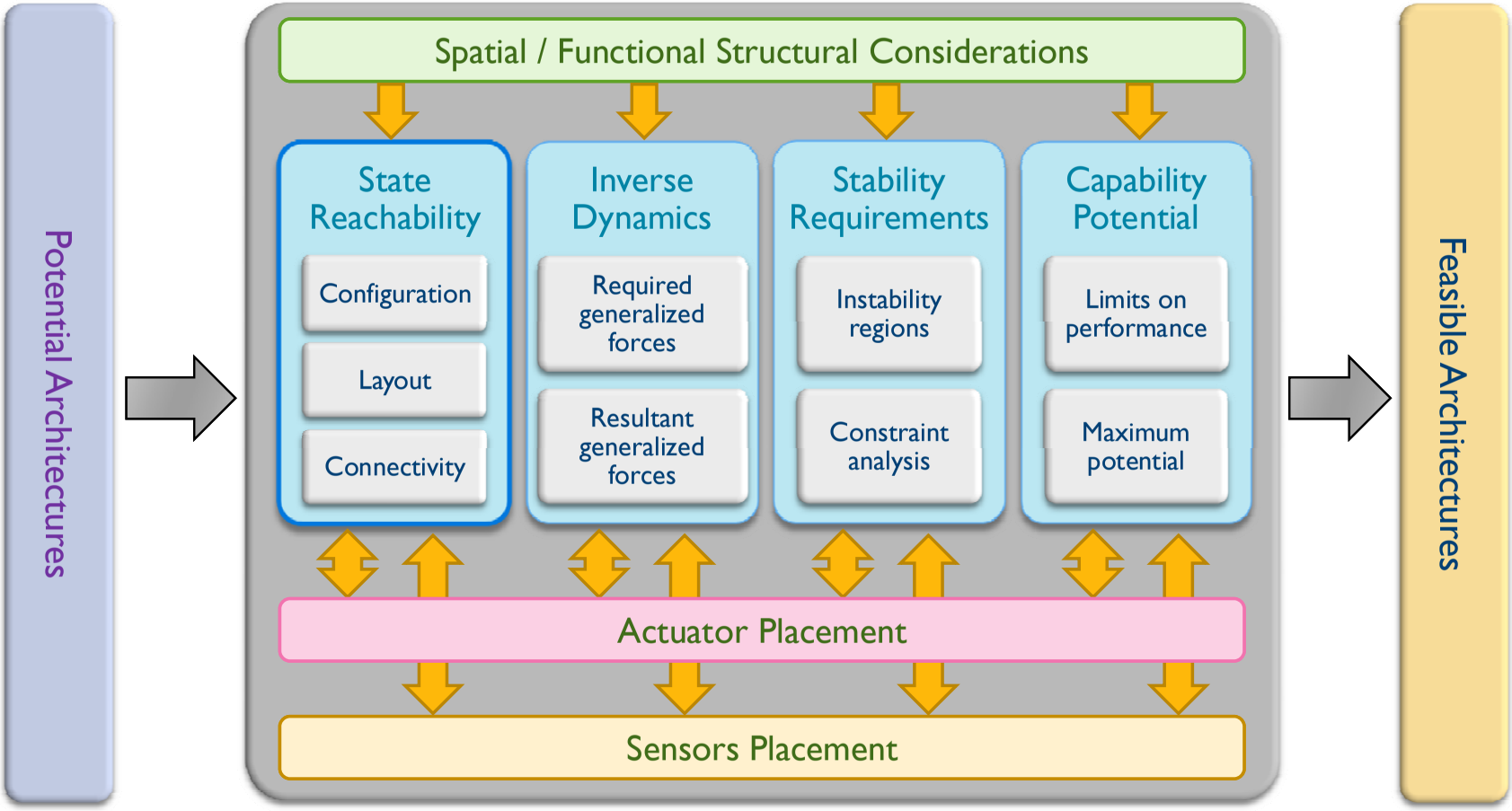


Figure 32: Feasibility and Initial Evaluation Tests

4.7.1 State Reachability Analysis

State reachability analysis aims at defining the range of the physical system states. In other words, the goal behind conducting this analysis is to study the extent of "how far the system can go". The concentration here is on the more desirable states, or states that define the performance of the system, and not on the full system's state set. For example, in a fluid network, if the objective of the network is to deliver a certain amount of flow to a service, then the desirable state is the mass (or volumetric) flow rate. Pressure is another computed state. Although it is essential for the proper system simulation, it is not one of the states that define the main system function.

There are several drivers that influence the system's reachability. The first is the physical system configuration, i.e. the system's components, their function, setup and behavior. The system's layout, or the structural arrangement of the system components - topology , is another reachability driver. Finally, the components connectivity and information exchanged on the connection links count as the final driver.

Reachability is influenced by two main factors:

1. The capacity of control authority the control architecture can deliver: The control architecture actuator network should have enough resources (routed along the physical system's components) and deliver enough driving forces to change and adequately manipulate the system's states. An example of this is an aileron system on an aircraft wing. If the required design criteria is that the aileron deflects at $x\ ^0/s$, the aileron actuator component has to be supplied by enough resources (hydraulic pressure or electrical current or ...) to move the aileron at this speed. Moreover, the control architecture has to able to maintain the deflection speed at maximum loading conditions, i.e. providing enough control authority.

2. The physical limitations of the system: If the physical system cannot make

it to the final required state because of inherent constraints , then it does not matter how well a control architecture is designed. It will not be able to achieve the system requirements. Building on the previous aileron example, consider a required maximum aileron deflection of y^0. If the aileron mechanism is limited to less than y^0, then there is no control system that is capable of achieving the design requirement. The control architecture is always limited by the potential in the physical system.

State reachability, unlike the following three analyses, is always performed in systems design, regardless whether the system has a control architecture or not. In some cases, it is referred to as "trajectory study" [12]. In fact, it is one of the main tasks in the traditional design process. The next three analyses are more related to the control architecture and its relationship to the physical system.

4.7.2 Inverse Dynamics Analysis

Many systems can be represented by a set of mathematical / logical equations, which describe the system states. For most engineering systems, the set of equations require generalized forces as their inputs and produce state trajectories as outputs. For instance, a force acting on a mass results in a certain velocity and acceleration of the mass. The force counts as the input to the governing equations set, which is solved for velocity and acceleration. This solution is referred to as the forward dynamics solution of system equations.

Inverse dynamics analysis is, as the name implies, the solution of the inverse problem. If a prescribed velocity and acceleration profiles are required from the above mass, inverse dynamics analysis attempts at calculating the driving force. This type of analysis is very popular in the area of robotics and biomechanics. One of the earliest works on inverse dynamics was in [3], where the problem of the control of a flexible link was solved.

The main assumption required to solve inverse dynamics problems is that transients do not affect the solution. In other words, transients are ignored. For many systems, specially systems with higher time constants such as mechanical systems, this is not a drastic assumption. A quasi-equilibrium is assumed at every time step within a simulation run, hence balance equations (energy, force, ...) can be solved for generalized forces.

Inverse dynamics analysis is considered a very effective tool in sizing systems. Building on the example described in Section 4.7.1, given an external aerodynamic moment on an electrically driven aileron, and for a prescribed aileron deflection, the analysis calculates the required motor torque, and in turn the electric system power demand.

Another major output of the inverse dynamics analysis is the generalized resultant forces, or the impact of the states change on the system. The prescribed motion of the above aileron is caused by the motor torque, which together with the aileron and mechanism, are attached to the aircraft wing. The resultant forces and moments that the aileron system exerts on the wing can be calculated as well as the driving forces.

4.7.3 Stability Requirements Analysis

Traditionally, stability is addressed through rigorous mathematical analyses to the equation set of dynamic systems. Accurate stability ranges are identified and stabilizing controllers are designed. Dynamics systems are, many times, linearized to be suitable for wide variety of stability analyses methods. Since stability is greatly affected by system parameters, such analyses is usually conducted after the physical system is designed.

But addressing stability in the conceptual design phase is not as straightforward since the system's parameters are not set yet. Hence, accurate stability properties are not available at this stage. However, there is much information about stability

that can be inferred about the system in the conceptual design phase. This is the objective of the the stability requirements analysis.

Similar to state reachability and inverse dynamics, stability analysis relies on the system behavioral model. In addition to the model, it is also necessary to have an elementary mathematical (or logical) representation of the system. Simulation scenarios that stimulate system states to change within their domain limits span the full range of operation of the system while the mathematical representation helps the designer understand the the system's behavior within the full range. The output of the stability analysis is not necessarily an accurate identification of the systems's stability limits, but it is a general understanding of the stability properties of the system.

Another output of the stability analysis is more related to the constraints and operational limits. Constraints arise from different sources, such as the system inherent characteristics, actuator power, mathematical model of the control architecture, among others. Physical constraints resulting from the system's properties, architecture, components or connectivity are the objective of stability analysis in this context. The control architecture has to conform to these constraints during operation while either avoiding instability regions or stabilizing the system within such regions.

Note that the definition of stability analysis, as defined in this section, is different from traditional stability methods employed in the field of control systems. While traditional stability aims at stabilizing a previously designed physical system, the current analysis tries to qualitatively describe instability regions and quantify the systems's physical constraints. The fact remains that traditional stability cannot be accurately addressed until the final systems parameters are set in place.

4.7.4 Capability Potential Analysis

System capability is usually defined as part of the systems requirements in the early design phase. Section 1.7.3 outlines the basic concept of capability and capability based design. The mathematical definition of a system capability depends on the performance metrics (also may be defined in the system's requirements). The system behavioral model, if properly designed and implemented, can predict many of the system's performance metrics enabling the calculation of required system capabilities.

The key enabler for the capability potential analysis is the computer simulation model. Modeling and simulation environments have been previously used for capability assessment of acquisition and life cycle management in defense systems (for example [138], [93], [132], [1] and [146]) and in concurrent engineering (see [88]). However, it is the assertion in this dissertation that modeling and simulation can also be used to assess the capabilities of dynamic large scale systems.

Some capabilities, such as effectiveness or survivability, require numerous number of simulation runs with different scenarios. System performance metrics are calculated for each run. The collective set of simulation results is used to predict the required capability. Also, note that not all capabilities can be predicted. Capabilities depending on system's state transients require more than a behavioral model (detailed dynamics model are usually required). Hence, the outlined analyses will not be possible unless such detailed model exists.

4.8 Summary

The objective of this section was to find a hybrid design process that includes both the physical system design and the control architecture design in the conceptual design phase. The traditional systems design process was outlined and explained. In addition, a typical control design process was outlined in full detail. Analysis of this process showed that there are three control architecture decisions, which were

identified as control configuration, control method, and controller.

Further discussion showed that the first two decisions can be included in the conceptual design phase, namely the control configuration and control method. A hybrid design process was created by merging the aforementioned design processes, such that critical information is exchanged at specific milestones. This resulted in several design loops that involved the control architecture as well as the physical system.

To assess the feasibility of a control architecture / physical system pair (and hence break the loop by proposing a design), four different analyses were developed. All four analyses affect the actuator and sensor networks. The four analyses were State Reachability (what the system can do), Inverse Dynamics (how much resources it costs), Stability (regions of stability and constraints) and finally the Capability Potential (how good of a system it is).

Hence, all tools were set in place for addressing the control architecture and the physical system designs, coupled, in the conceptual design phase.

CHAPTER V

INTELLIGENCE AND CONTROL ARCHITECTURE METAMODEL

5.1 Introduction

Architecture is the conceptual description of the structure and behavior of a system. Focusing on intelligence architectures, it is necessary and sufficient to set the structure/configuration and method/technique of a specific design to completely define the concept.

However, representation of architectures (structure and behavior) is not a standard process. There has been many attempts at standardization for specific domains [36] [91]), such as the Unified Modeling Language (UML) in the field if object oriented software engineering [23], Department of Defense Architecture Framework (DODAF) [101], and others. No architecture modeling language exists for intelligent systems.

Architecture is described by defining three basic concepts [87]:

1. Decomposition: what components are included in the system
2. Interface conformance: each component in the architecture must conform to its interface rules. This includes behavior constraints
3. Communication integrity: system components only interact per the rules specified by the architecture.

This chapter presents a novel approach to representing architecture concepts of intelligent systems. The focus on intelligent systems does not keep the proposed approach from being applicable to large scale emergent systems, and to autonomous systems.

The novel approach presented therein decomposes the intelligence architectural concept into two main members: the configuration or structure and the method or technique. This is in accordance with the novel control systems taxonomy presented in section 5.2.1. Intelligence or control configuration is covered in section 5.6 while the control methods representation is covered in section 5.7.

5.2 Control Systems Taxonomy

Taxonomy is the science, practice, principles or techniques of systematic classification within a given domain. Although it is a term primarily used within the realm of biology, its meaning or significance applies to any domain encompassing components that share similar and dissimilar traits, or in domains which rely on a hierarchical representation of their structure. Controls Engineering is not an exception. Controllers can be classified into groups and subgroups based on certain criteria, discussed later, that cover the controller design field.

System of systems design involves the systematic exploration of a given design space and the evaluation of a multitude of design alternatives. A specific design concept is chosen after many designs are compared. To generate such a pool of designs (out of which one is chosen), the designer has to have a method of parametrically representing these designs, and needs to explore different concepts which usually are very dissimilar. The task of generation of design alternatives can only be accomplished if a proper classification of system components is employed, and such components are placed in predefined organizational tree structures. Controller structure (or in a more general sense: Control and Decision Making Architecture) is an inherent component in Intelligent Systems. Hence, a systematic classification and categorization (taxonomy) of controllers is an essential enabler for intelligent systems design. Unfortunately, the traditional controller taxonomy does not help much.

Table 2: Typical Controller Matrix of Alternatives Based on a Traditional Taxonomy of Controllers

Controller Class	**Controller Subclass**	**Controller Family**
Simple Control	Proportional Integral Derivative	-
Adaptive Control	Gain Scheduling	Stability Optimized
		Gradient Optimized
	Model Reference Adaptive	Parametric
		Nonparametric
	Model Identification Adaptive	-
Optimal Control	Linear Quadratic Gaussian	Reduced Order
		Full Order
	Model Predictive	Centralized
		Decentralized
Robust Control	H^∞ Loop Shaping	-
	H^2 Minimization	-
Stochastic Control	Linear Quadratic Gaussian	-
	Robust Control Techniques	-
Intelligent Control	Neural Networks	Rule Based
	Bayesian	Optimized
	Fuzzy Logic	Artificial Intelligence
	Genetic Algorithms	...
	Agent Based	
Hierarchical Control	Networked Control	Centralized
		Decentralized

5.2.1 Traditional Control Systems Taxonomy

Traditionally, control systems are mainly viewed and categorized from two perspectives: the less common overall *control structure* perspective, and the more common the low level *controller* perspective. Yet, the current published literature does not present much detail on either perspectives. In fact, because of how controllers (and control systems in general) are designed, this subject seems less important for control engineers. Reasons for the sparse research on this subject are suggested in a later section. The subject of this section is to span different approaches on the taxonomy of control systems.

Controllers receive more attention, research, analysis, design and testing than the overall structure of the control system configuration. Controllers are classified based on the computational techniques they employ to produce a control command signal (which is applied to the actuators in the plant). How the controller "computes" and "behaves" is the main classification criteria, followed by less important criteria of how the controller (or the control system as a whole) is structured or interacts with the plant.

Surprisingly, the subject of controllers classification is hardly addressed in published literature. It is limited to classification approaches based on the numerical algorithm of the lower level controller, or to a top level control architecture

Table 2 presents a proposed taxonomy gathered from the literature, and based on the essence of the science of control systems: concentration on the controller computational algorithm. It can be regarded as an equivalent to a Matrix of Alternatives in system design terms, applied to controllers. Note that entries in table 2 by no means present a comprehensive set of controllers, it is a representative breakdown of the most common controller types used in current engineering systems. Parallel to the field of biology, controllers are grouped into classes, and further broken down into subclasses and families.

The first family of controllers is the simple control family. It has this name because no processing of the feedback signal takes place, the system state is only multiplied by a gain, its derivative by another gain and its integral by a third gain, and is fed back to calculate the error. All gains are constant real numbers, and are chosen to satisfy certain performance criteria. No feedback control system can take a form simpler than this.

Adaptive control, on the other hand, uses different values for the gains, depending on the system's surrounding conditions. Details on how to design an adaptive control system is beyond the scope of this text. However, an example helps introduce the idea behind this controller class. In a conventional aircraft, the gain on the channel corresponding to a complex pole in one flight condition, can be changed (in value) if this pole changes location during another flight condition. This allows for better overall system performance. The gain can have different preset values as in *Gain Scheduling* or on line calculation as in the *Model Reference* or *Model Identification* subclasses. The gain scheduling subclass can have a stability optimized set of gains or a gradient optimized set. The model reference subclass may employ parametric or a non parametric model. Hence, the objective of the controller class is to have a set of different gains to improve piecewise system performance, each subclass uses a different setup for the gain calculation, while each family within the subclass uses a different numerical approach to calculate these gains.

Optimal control is based on finding a control law (a set of gains) for a particular system such that a set of optimality conditions or constraints are satisfied. The optimal control problem contains a cost function, to be minimized, that is a function of the system states and control variables.

Robust control is focused on the uncertainty associated with the system; uncertainty in the system's model, in its inputs, in the applied disturbances. The uncertainty is prescribed and bounded within a compact set, and the objective of

the robust controller is to achieve stability and acceptable performance within this uncertain system domain.

Stochastic control assumes noise (with known probability distribution) on any of the measured states, and tries to find an optimal controller that minimizes a cost function in the presence of noise. Note that optimal control, robust control, adaptive control and simple control all have very similar structures from a configuration point of view; they all feedback states or their derivatives, and multiply them by gains.

Intelligent control does not necessarily solve matrix differential equations, nor other advanced computational methods to reach a control action (or control law). Most intelligent control subclasses use some sort of logic embedded in software modules to achieve a certain control action. Data used by these software modules can be obtained on line during real-time operation of the system, or off line using methods such as training and fine tuning, or both ways.

Hierarchical control deals more with the controller configuration, and how the control law propagates down the hierarchy from the top level overall system to lower level component controllers.

5.2.2 Inadequacy for Conceptual Design Phase

The difference between adaptive control and simple control is in how the gain is calculated. The simple control class has a constant gain calculated off line (during the controller design phase) to satisfy certain performance criteria. The adaptive control class substitutes the single gain with a set (continuous or discretized) of gains to satisfy a set of performance criteria in different system conditions. Hence from a controller design perspective, these classes are very different. On the other hand, both controllers will take the system state as their instantaneous input, multiply this state by a real number, calculate the error by comparing to the desired state, and based on this error apply a control input to the actuator. Therefore both controller

classes have the ***same*** architecture or configuration from a systems design perspective. Another example is given with the intelligent control class. From a systems engineer, a genetic algorithm using a set of states is different from another algorithm using a different set. The two different algorithms might have different software and hardware requirements, might have different physical limitations or constraints. On the other hand, the control engineer only sees a piece of computer software that runs on specific hardware, requiring certain inputs. What the system engineer vies as very dissimilar subsystems are viewed as similar subsystems by the control engineer. Akin arguments can be made for the other pairs of control classes according to table 2. This is the first major issue with such a taxonomy; a taxonomy based on the computational algorithm used by the controller. Controllers a dissimilar according to a control engineer, but similar according to a systems engineer, and vice versa.

Researchers, control engineers and scientists might agree or disagree with some components of this classification. For instance some might argue that model predictive control can be a class on its own due to its popularity (model predictive controllers and proportional integral derivative controllers are by far the most common controllers used in engineering). This is a very valid argument, but if this is the case, where does optimal control fall? It cannot be a subclass to model predictive control because model predictive control uses optimization techniques in calculation of gains. Model reference control is very similar to model predictive control, so does it fall under its class? But it is by definition an adaptive control technique which has little to do with model predictive control. It is clear that classification of controllers in a taxonomy based on the controller computational algorithm results is overlapping areas of applicability of controllers. As another example, some control engineers will choose to have robust control as a subclass to stochastic control, while others might see the opposite. Or that stochastic control is in fact optimal control used with noise. Network control is not necessarily hierarchical and can be completely distributed

within a flat configuration. On the other hand, model predictive control can be distributed over a network, or setup in a hierarchical configuration, or existing as a one central overall system controller.

The disagreement emphasizes the point of this section: there is no general consensus on how controllers are classified. This is the second major problem that hinders the usage of such a taxonomy (or any similar one) in distributed intelligent emergent systems' design. It prohibits the spanning of the control architecture design space during conceptual design.

The major reason why this is the case in the control society realm is that controllers are designed for an already finalized (or almost finalized) plant design. Very little change in the plant structure is allowed, parameters are set, and in some cases, the physical plant is already in the fabrication phase. This is beyond the the conceptual design phase presented in this dissertation. The plant structure or hierarchy dictates the control configuration, which (coupled with the control engineers' experience) in turn results in very limited choices, if any, for control methods. So not much improvement can be accomplished regarding the intelligent architecture. For most complex systems, this does not present a major issue. Yet, for the intelligent large scale systems under consideration, the intelligence attribute is inherent to the system's requirements and system's structure. Hence it cannot be put aside for later design, i.e. after the system is built.

The designer needs to pay more attention to the intelligence structure early on in the conceptual design phase. Hence the intelligence attribute or requirement should be considered from a structural point of view. Computing control law parameters (such as gains) or any numeric specific system definition parameters should be left to later design phases. In fact, these tasks should be avoided in the conceptual design phase, or else, the essential step of design space exploration will be substantially compromised, locking the designer within very limited intelligence architecture design

choices. The third major problem with using the current controller taxonomy is that it overlooks the intelligence architecture structure and concentrates more on how to compute control laws. Therefore it renders itself unuseful in the conceptual design phase of intelligent emergent systems.

As an interim summary, the three main issues that arise from using the current controller taxonomy are: 1) it allows for structurally similar controller classes to be different entries in the taxonomy, 2) in other cases different controller classes overlap and finally 3) it does not take the controller structure into consideration.

5.2.3 Novel Control Systems Taxonomy Approach

Rather than treat control architecture classification from two traditional perspectives, namely: mathematical approach regarding the controller and an arbitrary approach referring to the way control systems courses are arranged in educational institutions, control architectures taxonomy ought to be considered in the systems engineering context. Since, the overall motivation behind this research is to include control architecture design consideration in the conceptual design phase, it makes more sense to classify control architectures based on the anticipated decisions made during this phase.

Section 4.3 outlined the three main decisions a control engineer needs to make when designing a control architecture. These decisions are made regardless of the current design phase, but usually enter the design process at or after the detailed design phase (refer to Section 1.9). However, the hypothesis is that not all control design decisions need to wait for a detailed physical system design. The control configuration and the control method can be defined, in a general sense if not in detail, in the conceptual design phase.

The current control architecture design tradition highly couples the control configuration and control method concepts (Figure 29). This is in part due to how the

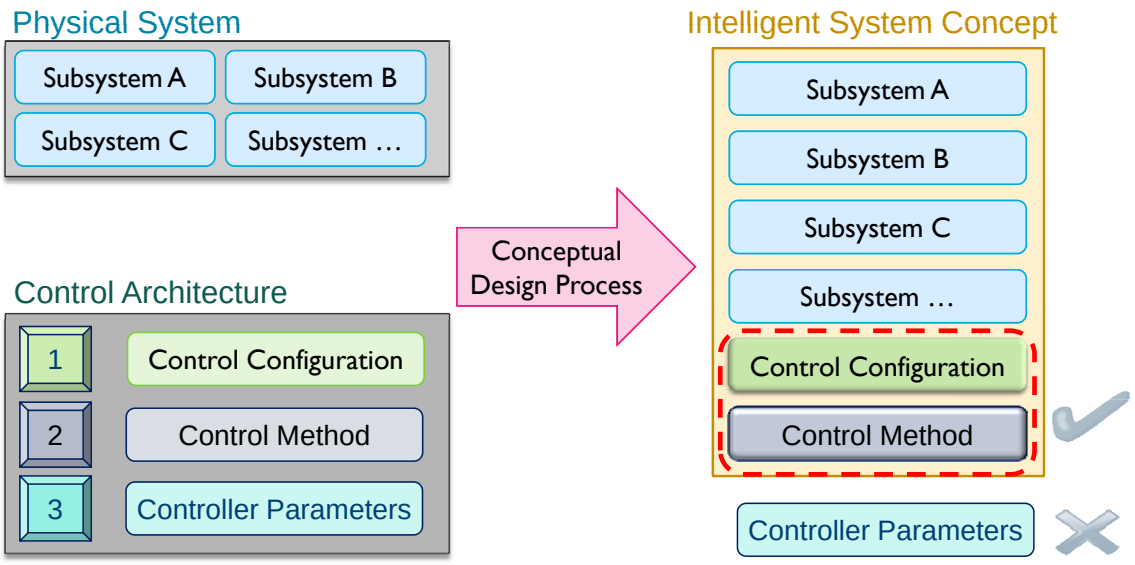


Figure 33: Three Major Control Architecture Decisions in Conceptual Design

traditional control systems taxonomy is arranged, frequently mixing between the two concepts. The first step in developing a novel taxonomy is to decouple (or at least relax the hard coupling) of these two concepts. Hence making it possible to treat each independently.

First, It will be assume that any control configurations can accommodate all control methods, and any control method can be implemented in all configurations. As an example, consider a hierarchical control configuration. PID and model predictive control methods can be implemented in such a configuration. Of course there are exceptions to this assumption arising from two different sources: 1) control architectures unsuitable for particular physical systems, and 2) control methods requiring specific control configurations. For instance, decentralized control of highly coupled subsystems is nearly impossible. Also, network control methods and centralized control configuration are by definition incompatible.

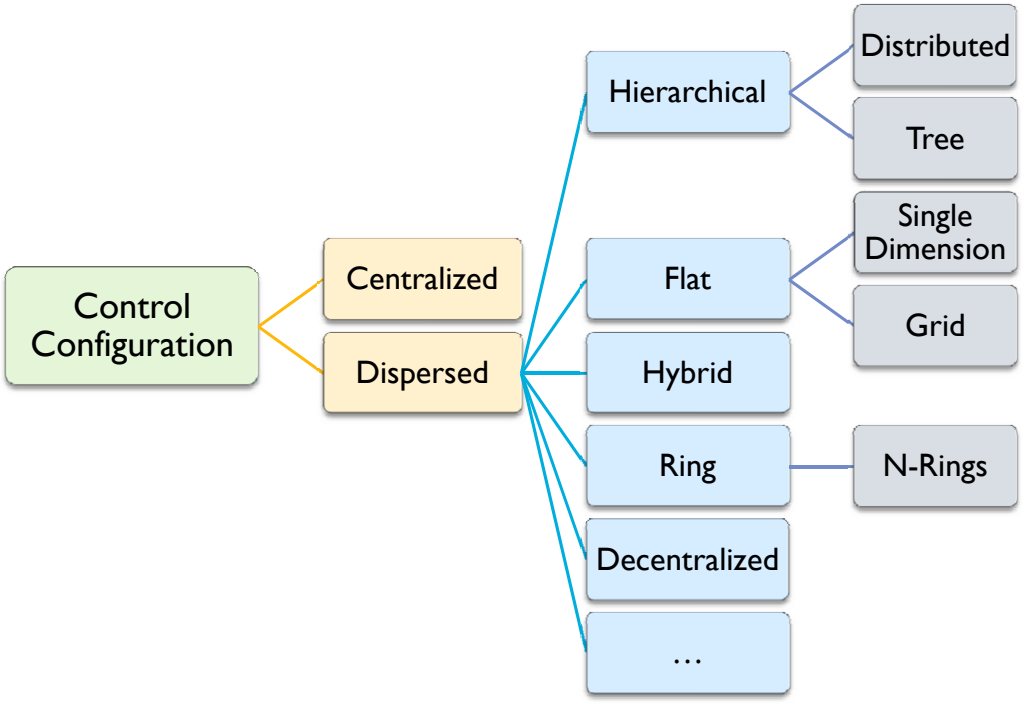


Figure 34: Novel Control Configuration Taxonomy

5.2.4 Novel Control Configurations Taxonomy

Next, the control configuration is examined from a structural or topological point of view. A control configuration taxonomy is proposed in Figure 34. The centralized control configuration is mutually exclusive to any dispersed control configuration, and stands alone in its own class. There can be no reduction in topology for centralized control because, by definition, it is composed of one controller. Centralized control configurations encompassing distributed control (refer to Section 2.2.4 and Figure 13) are assumed to be a special case, but do not stand in its own subclass or family.

Hierarchical configurations are ones with sub-controller multi-levels, and stands in its own subclass. It can exist in its pure form, or can be a simple tree (forming a family of the hierarchical subclass). Another family under the hierarchical configuration is distributed configurations, in which controllers on lower levels can have direct cross communication without reference to the parent sub-controller.

The next member in the dispersed configuration class is the flat control configuration subclass. It is characterized by having all controllers on the same level, with no controllers having a higher authority level, hence "flat". The flat subclass has two families. The single dimension family is one in which all sub-controllers are connected in a single one dimensional chain topology. Sub-controllers in the grid family are arranged in a two- (or n-) dimensional grid. Note that with the flat subclass, some sub-controllers can be designated as boundary nodes, at which a control signal terminates and is not relayed to the next sub-controller.

The ring subclass is similar to the flat subclass, with the only difference that a signal can be relayed between sub-controllers such that it ends where it originated. Ring control configurations can be a single ring, or multiple rings arranged in a sequential, intersecting, concentric or general fashion. The hybrid subclass is any control configuration that contains elements from more than one other control configuration subclass.

Note that the aforementioned taxonomy is not necessarily comprehensive, i.e. it does not span all control configurations or topologies in the current practice. Nevertheless, it is the proposition of the author that such a taxonomy is sufficient in covering most commonly used control architectures. Moreover, the novel control configuration taxonomy is easily expanded to other subclasses, if need arises.

5.2.5 Novel Control Methods Taxonomy

Instead of the topological approach adopted for control configuration classification, control methods are classified on an internal structural basis in addition to function accomplishment basis. All control methods (and/or sub-controllers) aim at driving the physical system to a final state, by considering the current state and the required final objective, and changing the control signals to the physical system. However, different types of control methods achieve this goal using a variety of pathways. The

proposed control methods taxonomy is based on the mechanisms these pathways are implemented such that the controller accomplishes its function.

The first member of control methods is the simple subclass. It includes classical control techniques such as PID control. LQR control is similar in internal structure as PID control, but differ in how the controller parameters are calculate off line. Hence it is considered as another family within the simple control subclass. Other variations on PID or LQR control can extend the simple subclass to to new families.

Optimized control methods form a second subclass. They are characterized by solving an optimization equation on line, i.e. in real time or during the controller execution not during the controller design. For example, model predictive control calculated a set of inputs to drive the system along a certain trajectory by minimizing a function, and this is done in line. Model predictive control and optimal control, among others, are families of the optimized subclass.

The intelligent subclass encompasses all methods with an internal structure that allows for some sort of higher level decision making or provide an internal learning mechanism. Also, it may encompass all control methods with embedded inference engines within sub-controllers. This subclass is very diverse, and includes the families of all artificial intelligence algorithms, all fuzzy control algorithms and neural network control.

Another subclass in the control methods class is agent based control. This subclass and its members are characterized by independent sub-controllers in the form of self contained software entities, that interact to achieve the control architecture objective. Both the intelligent subclass and the agent based subclass are software based. However, the agent based subclass differs from the intelligent subclass in the software structure since the former relies on agent interactions, while the latter is built on modules exchanging information.

Network control systems are classified in a separate subclass, since extra controller

components are required for communication issues handling. Properties of communication networks in control systems, such latency, throughput and bandwidth, affect response time, performance and stability of the entire physical system.

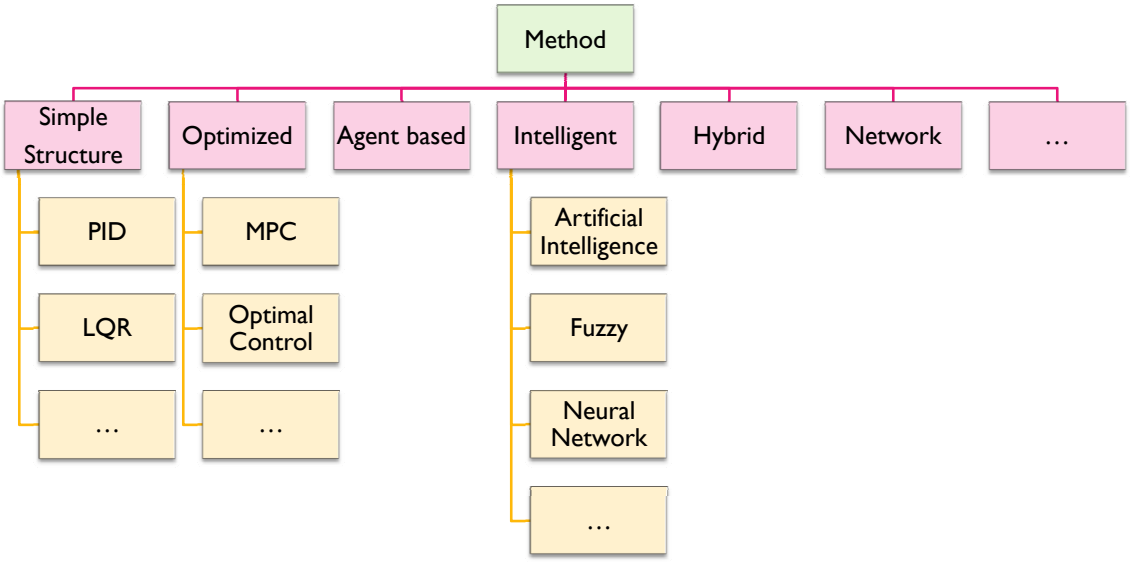


Figure 35: Novel Control Method Taxonomy

5.3 Control Architectures Meta-model

Metomodeling can be defined as the assembly or construction of a set of "concepts" within a specific domain of science or engineering. "Concepts" may refer to things, objects, procedures, or functions. A meta model is a level of abstraction higher than a library of objects abstraction, which in turn is another level higher of an actual model. Examples of metamodels are UML and SysML. The metamodel sets the rules and standards to build object libraries which are used to build models.

A control architecture can be realized in different ways. The most straight forward is to design the actual control system, set all its parameters, deploy it on the physical system, test it and confirm its functionality with respect to system requirements. This means that the physical system has to have been designed and manufactured, far from being in the conceptual design phase. Since the main objective is to address the

control architecture in the system conceptual design phase, the traditional approach

The current state of the art is to wait on the sy

5.4 Physical Subsystems vs. Control Framework Decomposition

To avoid confusion, it is necessary to point out the difference between physical system and intelligence framework decomposition. The physical system, as discussed Chapter 1, is the actual system that the designers are trying to add intelligence to. Decomposition of this physical system follows system design practices and is based in functional decomposition or spatial decomposition. The current traditional intelligent system design practices dictates that the decomposition of the physical system is done independently from the intelligence architecture design. This results in the intelligence system being designed after the physical system's design is finalized, as discussed earlier.

The new design methodology presented in Section 4.6 allows for the intelligence or control subsystem and the physical subsystem to be concurrently designed. Certain information are is exchanged at pre-specified design steps, including the system decomposition.

Initially, the physical system decomposition is done first without considering the intelligence architecture. Next, information about the system is used to analyze the intelligence architecture through a separate decomposition. The broken down intelligence architecture resulting from the second decomposition is used to update the original system decomposition. This continues in an iterative process until a satisfactory design is reached.

The physical system decomposition and the intelligence architecture decomposition are two separate but dependent tasks. These two tasks result in two different decompositions, one for the physical system and another for the intelligence architecture.

5.5 Intelligent Systems Features - Revisited

The proposed control architecture metamodel has to enable the following concepts:

1. *Scalability* is defined as the ability of the system to handle more tasks, or the ability of the system to enlarge in size or complexity [21]. In terms of control architecture, scalability is defined in the context of increasing the control architecture components (controllers, actuators, sensors, communication modules, processors,etc.), on different levels of the hierarchy, to handle more subsystems. In general, scalability is difficult to quantify and define even though it is frequently referred to in software systems [52]. There has been numerous attempts at presenting a rigorous definition of scalability, such as [22], [131] and [26]. "An architecture is scalable with respect to an IT profile and a range of desired capacities if it has a viable set of instantiation s over that range. Viable taken to mean with a linear (or sub-linear) increase in physical resource usage as capacity increases . . ." [26]

   Definition given by [131] is more appropriate for control architecture: "Scalability is the ability of a system to continue to meet its response time or throughput objectives as the demand for the software functions increases." [131] The control architecture is scalable if it is flexible enough to increase in size so as to handle more modules, while achieving its original design capability. In addition, scalability can be defined in terms of speedup. That is the increase in performance if the control architecture is extended with additional processor modules while the physical system remains unchanged.

2. *Modularity* is a software paradigm that stresses on the separation of the functionality of a program into independent, interchangeable modules. Each module contains everything necessary to execute only one aspect of the desired functionality. The collection of all modules may be placed in a single container,

such as a library or framework. A computer program, or in our case: a software based control architecture, is an assembly of some / all of the modules within a library. Modularity is essential in building complex applications [135].

The proposed metamodel has to be able to represent control architectures in the form of independent self contained modules that have a clearly defined function and interface. This provides the means for reusability of components. More importantly, the standardized representation also enables the consistent comparison between control architectures on a component (or sub-component) level. In addtion, the metamodel facilitate the modeling of modular networks (modules connected over a network such as their interface is optimized) in which supervised and unsupervised learning functions can be conducted on different parts of the network [130].

5.6 Proposed Control Configuration Metamodel

The control architecture configuration, sometimes referred to as the control architecture structure, is defined by the its included components, their interface standards and the properties of the interconnecting network. Due to current control systems design practices, control configuration is confused with control methods (such as agent based control). Control methods, on the other hand, refer to the algorithm that the control architecture will adopt to accomplish its required function. For example: distributed model predictive control is a control architecture which uses the model predictive control algorithms, but has a distributed structure.

In other cases, control configuration, control method and the actual controller are all put in one package and designed as one single entity in the overall complex system. This is acceptable with simple systems, or with more traditional well studied systems. In most cases, control system designers will conduct minimal studies on both the control configuration and method, while spend most of the design effort on

synthesizing the controller. Distributed intelligence emergent system pose a challenge for this approach.

It is not sufficient to simply state that the control configuration is distributed, decentralized or hierarchical. These are all indications of the overall structure of the control architecture is set. Detailed information is needed for a complete conceptual design specification.

To enable consistent quantitative and qualitative comparison of different control architectures, a standard framework for representation has to exist. There are two types of standard frameworks that are used in the literature. The first is standard architecture languages such as UML (section 3.4.1) or AADL (section 3.4.2). UML is too generic and is not specific to control systems. AADL is more suitable to control systems, yet it does not present guidelines to represent standard well known architectures. Although they can be used for control configuration representation, control methods are not directly supported.

The second type of frameworks are ones developed for particular physical system, keeping in mind a system specific control hierarchy. For example the real time control systems (RCS) by James Albus [7] and [8], or supervisory control [94]. It is worth mentioning that the former presents a generalized framework that is applicable to a wide varieties of physical systems, and stands as a significant step towards abstraction. The problem with such frameworks is that they do not present an sufficient level of abstraction, making most of them specific to particular control methods.

Hence, there is a need for an abstract representation framework, that is suitable for control architecture representation, and provides guidelines for standard control frameworks. In addition, it should allow for the representation of standard control methods. This section presents a proposed architecture framework that is used to represent two standard control configurations. The proposed architecture is graphical, with an AADL backbone. Section 5.7 presents a proposed abstract control method

representation.

5.6.1 Spatial Federation Framework

The first abstract control configuration framework to be presented is based on a spatial decomposition of both the physical system and the control architecture. Note that spatial decomposition is part of the conceptual design process is readily available at this point in design. It does not involve extra steps to be taken.

The components included in the control architecture have a one to one correspondence to the spatial zones within the physical system. The graphical representation of this approach is presented in figure 36. The components are arranged in a hierarchy that is customized to the physical system under consideration. There are several levels of control, hence can represent control architecture including top level intelligence or decision making and lower level hardware control. Lower level distributed intelligence can be modeled too using this abstract framework.

The control within a zone is further broken down based on function. For example a zone might have three subsystems, with a sub-controller for each subsystem. An internal control hierarchy can be present within the zone to coordinate these controllers.

However, not all intelligence architectures are hierarchical. The proposed spatial federation can represent flat architectures, by canceling all the controllers on the higher hierarchy, and retaining the inter-zonal coordinating controllers. In fact this particular abstraction can be used to model a control architecture that is a hybrid between architecture and flat architectures. Note that inter-zonal coordinating controllers are not necessarily in between spatially adjacent zones.

The graphical representation helps set the components and their connectivity. It does not define the interface nor the functionality of the components. This task is accomplished using the AADL representation of the framework. AADL representation

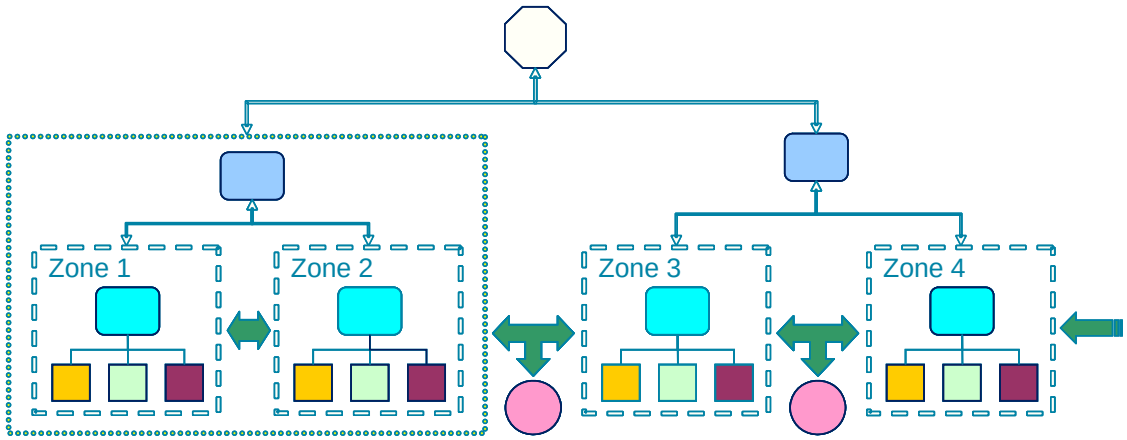


Figure 36: Spatial Federation Framework

is still an ongoing task and results are not shown therein.

5.6.2 Functional Federation Framework

The functional federation framework is similar to the spatial federation framework, although it is based on a functional decomposition. The conceptual design process will almost always have functional decomposition data as a result of system requirements analysis and hence this proposed framework requires no extra steps in the design process. It is shown in figure 37.

It is worth noting that a functional decomposition is more common among current control architectures of complex system. The reason being that subsystem controllers are designed by subsystem domain experts, then the set of controllers are integrated into one control architecture.

5.7 Proposed Abstract Control Method

The second step in standardizing intelligence architectures is to address the control methods. As discussed in Section 5.2.1, the current state of the art control taxonomy mainly relies on the computational algorithms used to calculate the control law. Problems arise from such a categorization principle (Section 5.2.2. To avoid these

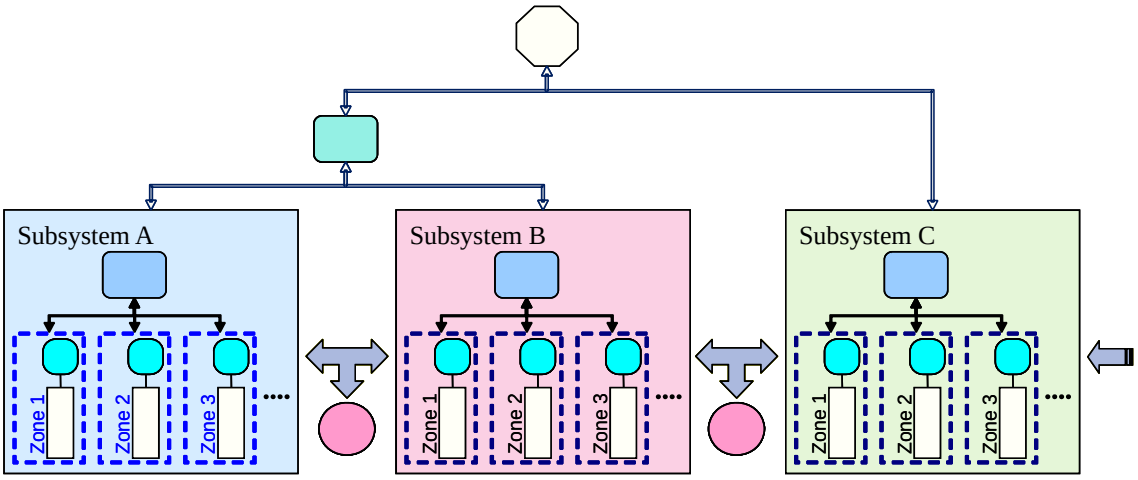


Figure 37: Functional Federation Framework

problems, abstraction should be used to bypass the computational algorithms and concentrate more on the control method structure, as explained in Section 5.2.3.

All control methods share similar components, in addition to method specific components. In fact, some control methods are structurally identical, and only differ in their computation algorithm. As far as a systems engineer is concerned during conceptual design (or even a control engineer participating on the team at this level), the structure of the controller counts more than the details of the computation. Hence, it is possible and beneficial to abstract control or intelligence methods into a generalized structure that standardizes control methods representation and manipulation. Several general control architectures are present int he literature, such as the Real-time Control System presented in [7], but are do not accommodate for different control methods. A higher level of abstraction is needed to accomplish this task.

Figure 38 shows a proposed structure for an abstract control node or module based on a functional decomposition. Note that this representation is developed for the a general control method not for a specific computational approach. For example, this architecture is able to represent a PID controller as well as a MPC controller architecture. The advantage is having the same form of representation,

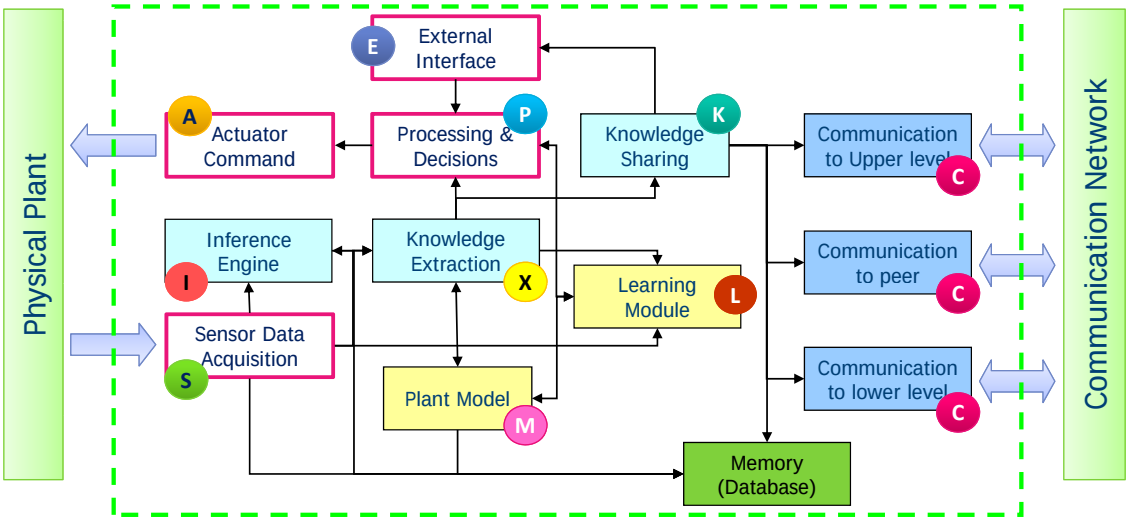


Figure 38: Proposed Generalized Control Node

hence common basis for comparison and evaluation.

5.7.1 Essential Modules

The control node has two main groups of inputs/outputs. The first is the interface to the physical plant through which the node receives measurements, and applies commands or control laws. The second is the interface with other nodes through the communication network. At least one of these two sets of interfaces need to be present for the control node not to be isolated. In some cases, only an interface to the plant is needed as is with the case of centralized control. In other cases, only the interface with other nodes is needed as with the case of hierarchical control.

Similar to all control architectures, there has to be a module that conveys the external commands of how the final system state should be to the actual controller. This is represented by the *External Interface* module. It is processes the external required state in the form of a command. The simplest form of this module is a unity gain which means that the external input is directly the required state. Other forms can be simple settings to the controller, or transformations from high level user inputs to specific numerical values applied to the control node.

The second basic module is the *Sensor Data Acquisition* module. It's responsibility is linking to the physical system or plant to acquire measurement data, and process these measurements. Sensor models should be included in this module. For example, a temperature sensor might have some transitional dynamics or a can be modeled as a simple gain.

The third and last basic module is the *Command Generation Module*. It processes data from the sensor acquisition module and external interface module to generate a control law that is applied to the plant. A simple proportional controller method is represented as a simple gain in this module. Higher order controllers can also be included in this module. The actuator models or dynamics (such as an electric motor) are also part of this module.

The external interface, sensor data acquisition and command generation modules are basic and essential to any control system, and hence are essential to any intelligence architecture. An intelligence architecture has be able to receive external commands, have situational awareness in the form of meaningful data and process both to generate a suitable form of action.

5.7.2 Non Essential Modules

In many cases, some direct measurements are missing through the sensors. Hence, some control methods employ an *Inference Engine* just after the sensor data to estimate the required missing states. Inference algorithms vary considerably, but all use the available sensor data as their main input. Measured data and inferred states are used to extract meaningful information in the *Knowledge Extraction* module. This module can include all forms of regression, data mining or similar methods.

Several control methods, such as model predictive control and model based control, require a software model of the physical plant to enable their optimization algorithms. Other control algorithm might require the plant model to estimate current system

states, avoid failure states and plan for future system setting to result in a required performance. This function is accomplished by the *Plant Model*. It links to all the previous modules as needed.

Intelligence architectures require learning mechanisms in some cases. This function is modeled in the *Learning Module*. It can be based in neural networks, Bayesian networks or any similar algorithm. It receives input from the the knowledge extraction module, the sensors module and the current control decision. It produces output that is used to generate future control decisions.

The core of intelligence is modeled in the *Decision Making* module. This module can represent any within the full spectrum of intelligence approaches from rule based fuzzy control to evolutionary artificial intelligence methods. It receives the information from the knowledge extraction and up stream modules, taking the external interface commands into consideration, and processes these signals to determine an appropriate course of action. It then applies its output to the command generation module to generate a control law command signal(s) to the plant, and to the learning module to aid in better future decision making.

Connections through missing modules from a certain control method are simply short circuited. For example, all modules but the basic three disappear from simple PID control. Adaptive control requires the extra decision making module to choose between the different sets of gains that are applied to the command generation.

This node stands as a single control entity in the whole intelligence architecture. It is possible that it is the only control entity such as in centralized simple intelligence architecture. Yet it is primarily developed for large scale intelligent systems that are usually characterized by distributed, decentralized, or hierarchical architectures. Hence the generalized control node is able to link to a communication network through three different modules. The *Communication to Peer* module handles communication to similar nodes on the same level of hierarchy. The *Communication to*

Upper Level module handles communication up the intelligence hierarchy while the the *Communication to Lower Level* module handles communication down the intelligence hierarchy. Decisions regarding which information is sent to which destination are handled by the *Knowledge Sharing Decisions* module. The communication modules' arrangement enables this general node structure to be a member of a multi level intelligence architecture, together with flat distributed architectures.

A subset of all information exchanged within the node can be stored in a central repository or database. This is represented by the *Memeory* module. It receives information from the knowledge sharing decision module, from the plant software model and from the sensor data. It can be a fully shared database or can partially share specific information with certain modules. Although it might be required for certain control methods or intelligence architecture, it does not affect the evaluation of different architectures against each other.

The proposed node architecture provides a standard method of representing the following information about the intelligence architecture.

1. The component modules present in a single node
2. The computational complexity and structural complexity of each component within the node
3. The connectivity of the components within the single node
4. The data requirements on the inter-components connections within the node

This enables a common basis of comparison of different control/intelligence architectures from different designs. Hence evaluation of these architectures is consistent.

5.8 Hypothesis II.b

The generic architecture building blocks or components are capable of representing different classes of common control methods

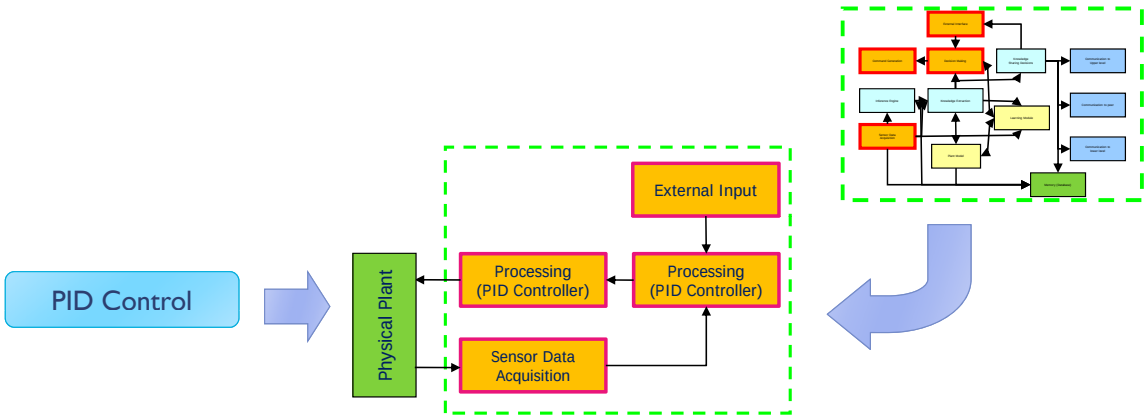


Figure 39: PID Control Method Representation

The general abstraction of the control node presented in section 5.7 can be used in realizing the structure of different control methods. All modules not applicable to a the control method under consideration are ignored and any connections passing through are short circuited.

5.8.1 Experiment II.b.1 - PID Control

PID control is the simplest control method available with feedback control systems. It relies on acquiring some measurements from the plant, combining these measurements with external inputs then multiplying the measurement (and possibly its derivatives and integral) by a constant gain. The output is applied to the actuators connected to the physical plant. All other modules are not required. Note that the three essential modules corresponding to the three basic functions cannot be ignored. The result is shown in figure 39.

5.8.2 Experiment II.b.2 - Intelligent Decentralized PID / Adaptive Control

This non standard control method is a hybrid between a PID controller applied to part of the plant and an adaptive controller applied to the rest of the plant. Both controllers constitute the lower level of the intelligence hierarchy, and are connected to a top level intelligent controller that employs rule based methods. The representation

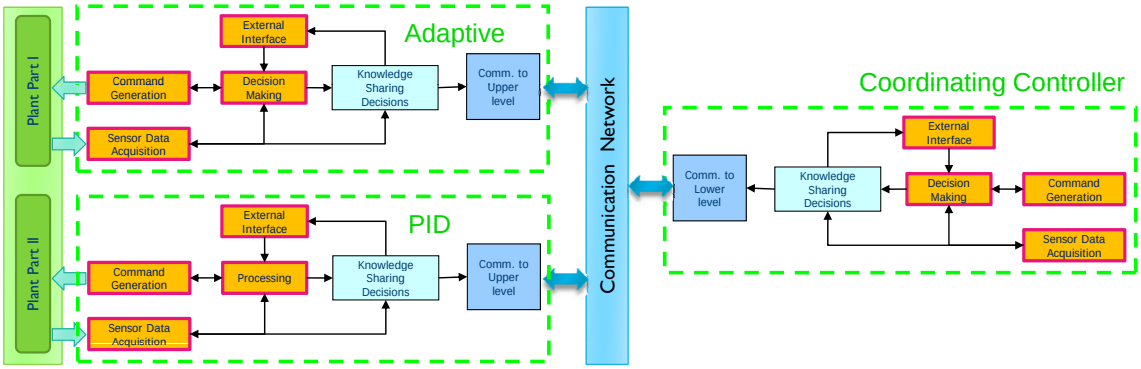


Figure 40: PID/Adaptive Intelligent Hierarchical Control Method Representation

is shown in figure 40.

CHAPTER VI

BENCHMARK PROBLEM: MECHANISM CONTROL ARCHITECTURE

6.1 Controlled Robotic Mechanism Design Approach

Robotic manipulators or mechanisms are an integral part of many large scale systems. Often such systems employ sophisticated control layers to coordinate the different components in it, making the control architecture an essential subsystem, and a major player in the system capability. However, such systems are almost always designed by first addressing the physical system design (actuators, links, hinges, ...etc.), finalizing the physical system design (dimensions, actuators, material, ...etc.), eventually followed by a detail design of the control architecture and its components. The objective of this demonstration is to show that the two step design as described above is not always necessary. In fact, the two step design approach may result in a suboptimal system. On the other hand, a comprehensive design approach of the both the physical mechanism components and the control architecture elements results in more information available to the designer. It is important to note that minimal resources are spent thus far to reach an optimal design.

The design exercise goal is two fold: to demonstrate the difference between the traditional design approach and the alternate approach of considering the control architecture in the conceptual design, and to show that proposed design suite provides more information about the control architecture early on in the design cycle. In reaching these goals, the effect of the physical system design on the control architecture choice will be explored. In addition, the effect of the control architecture structure

on the overall capability and performance limits of the full system will be quantified. Finally, the ramification of degradation of system (physical system and control architecture) components is also compare across different control architectures.

6.1.0.1 Design Problem Statement

It is required to design a manipulating robotic mechanism that is able to move from an initial point A to a final point B in the vertical plane. This should be accomplished within a certain accuracy, speed and for a pre-stated range of motion. The mechanism tip should be able to move on a path that follows a user set trajectory. The mechanism should be able to handle specific loads. It is preferred that the mechanism design be as light as possible, and does not occupy a large space. This particular problem has several features that render it particularly attractive for the demonstration of the control architecture design and evaluation suite. Firstly, it is a common design problem in the area of robotics. It has been addressed in numerous texts such as [95],[122] or [128]. The actual physical system options of this problem are established, extensively studied and well understood. Secondly, the kinematic and kinetic equation systems of most manipulating mechanism configurations are either easy to derive, or published in the literature. Many of these equation systems have an analytical solution. Thirdly, the design problem can be approached at different levels of fidelity. Computer simulation models can be built for such a physical system at different levels of fidelity, ranging from a lower fidelity large scale models treating components as single simulation units to highly accurate models representing the internal states of various system components. Lastly, although a manipulator mechanism can have as many degrees of freedom (as required), allowing for a host of various control architectures associated with the physical design. This means that different control architectures can be designed, studied, compared and evaluated. Although

that this system is a small scale system (relatively easy system to design), the following analysis will not make use of this feature. The system is treated as a large scale system with no assumptions made regarding its complexity and size. Also, the focus will be more on the control architecture design suite, and less on the physical design of the system.

6.1.1 Traditional Approach

The actual design process of the physical system is beyond the scope of this dissertation. Emphasis is more on the relationship between the control architecture and the physical system, and on the This refers to the structures, material, detailed mechanical design manufacturing processes, ... etc. It is assumed that two physical system concepts are proposed by the design team. Both, theoretically, achieve all the design requirements as stated above. The two proposed systems are shown in Figure 41 and Figure 42, which are described in the following sections. If a traditional design process approach is followed, one of the two systems will be chosen based on several selection criteria, including but not limited to, cost, capability, reliability among others. However no attention will be given to the control architecture selection criteria. This will be addressed after a particular physical system design option is chosen. For example if design A (Figure 41) is selected, a packaged solution of matching control architecture and method is put in place (without real design space exploration of the control architecture). A centralized control architecture is a straight forward design solution and if possible, is preferred by most control engineers. This is accompanied by a control method such as PID control. This is followed by the calculation of controller parameters (such as the gains in a PID control method), in which most of the controller design effort is spent.

It is to be noted that the relatively straight forward choice of a control architecture for such system is only possible because of the inherent physical system lack of

complexity, originating from small scale. This feature - the small scale resulting in reduced complexity - does not hold when the system increases in scale or complexity. For example, if the above proposed mechanism design problem is exchanged with a naval vessel design problem, control architecture design may not be as straightforward. Therefore, for the rest of the analysis, the information inferred from the small system scale feature should not, and will not be used in any conclusions.

The traditional approach fails to detect several issues with the design as will be shown. In fact, most of the constraints on the control architecture and method are set by the initial choice of the physical system, before the control architecture is addressed. The control architecture design suite analysis below call attention to these constraints.

6.1.2 Physical Design A - Four Link Design

The first proposed physical design consists of four links joined by rotational frictionless joints. Two of the links are connected to inertial points, forming a closed-loop mechanism. this mechanism contains four rigid bodies and five joints. Each rigid body can be represented by three equations of motion, therefore the whole mechanism can be modeled with 12 equations of motion. Each joint has three associated constraints, giving a total of ten the constraints. Hence, the number of degrees of freedom of the system is equal to 12 equations minus 10 constraints resulting in 2 degrees of freedom. This is sufficient to move the mechanism tip from an initial to a final point in the two dimensional vertical plane. A diagram depicting this design option is shown in Figure 41. The kinematic equations this configuration are described in [39] while [42] presents a detailed design approach for such a configuration. Reference [**?**] presents a variation of the four bar mechanism. The governing kinematic equations of the physical configuration shown in Figure 41 are given by [39]:

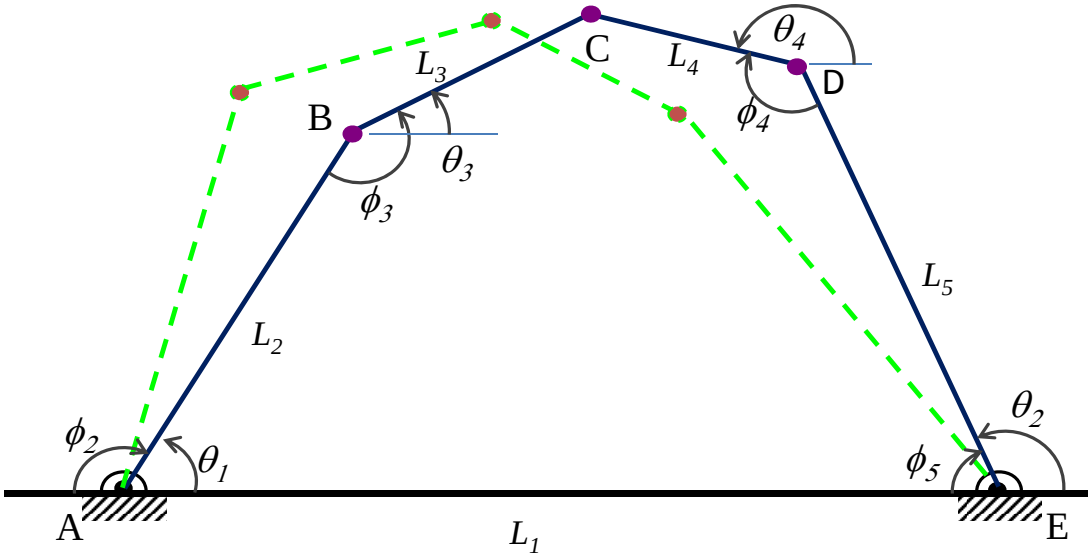


Figure 41: Design A Diagram - Four Link Mechanism

The position of the upper bar can be determined from

$$\phi_3 = \arccos \frac{K_1}{K_2} + \arccos \frac{K_3}{K_2 K_4} \tag{8}$$

where

$$K_1 = M_1 + M_3 \cos\phi_2 - M_4 \cos(\phi_2 - \phi_5)$$

$$K_2 = \sqrt{M_6 + M_{10} \cos\phi_2 - M_{11} \cos\phi_5 - M_{12} \cos(\phi_2 - \phi_5)}$$

$$K_3 = M_{19} + M_{20} \cos\phi_2 - M_{21} \cos\phi_5 - M_{22} \cos(\phi_2 - \phi_5)$$

$$K_4 = L_3$$

such that

$$M_1 = L_2$$

$$M_2 = M_7 = M_8 = M_9 = M_{14} = \ldots = M_{18} = 0$$

$$M_3 = L_1$$

$$M_4 = M_5 = L_5$$

$$M_6 = {L_1}^2 + {L_2}^2 + {L_5}^2$$

$$M_{10} = 2L_1L_2$$

$$M_{11} = 2L_1L_5$$

$$M_{12} = M_{13} = 2L_2L_5$$

$$M_{19} = \frac{1}{2}\left({L_1}^2 + {L_2}^2 + {L_3}^2 - {L_4}^2 + {L_5}^2\right)$$

$$M_{20} = L_1L_2$$

$$M_{21} = L_1L_5$$

$$M_{22} = M_{23} = L_2L_5$$

The position of the other upper bar is given by

$$\phi_4 = \arccos\frac{K_5}{K_2} + \arccos\frac{K_6}{K_2K_7} \tag{9}$$

where

$$K_5 = N_1 - N_3\cos\phi_5 - N_4\cos(\phi_2 - \phi_5)$$

$$K_6 = N_6 + N_7\cos\phi_2 - N_8\cos\phi_5 - N_9\cos(\phi_2 - \phi_5)$$

$$K_7 = L_4$$

such that

$$N_1 = L_5$$

$$N_2 = 0$$

$$N_3 = L_1$$

$$N_4 = N_5 = L_2$$

$$N_6 = \frac{1}{2}\left({L_1}^2 + {L_2}^2 - {L_3}^2 + {L_4}^2 + {L_5}^2\right)$$

$$N_7 = L_1 L_2$$

$$N_8 = L_1 L_5$$

$$N_9 = N_{10} = L_2 L_5$$

Any closed loop mechanism is associated with a set of different constraints. These constraints introduce another level of complexity on the control architecture. the control system has to make sure that the trajectory of the mechanism does not violate these constraints. Hence a separate module for processing the constraints should be implemented in the control architecture. This in turn increases the computational requirements of this control architecture. The constraints are given by:

$$\begin{aligned} L_1 \cos\theta_1 + L_3 \cos\theta_3 - L_2 \cos\theta_2 - L_4 \cos\theta_4 = L_1 \\ L_1 \sin\theta_1 + L_3 \sin\theta_3 - L_2 \sin\theta_2 - L_4 \sin\theta_4 = 0 \end{aligned} \tag{10}$$

As a simplification, the two outer bars are assumed to have the same length, so do the inner upper bars. Note that the initial position of the the two outer bars is vertical.

6.1.3 Physical Design B - Two Link Design

The second proposed physical design option is shown in Figure 42. It consists of two bars joined by a frictionless rotational joint. The bottom bar is connected to an inertial point. This configuration is very common in the area of robotics and has been used multiple times in many applications [12]. The research paper [79] attempts at designing the software architecture for a six degree of freedom manipulator, similar

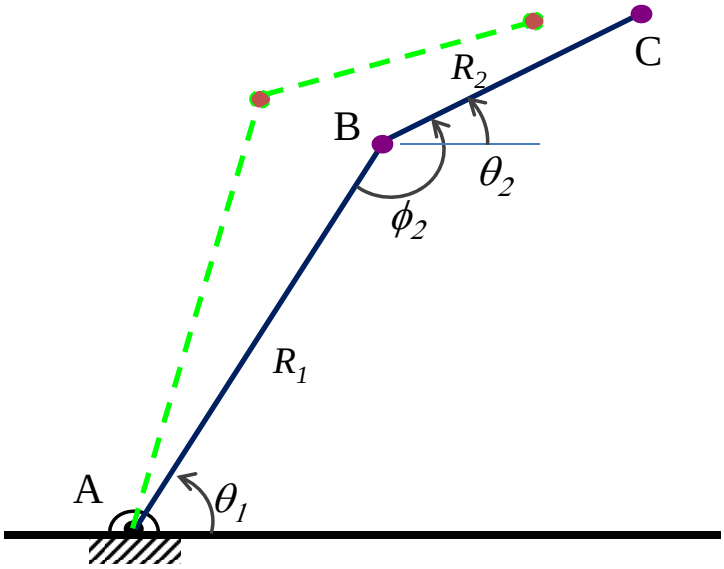


Figure 42: Design B Diagram - Two Link Mechanism

to the one considered in this section. The mechanism has two rigid bodies with three equations describing the motion of each, and two joints associated with two constraints. Hence Design B has 2 the degrees of freedom, similar to Design A. The equations of motion are relatively straight forward and are given by:

$$x = R_1 \cos\theta_1 + R_2 \cos\theta_2 \tag{11}$$

$$y = R_1 \sin\theta_1 + R_2 \sin\theta_2 \tag{12}$$

The two bar design differs drastically from the four bar design. While the closed loop four bar design has inherent constraints that restricts the motion (by restricting the range of values of the two independent variables), the two bar design is constraint free. Both independent variables describing its motion have no kinematic constraints that limit their ranges. This fact is very useful in controller design, because a controller does not have to take the any constraints into consideration.

6.2 Control Architecture Analysis Suite

The main goals behind applying the analysis suite to control architectures are twofold: exploring the limitations that the physical system imposes on the control architecture design, and identifying the effects of the control architecture on the physical system performance. The four main analyses presented in Chapter 4 are discussed in the current section.

6.2.1 State Reachability Analysis

Reachability analysis aims at defining the limits of operation of the system, i.e., how far can the system go. this is simple to understand when it comes to mechanisms. The kinematics equations for design any and design be are known and presented in the earlier analysis. These equations where to put in a Matlab computer program to check the expense of mechanism operation. The computer code is given in Appendix A. The range of each mechanism is shown in Figures 43 and 44, together with the nominal configuration of each mechanism. Note that both figures are drawn to the same scale.

6.2.2 Inverse Dynamics Analysis

The objective of this analysis exercise is to determine the required resources to operate either mechanism. For such a design problem, the required resources are the amount off electric power supply to the mortal moving the links. In addition, the electric motors' torques and their resultant torques are also calculated.

To conduct the inverse dynamics analysis, a path is assumed such that each mechanism is required to move along that path. The path schematic is given in Figure 45. It consists on five segments (six way points). The segments are as follows: from top left to top middle, from top middle to bottom middle, from bottom middle to top right, from top right to top middle, and back to top left. The path spans the extents design A's reachable limits, and is chosen so as to have a fair comparison

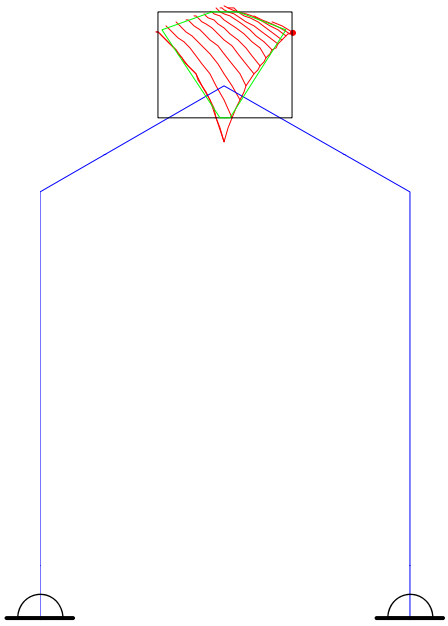


Figure 43: Design A Nominal Configuration - Reachability

with design B. Note that design B is capable of reaching a wider range of points, yet since the objective of is to compare both designs, then a common required path needs to be implemented on both mechanisms. In addition, the mechanism is required to maintain its position for one second at each of four internal waypoints.

Two separate inverse dynamics models are built for both the mechanism designs. The SimMechanics Simulink toolbox is used, since it provides much functionality and enables the programmer to concentrate more on the physics of the problem rather than the mathematical formulation.

The simulation model for design A is shown in Figure 46. The path way points are transformed into their corresponding angles on the two sidebars. These angles are used as inputs to the model rather than using the way points themselves. this technique avoids building a separate controller module. Inverse dynamics assumes that controllers are ideal. Also, the model assumes no control in between the way points. Only the error in between the current point and that is the destination way point is used to drive the mechanism. Ideally a control layer that controls the path

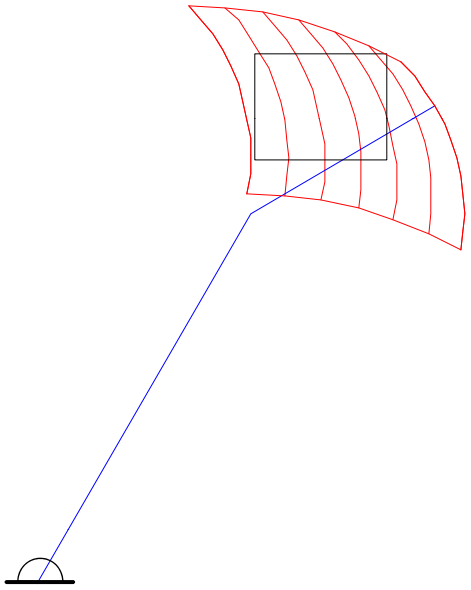


Figure 44: Design B Nominal Configuration - Reachability

in addition to the final points should be used.

Figure 46 shows the top level representation of design A. At this level, the model processes the inputs and gathers the outputs. The actual inverse dynamics are modeled one level deeper and are shown in figure 47. The mass and physical properties of the mechanism are inputed in this level. It is important to note that SimMechanics models the actual actual components of the mechanism using a physical representation, not merely using mathematical constructs. This allows the modeling of dynamical systems a more physics focused approach rather than a mathematical approach.

The simulation model for design B is shown in Figure 48. Similar to the design A model, the design B inverse dynamics model is built using SimMechanics. The top level model in Figure 48 manipulates the inputs and outputs. Figure 49 shows the internal dynamics of the two bar mechanism.

Inverse dynamics, in mechanisms context, is calculating the forces and torques that can drive a given motion profile. The motion profile is defined by calculating the bar angles corresponding to the initial and final way points. Universal joints are

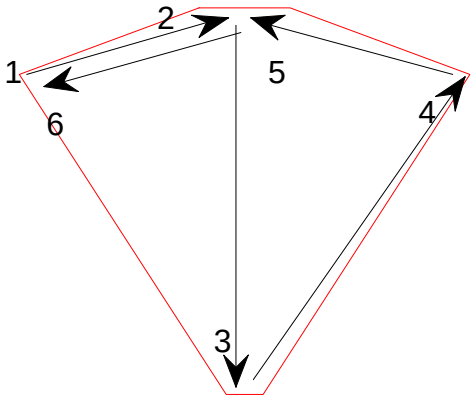


Figure 45: Mechanisms A & B Required Path

placed in between the bar masses in the design A and design B models (see Figures 47 and 49). Constraints are placed on the joints such that they only rotate around a single axis (the z-axis in this case, since the mechanisms are planar). Each joint is connected to a "joint actuator" which commands a joint rotation angle. The angle, the angular velocity, reaction force and torque, and the driving torque in each joint (which is equal to the motor torque) are measured.

The two actuators (electric motors) driving the joints are assumed to have a maximum speed of 10 ^{o}/sec and a maximum acceleration of 20 ^{o}/sec^2. The electric motor is assumed to accelerate with maximum acceleration to reach its maximum speed, maintain this maximum speed until a different commanded angle is received, at which time it decelerates or accelerates depending on the value of the angle, again to reach its maximum speed.

A simple model of the motor is given by

$$T = i \cdot K_t \tag{13}$$

$$P = v \cdot i \tag{14}$$

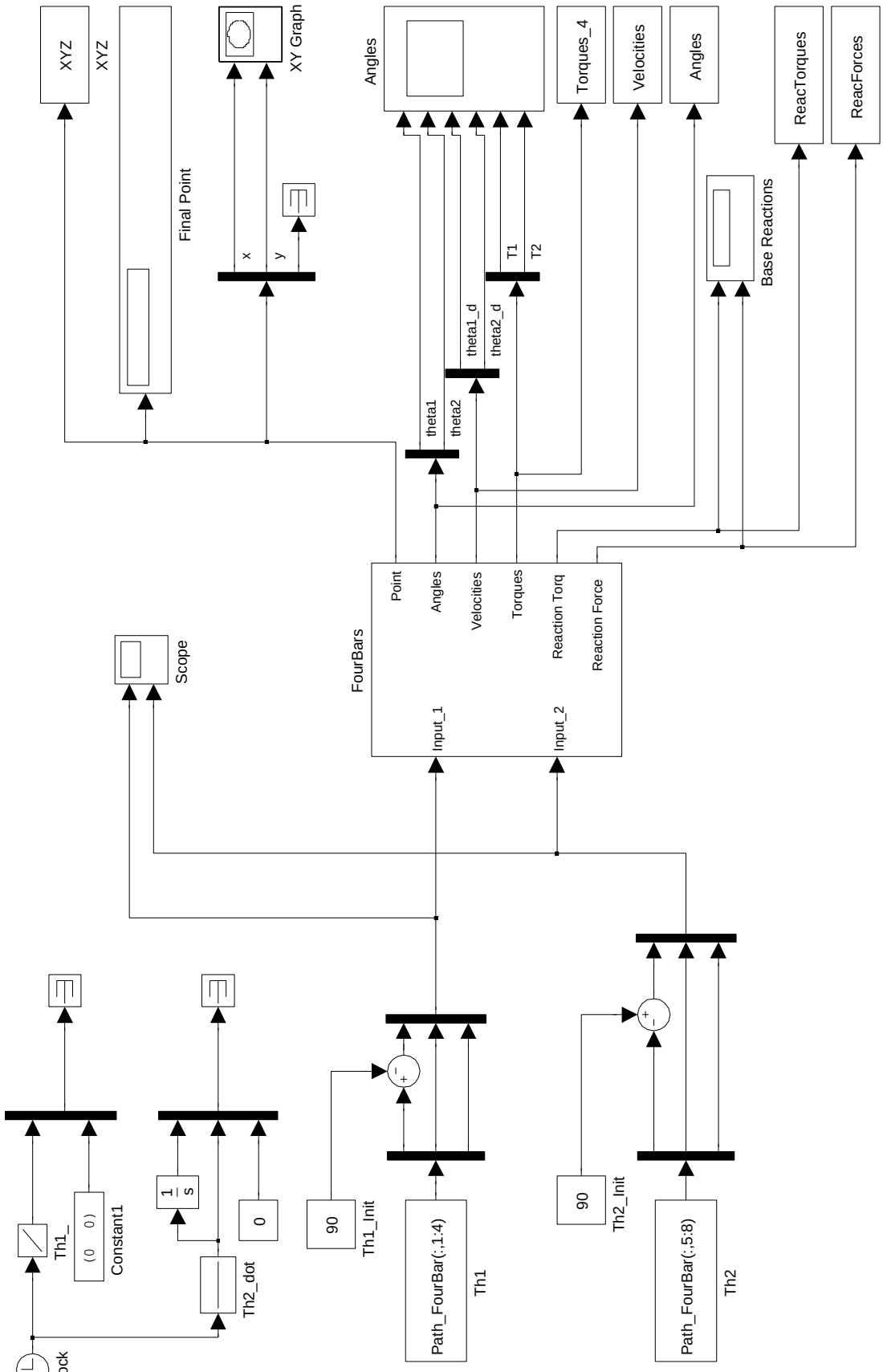


Figure 46: Design A Inverse Dynamics Model - Top Level

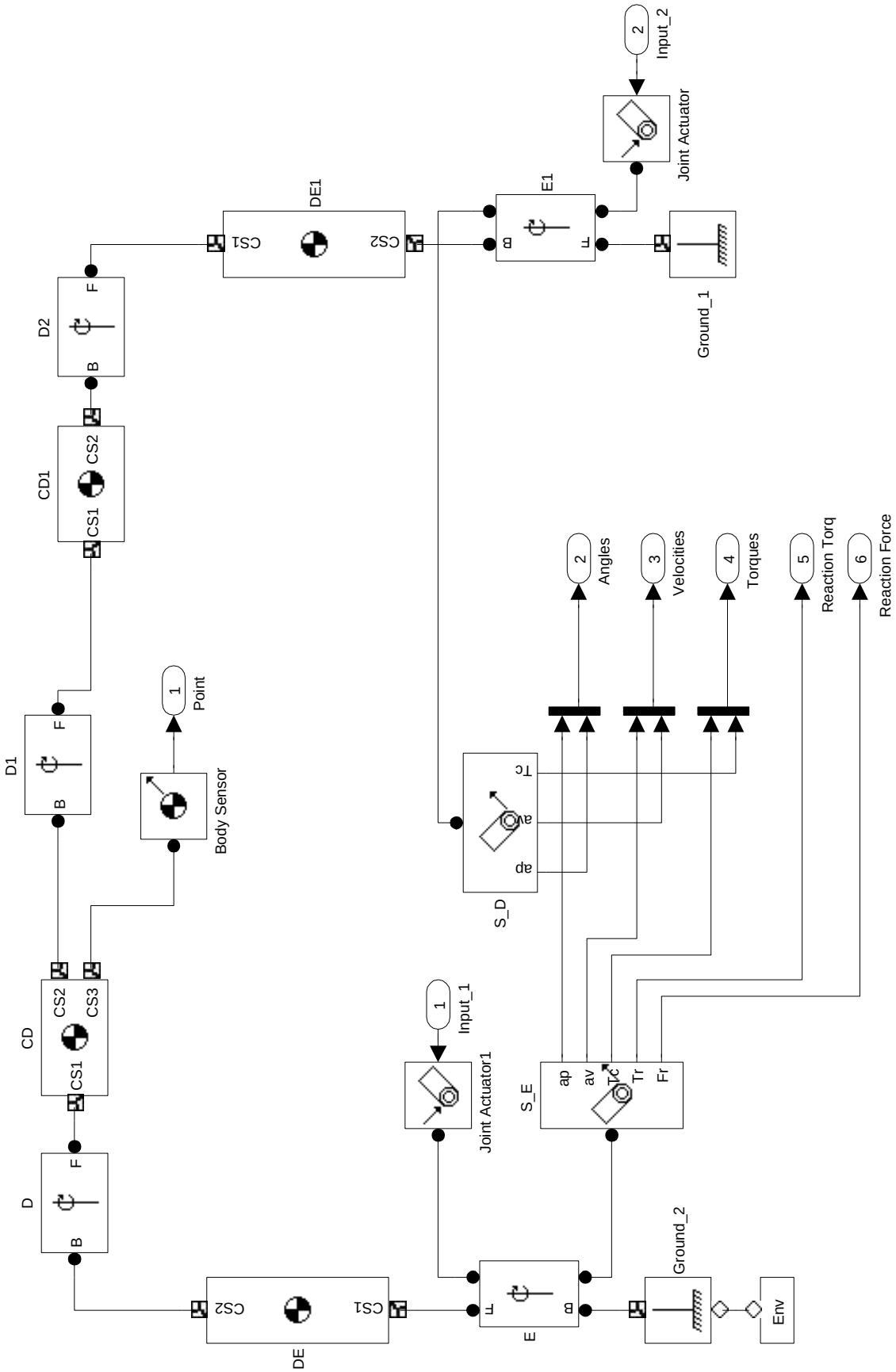


Figure 47: Design A Inverse Dynamics Model - Internal Structure

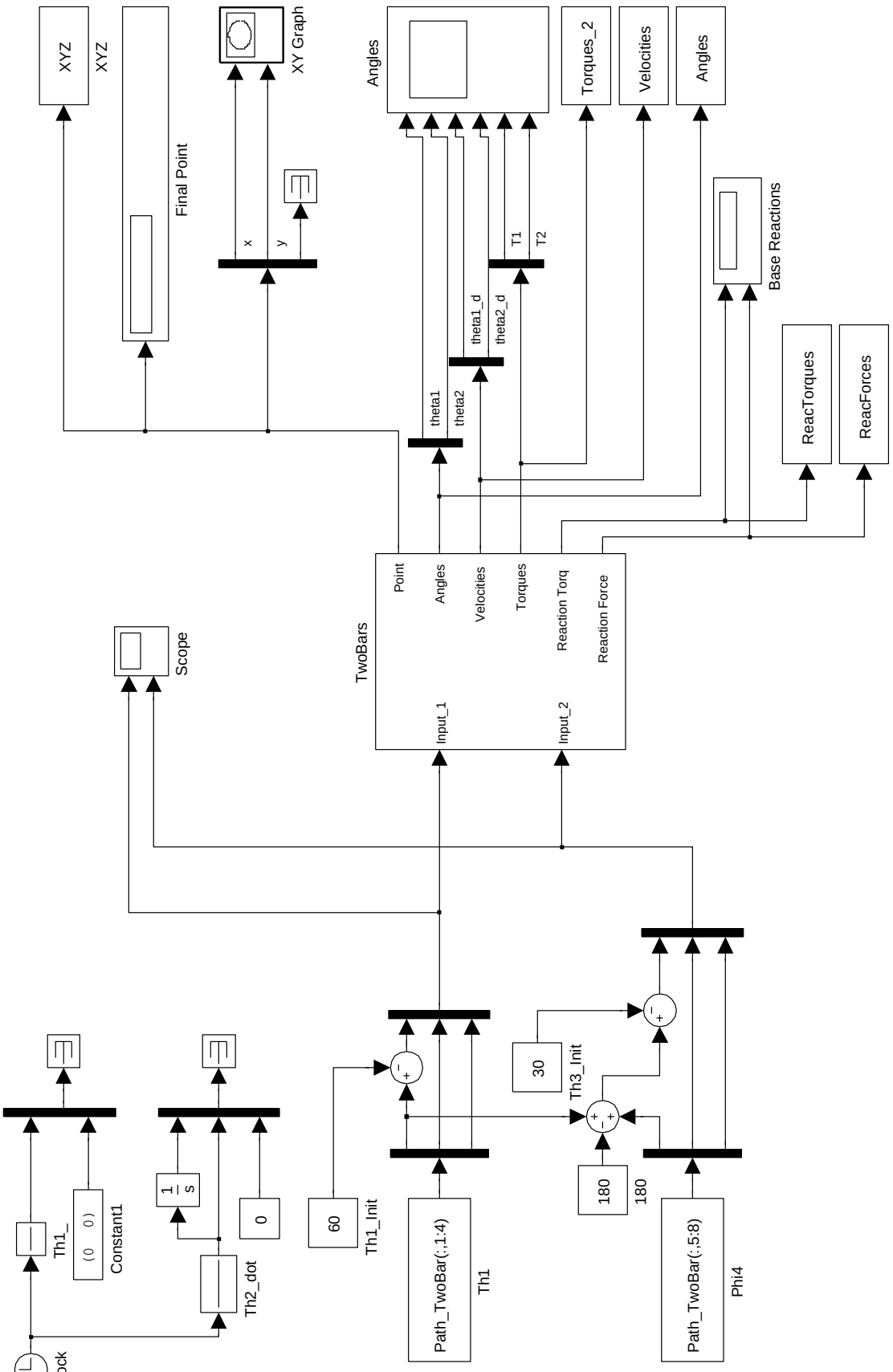


Figure 48: Design B Inverse Dynamics Model - Top Level

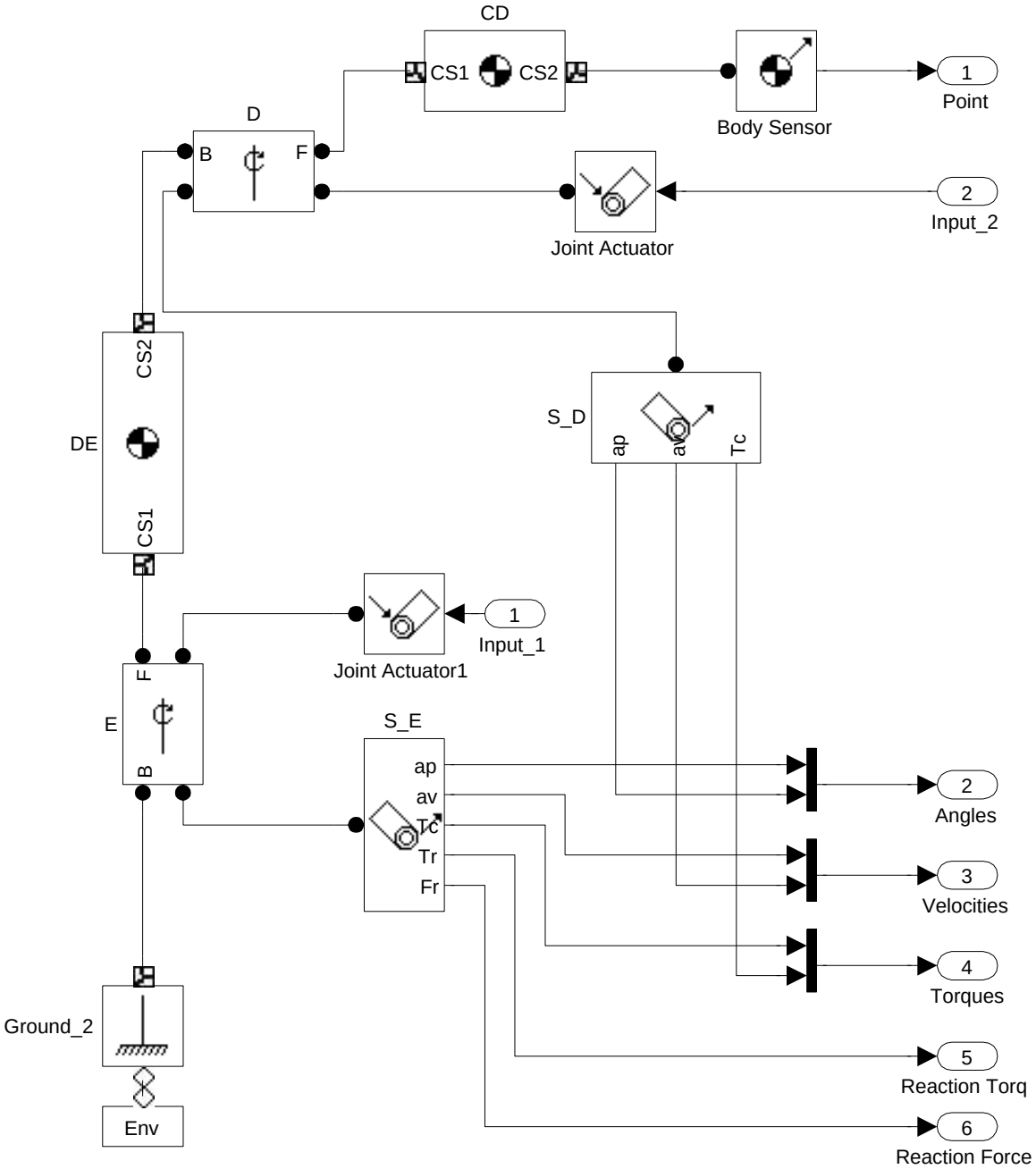


Figure 49: Design B Inverse Dynamics Model - Internal Structure

where

T : Electric motor torque

P : Electric motor power consumption

i : Electric motor current

v : Electric motor voltage

K_t : Electric motor torque constant

The energy E consumed by the electric motor can be calculated by integrating the electric power:

$$E = \int P\,dt \;=\; v/K_t \int T\,dt \;=\; C \int T\,dt \tag{15}$$

where C is a constant with appropriate units. The last equation assumes a perfectly efficient electric motor.

Each motor is assumed to have a weight of 4 lbs. In case of design A, the motor weight does not affect any results since both motors are mounted on the the two bottom hinges. For design B, one of the motors is mounted at the bottom hinge, while the other is mounted on the lower bar to drive the at the upper hinge. This puts an additional load on the bottom motor since it has to carry the weight of the whole mechanism in addition to the other motor.

Next, a measure of performance and another for efficient operation of each mechanism is discussed.

6.2.2.1 Inverse Dynamics - Performance

One of the measures of mechanisms performance can be taken as the time to complete the preprogrammed path. Both mechanisms are required to complete the path shown in Figure 45. Only way points are provided for the system to operate.

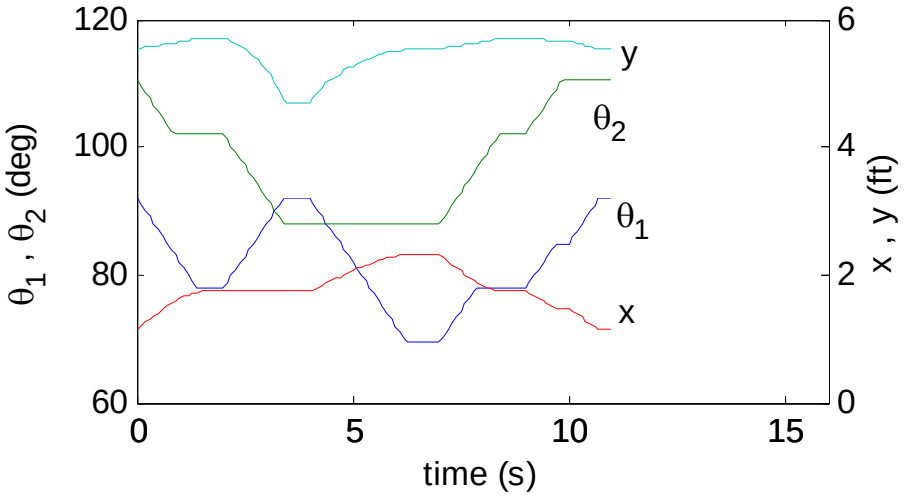


Figure 50: Design A Inverse Dynamics Analysis - Performance - State History

Design A's (the four bar design) states trajectory is shown in Figure 50. The motors are mounted on the two outer bars, hence driving θ_1 and θ_2. It is clear from the plot that there are time periods during which one of the two angles is constant, while the other is moving to reach its next way point. The bar angles plots are smooth, and the angles seem to move from one way point to another without much jitter. Similarly, the tip position coordinates, x and y, have smooth trajectories. Note that the mechanism ends where it starts, at way point 1. Note that the angles as well as the point coordinates remain within Design A's reachable range.

Figure 51 shows the x and y coordinates plotted on Cartesian axes. The thick line defines the path, while the thin lines define the mechanism's tip coordinates. Arrows show the tip's direction of motion. The actual path tries to closely follow the required one, but never crosses it due to imposing the reachability constraint.

The total time it takes Design A to complete the path is a little less than 11 seconds.

Observing Figure 52, the case with Design B is different. It does not have any constraints on its motion, hence one of the angles, namely ϕ_4 spans a wide range

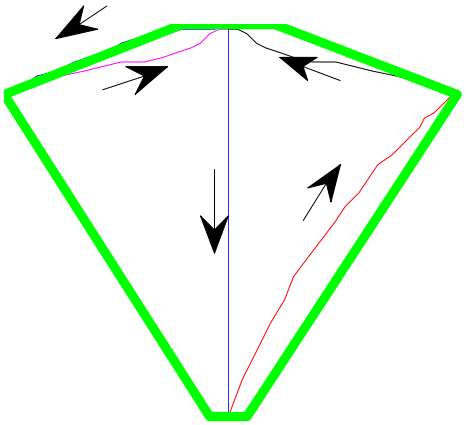


Figure 51: Design A Inverse Dynamics Analysis - Performance - Path

of values. On the other hand, θ_1 does not, and remains constant for much of the simulation time on between way points. θ_1 and ϕ_4 (or θ_2) are completely decoupled. The coordinates of mechanism tip, x and y, are also plotted in Figure 52. They both take a larger range of values if compared to design A's tip coordinates. In other words, design B is moving unnecessarily "all over the place". This is also clear in Figure 53.

It takes design B about 16 seconds to complete the required path. It is well noting that imposing a simple state value (angles values) constraints on the motion of design A resulted in an improved performance. On the other hand, since design B was left unconstrained, the mechanism angles were free to take any values resulting in degraded performance.

In addition, if the mechanism is required to move along a given path (not only to satisfy way points), the control architecture has to take into account the state interdependency of design A. This adds another level of complexity to the control method, beyond the simplicity of the state value constraints. Imposing the same requirement on design B does not involve the increase in complexity. The control

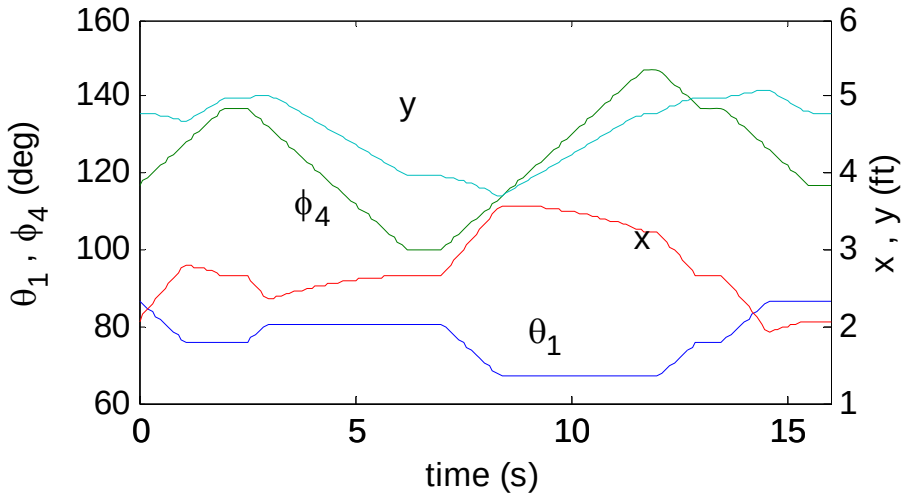


Figure 52: Design B Inverse Dynamics Analysis - Performance - State History

architecture does not need to consider any interdependency constraints among states since they do not exist.

6.2.2.2 Inverse Dynamics - Efficient Operation

Design A has the two motors mounted on the two outer bars, while design B has the two motors mounted to the bottom and top bars, such that the bottom bar carries its weight. Figure 54 shows the electric motor torques.

The bottom two torque curves correspond to design A. Both torques change over a large range of values. They are (almost) equal when the mechanism is moving along the second segment, i.e. from top to bottom along the path's line of symmetry. Notice the spikes in the torque curve occurring at instants when the bar angular is changing.

Considering design B, the torque on the upper motor remains within a narrow range, the reason being that the upper bar is close to horizontal over most of the path. The torque on the lower motor is always greater then the upper motor's torque. Also, both motor torques are greater than design A's motor torques for the most part.

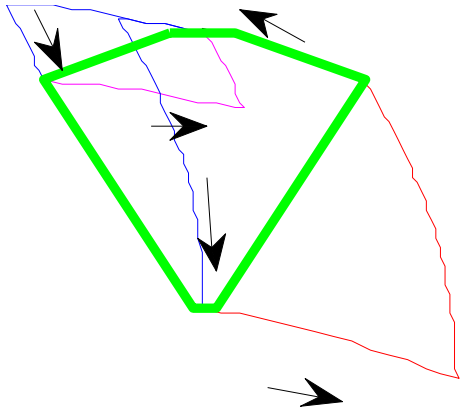


Figure 53: Design B Inverse Dynamics Analysis - Performance - Path

Design A has two points of contact with the inertial frame, hence reliving the torques considerably.

Using Equation 15, the electric torques are integrated to compute the energy consumption. The results are shown in Table 3. The advantage of design A over B is obvious: design A consumes more than 6 times less energy than design B. In fact, Motor 1, the motor mounted to the inertial frame in both designs, needs 10 times less energy if design A is used. As far as the control architecture is concerned, this translates into less power needed for actuators, smaller more maintainable actuators, higher reliability, less cost among other advantages.

Table 3: Mechanisms A & B Energy Consumption

	Design A	**Design B**
Motor 1	27.5 C	297.6 C
Motor 2	34.6 C	104.2 C
Total	**62.1 C**	**401.8 C**

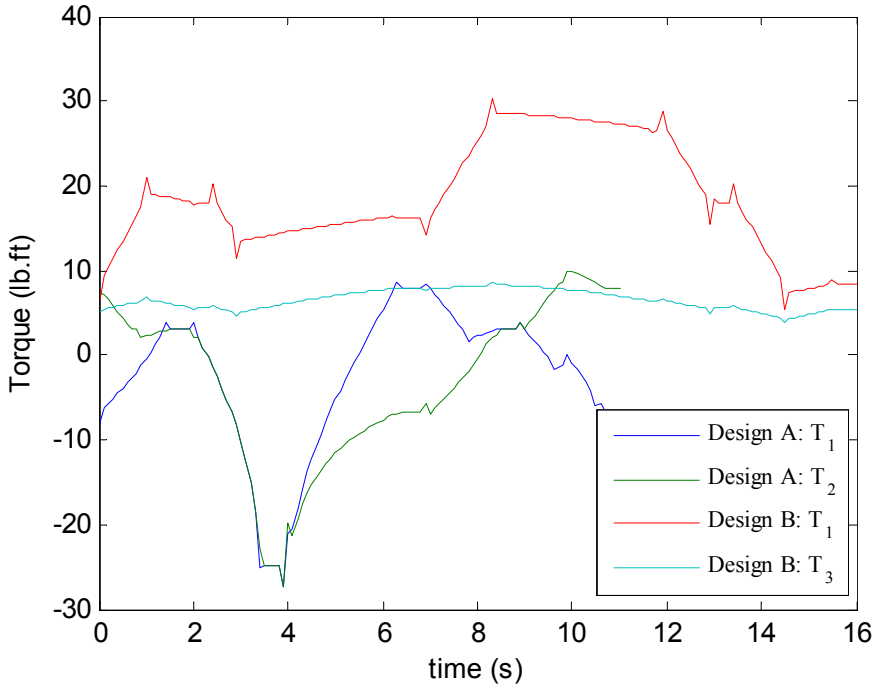


Figure 54: Design A & B Inverse Dynamics Analysis - Efficient Operation

6.2.3 Stability Requirements Analysis

Due to gravity, both mechanisms will collapse if the motors do not apply a holding torque. Both mechanisms need an energized actuator system to remain stable.

Design B is a non linear mechanism, since two independent states, θ_1 and θ_2, are needed to fully define the system are found in non linear system equation (the sine and cosine functions). Nevertheless using the small angles approximation, the equations can be linearized. This drastically simplifies controller design. Even if the non linear equations are used, the lack of constraints common to both states allows for the design of two separate and independent controllers.

Design A is a non linear design. However, its degree of nonlinearity is higher than design B's. The two independent states, θ_1 and θ_2 (or any other two states), have

common constraints given by:

$$L_1 \cos\theta_1 + L_3 \cos\theta_3 - L_2 \cos\theta_2 - L_4 \cos\theta_4 = L_1 \tag{16}$$

$$L_1 \sin\theta_1 + L_3 \sin\theta_3 - L_2 \sin\theta_2 - L_4 \sin\theta_4 = 0 \tag{17}$$

Any control architecture has to solve these non linear constraints so as to compute a control command. This adds a separate computation module to the control architecture, which requires proper integration and validation within the architecture. Although difficult, this problem is solvable. Several methods have been introduced in the literature for this specific problem, such as [41] which presents a method for the synthesis of similar mechanisms and [35] which outlines an analysis approach using the system transfer function Jacoby matrix. Such problem is non existent in the design B mechanism.

The effect of the above constraints go further. It is always easier to design a controller for system in which a positive (or negative) controller command signal results in a positive (or negative) change in one independent system state, consistently over the full range of the state. However, the ideal case is not always realizable. Provisions in the controller make sure that the proper control command sign is calculated based on the current state, or else the controller will drive the system unstable. Design A has solve the above constraints, simultaneously making sure that the angle between the two top bars, ϕ_5, is between 0^0 and 180^0, ϕ_3 is less than 180^0, and ϕ_4 is less than 180^0.

As far as design B is concerned, all what needs to be checked is that ϕ_2 is less than 180^0, a very simple condition if compared to design A's conditions and constraints.

Another effect of the above constraints is that θ_1 and θ_2 cannot take any arbitrary values. They are restricted by a feasible range, out of which the mechanism becomes unrealizable or the mathematical equations have no solution. The range of values is

(approximately) given by:

$$\theta_1 \in [65, 93] \tag{18}$$

$$\theta_2 \in [87, 115] \tag{19}$$

The above constraints are shown in Figure 55

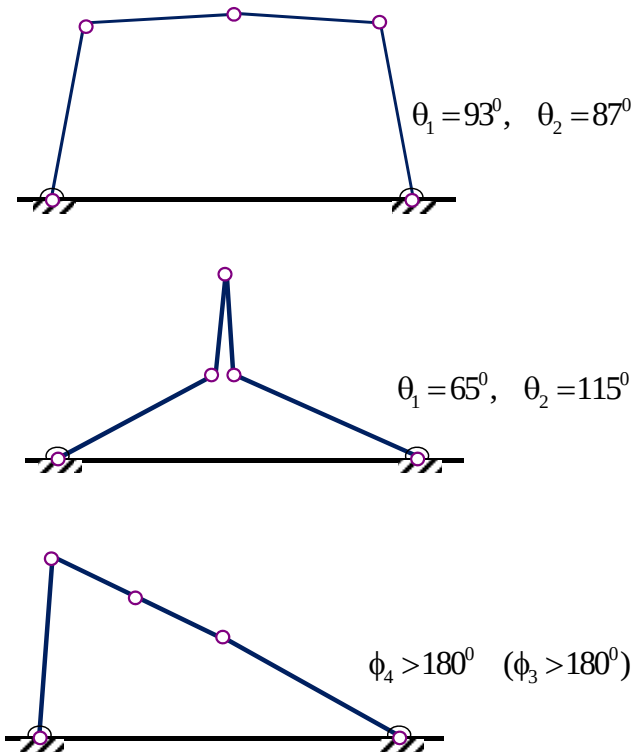


Figure 55: Design A Constraints

The angle restrictions make any controller type (even the simplest such as PID controllers) a non linear controller. The control architecture has to to have implemented modules that avoid infeasible regions. Design B has no such problems, since the two states are not connected in one constraint. In fact, any value for θ_1 can be accompanied by any value for θ_2.

Based on the above stability analysis, designing a control architecture for design A is more difficult since the controller: 1) it requires extra computation modules to

resolve the non linear constraints; 2) another module to compute the appropriate sign for the controller command; and finally 3) needs to make sure the system states (angle) remain within the correct operating range. Design B has no instability regions, together with one simple constraint, enabling a straightforward control architecture design.

6.2.4 Capability Potential Analysis

The capability potential analysis aims at exploring the limits of the control architecture / physical system capabilities. Since the main function of the mechanism is to manipulate an object, moving it from an initial position to a final position. Then it would be assumed in this context that the system capability is equivalent to that accuracy of the tip point position.

In accuracy and defined in tip point position takes place when there is an error in one (or both) of the actuator motor position. Sensor degradation might occur resulting in the same inaccuracies. another source with an accuracy might be the usage of lower-cost/lower quality actuators that are not as accurate. The study of the capability potential explores if it is possible to use degraded sensors, lower-cost actuators, or a controller with a steady-state.

The study is conducted over the full range of motion of both mechanisms. Since the mechanism states are the same states as the motor angle, an error is introduced on each state in both directions. This represents an error in the motor angle or an error in the sensor measurement. Since the mechanism has two states, then the total number of of cases to be checked is eight different combinations of error. The nominal case is computed to result in nine cases in total. The introduced motor angle error is 0.01^o.

The final coordinates of the tip point is calculated for each combination of motor angel error. The distance between this point and the nominal point is computed for

all the cases. The error is defined as the maximum of all errors at this point.

Figure 56 shows the error in tip point position for design A. For the upper half of the motion range, the error is less than 7E-4 units of length. In fact, the accuracy increases the closer we get to the upper boundary of the motion range. The lower half experiences a faster deterioration in the system capability as defined by the error. The error ranges from about 1E-4 to about 17E-4 units of length which is a large interval.

Design B's capability potential is shown in Figure 57. Note that Figure 56 and Figure 57 are drawn to the same scale. Design B experiences an almost uniform error of about 7E-4 units of length, higher than the average error in design A. However the error in design B has a very interesting property: it is uniform. This makes the control architecture performance very much predictable and robust.

6.2.5 Actuator Placement

So for two actuators were introduced in the analysis, one electric motor driving each free system state. other options might include using electric motors mounted on lead screws to rotate the bars, hence moving the position of the tip point. Yet this option, although using different kinematics, boils down to the properties of electric motor. Hence, it is a similar option.

Another alternative, in regards to design A, is to use a motor mounted to the bottom bar on the inertial frame (as with the previous analyses), and another motor mounted on the same bar at the other end. However this does not improve the design since it adds another layer of complexity resulting from the necessity of solution for the internal states. Also, all constraints need to be reformulated in terms of the new independent states. Therefore the current arrangement of actuators is optimal.

It is important to note that different mechanisms with different configurations can have another arrangement of actuators.

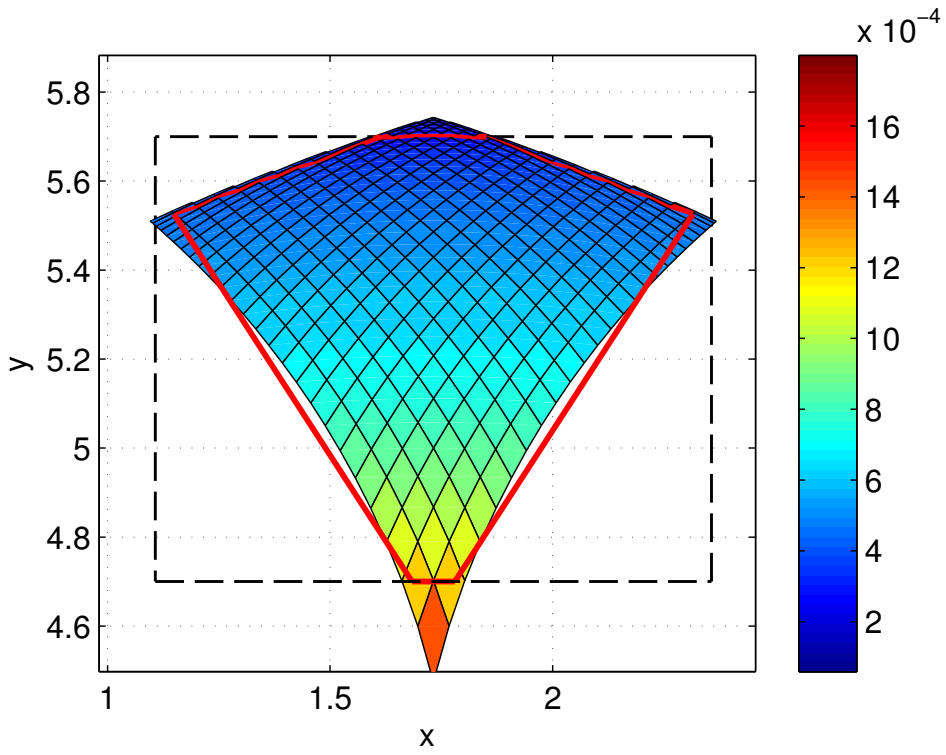


Figure 56: Design A Capability Potential - Accuracy

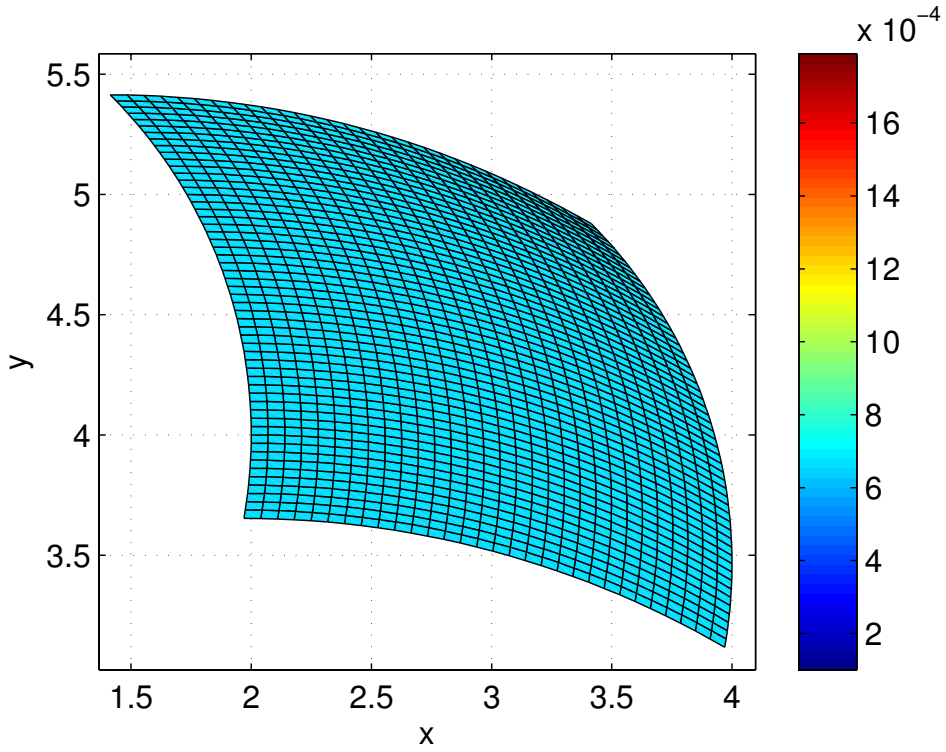


Figure 57: Design B Capability Potential - Accuracy

6.2.6 Interim Summary

The focus of this section is the design interactions between the control architecture (with no architecture specification) and the physical system. The control architecture places some capability restrictions on the physical system. Vice versa, the physical system imposes limitations on what the control architecture can do, and how it is structured. However no information was given on the control configuration or on control methods. The next section addresses the details of the control architecture.

6.3 Control Architecture Metamodel Description

The previous section addresses how the physical system and the control architecture affect one another, without exploring the different options available for the control architecture. The control architecture is completely defined by the choice of a control configuration, a control method and setting the controller parameters. It was shown in Chapter 4 that more information about the control configuration and method can be acquired during the conceptual design phase, specially along the lines of different architecture designs. This is subject of the current section.

Four different control architecture configurations are considered, namely: the Centralized, Hierarchical, Distributed and Decentralized architectures. These architectures span the landscape of control configurations from having all the control in one location (centralized), to having a central controller remotely driving local controllers (hierarchical), followed by a group of local controllers that communicate with no a high level central controller (distributed), and finally a group of completely isolated local controllers.

Local controllers (linked to an actuator or sensor) as well as higher level controllers (not linked to an actuator or sensor) are represented as nodes in the control configuration. The internal structure of each node is also presented, hence control methods can also be compare and evaluated.

All control architectures are going to be considered with both designs: A and B. Arguments regarding the feasibility and performance of these architectures are also presented. For all control architectures:

1. It is assumed that the physical system has two actuators: two motors mounted to the two side bars of design A and two motors mounted to the two bars of design B. No other actuator types are considered.

2. It is assumed that the physical system has two sensors. Two sensor alternatives are explored:

   a) All architectures / physical systems have two sensors, each measuring one of the mechanism bar's angle. These sensors might be embedded in the electric motor or an attached gearbox.

   b) In case of centralized control only, a tip point position sensor (measuring the (x, y) coordinates of the tip point) is also considered. For the hierarchical, distributed and decentralized architecture, arguments will be presented to show that such a sensor type is not possible or not desirable.

3. The actuator and sensor associated with the same bar are connected to the same local controller.

4. No communication loss, nor communication bandwidth limitations exist.

5. Every control node (local or higher level controller) is embedded on one processor, with multiple running processes if needed.

6. The simplest control method is assumed in each node such that the controller embedded controller is able to accomplish its function.

7. Two mechanism use cases will be considered; a main use case and an extended one. The main use case is to move the tip point from an arbitrary initial location

to a required final location. The extended use case is to do the same, but along a predefined path, such that the tip point trajectory follows that path as close as possible.

The analogy between control architectures and software is exploited. The control architecture metamodel allows for control configurations, as well as control methods, to be treated as connected software modules. Hence, evaluating many software metrics on the control architecture metamodel is possible. Two software metrics are calculated:

1. *Amdahls' Law* measures the maximum expected *speed up* in a computer program as a result of using multiple processors in parallel. This is possible because every controller may be embedded on single processor, remotely communicating with another controller on a different processor. This is a classical parallel programming problem. Amdahls Law is explained in Appendix B.

2. *Cyclomatic Complexity* measures the amount of decision logic in a source code function or piece of software [145]. The more decisions that need to be made within a computer program, the more complex it is. The cyclomatic complexity metric is applied twice. First, it is computed for the control configuration applied the full physical system. Next, the complexity of the the internal node structure is calculated for each node in the control configuration. Cyclomatic complexity is covered in Appendix C.

6.4 Mechanism Control Architecture Metamodel

6.4.1 Centralized Control Architecture

The centralized control architecture, as its name implies, consists of a single multi input multi output controller. It receives sensor measurements of the two independent states, processes the measurements, compares them to the external input, and produces two commands to the two electric motors.

Centralized control architectures are very common in simple engineering applications. As the number of independent states increase, and a number of constraints on the states increase and become more complex, centralized control does not seem like a good option for many for such systems.

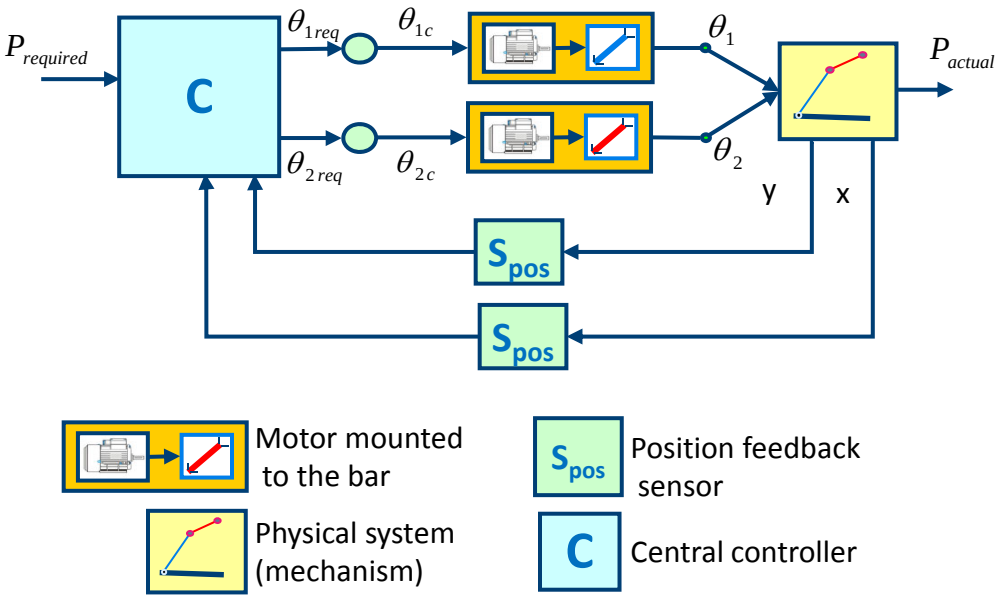


Figure 58: Centralized Control Architecture Block Diagram

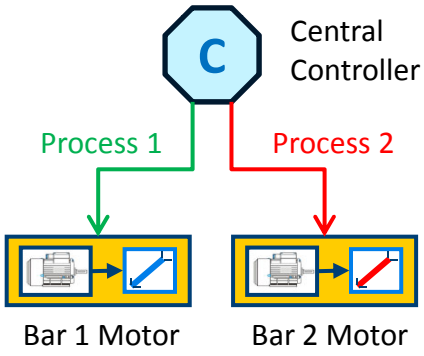


Figure 59: Centralized Control Architecture Configuration

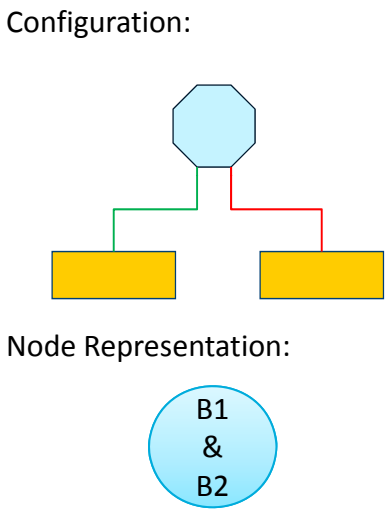


Figure 60: Centralized Control Architecture - Configuration Complexity

6.4.1.1 Configuration Feasibility

Figure 58 shows the block diagram for the centralized control architecture. The block diagram applies to both design A and design B. The controller takes a required point as its input and compares this point to the feedback from the sensors. The controller outputs are to commanded angles to the electric motors. it has to have a embedded function to transform the required tip point coordinates to a pair of angles that define the two sidebar orientation. This function would almost be referred to as $F(x, y)$. In case of design A, such a function is mathematically complex, yet is realizable using the equations presented earlier. However, design B has a simple relationship that describe the relationship between a required final point and bar orientation. Hence design A has a more complex, computationally intensive centralized controller as compared to design B.

The feedback signal, as shown in figure 55, has two sensors. The two sensors measure the x and y coordinates of the tip point, and feed them back. The centralized control is feasible because the controller function can compare an acquired point with an actual point. The error can then be transformed to two commanded angles. This

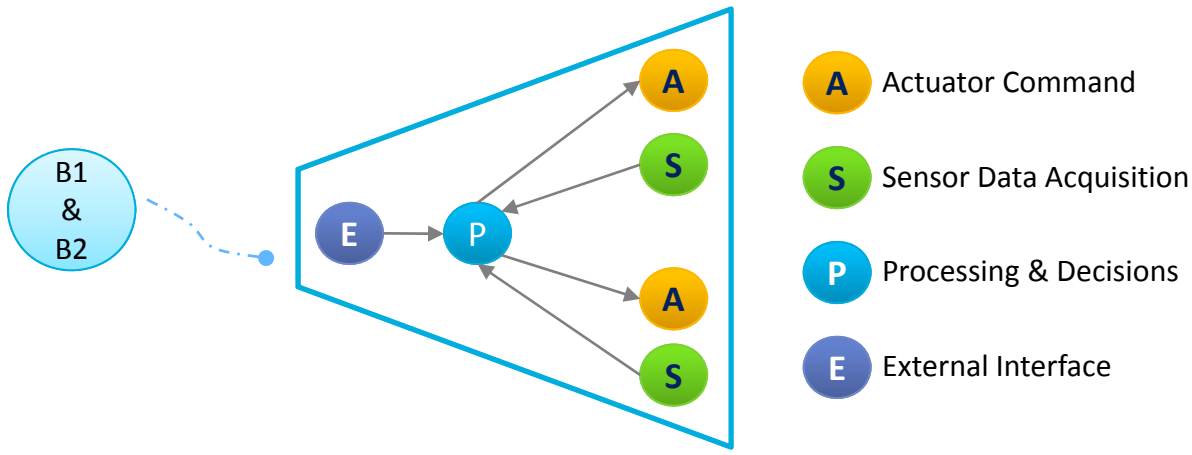


Figure 61: Centralized Control Architecture - Node Complexity

feedback method has the advantage of directly measuring the error in the mechanism output since its main function is to accurately move the tip point scratch that to a required location.

Another feedback method, discussed in later architectures, is to use the bar angles on the feedback loop. Most electrical motors have embedded sensors that measure the motor angle, speed, acceleration. The bar angle can be computed from the motor angle using the mechanism's kinematic relationships. However, this means that an extra module has to be embedded in $F(x, y)$ such that it transforms the feed back angles to a point on the plane and compare it with a required point, or different transformation to change the required point to bar angles and directly compare them with the feedback signals.

Considering the mechanism extended use case (following a predefined path), the controller function $F(x, y)$ can be modified to accomplish this task. This increases the internal complexity of the centralized control architecture node. The extended use case is feasible for both design A and B, although it is more computationally intensive in design A's case.

6.4.1.2 Amdahl's Law - Speedup

The block diagram in Figure 58 is redrawn using the meta-model representation discussed in section 5.6. The result is shown in Figure 59. Since centralized controller has two independent control channels, one for each motor. It is assumed that each channel has an independent process. Also, since the controller architecture has one single controller (the centralized controller), then it can be assumed that there are no other operations taking place on the processor associated with this controller. Hence the speed up S is be given by

$$S = \frac{1}{(1-P)+\frac{P}{N}} = \frac{1}{(1-0)+\frac{0}{2}} = 1$$

Centralized control counts as the nominal case because, by definition, it contains only one processor. The speedup is expected to be 1 for this reason. This is true for both design A and design B.

6.4.1.3 Control Configuration Cyclomatic Complexity

Figure 60 shows the metamodel representation and the node representation of the centralized control architecture for clarity. It is basically composed of a single node, with zero edges and one output (the node itself). Hence the cyclomatic complexity of the centralized control architecture configuration is given by

$$M = E - N + 2P = 0 - 1 + 2 \times 1 = 1$$

The centralized control configuration complexity counts as the baseline complexity of all other control configurations.

6.4.1.4 Control Node Cyclomatic Complexity

Next, the cyclomatic complexity of the node structure is addressed. This refers to the modules within the node that were discussed in Section 5.7 (see Figure 38). Figure

61 shows the internal node structure for the centralized controller. It consists of an external interface receiving the required final point from the user, a processing and decision making module that receives the user input and feedback signals and issues commands to the actuators. It has five edges, six internal modules and two outputs. Hence, the cyclomatic complexity of the node is given by

$$M = E - N + 2P = 5 - 6 + 2 \times 2 = 3$$

This is true for both design A and design B, since they both have the same node structure.

6.4.2 Hierarchical Control Architecture

The hierarchical control architecture consists of one high level controller and two low level controllers. The high level layer takes a required final point and transforms it into two commanded angles to the local bar controllers. Each local controller receives a commanded angle, compares it to the actual bar angle, and issues actuator commands accordingly. The feedback loop is local to the low level (local) controller.

Layered hierarchical control architectures are very common in large scale engineering applications. The top layers are usually associated with system level decisions or course of action. Mid level layers may take the system wide course of action and translates it into specific commands that lower level layers may understand. The lower level layers ensure the fulfillment of these commands. Low level layers report local status (but not necessary system states) to mid level controllers which in turn report subsystem's status to high level controllers.

6.4.2.1 Configuration Feasibility

The hierarchical control architecture block diagram is shown Figure 62. It applies to both design A and design B. The high level controller takes required final point as its input. It has a similar role to $F(x, y)$ discussed in the centralized controller

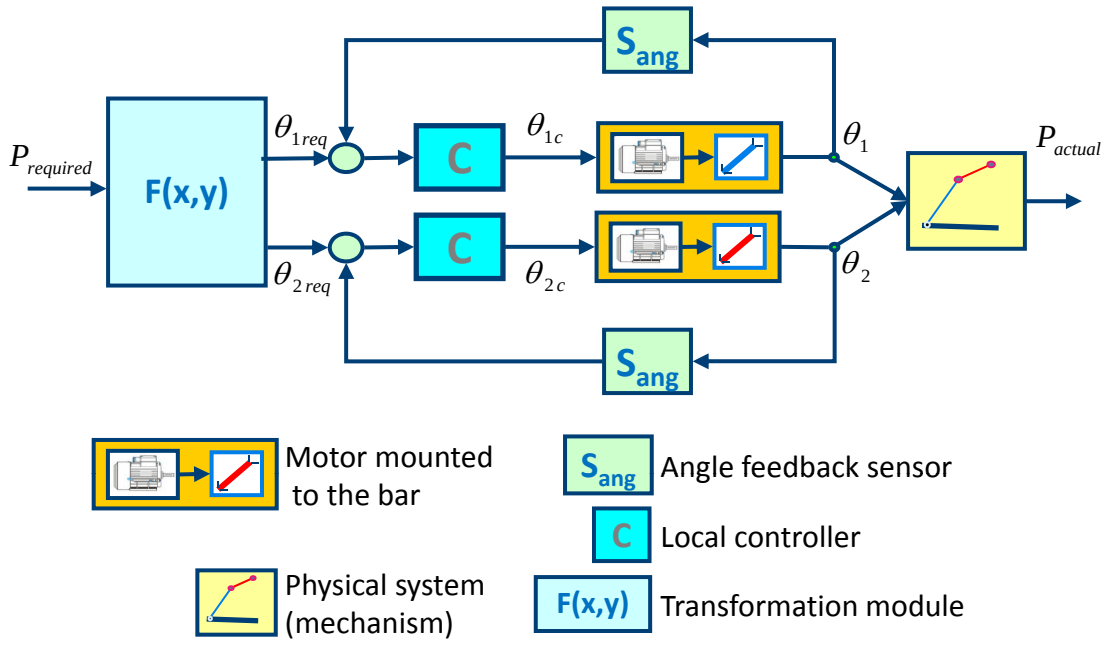


Figure 62: Hierarchical Control Architecture Block Diagram

architecture. It transforms the final point coordinated into a a pair of final bar angles. The low level controllers ensure that these final angles are achieved by the bars. The feedback is local, i.e. the bar angle is fed back to the low level controller.

Feeding back the (x, y) coordinates of the finest tip point is not a feasible option for both design A and B because of two reasons. The first reason is that each controller is local, hence it only has access to the local state. The (x, y) coordinates are not available for the two local controllers in design A since both controllers are associated with the motors on the sidebars, and the tip point sensor is between the two top bars. Similarly for design be, the sensor is located on the top bar, hence its output is only available for the top bar motor. Another reason is that even if the sensor measurement was available for both local controllers, each local controller should have the function $F(x, y)$ to compute the current bar angle. This has to be done within both local controllers, hence adding two extra modules with a additional computational expense.

The extended use case is better satisfied by hierarchical control than by centralized

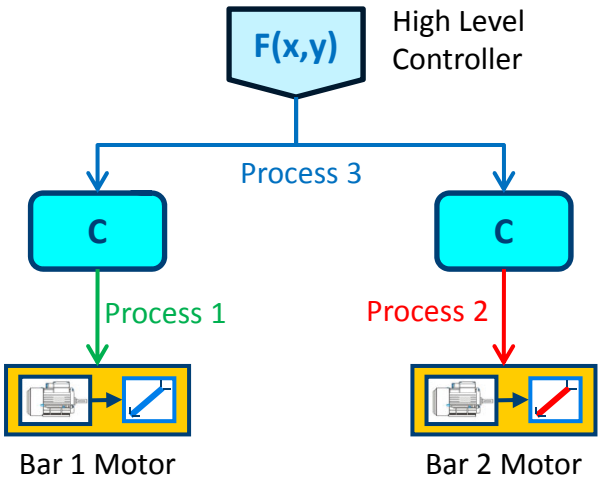


Figure 63: Hierarchical Control Architecture Configuration

control. The reason is as follows.: An extra module, a higher level one, is added to the current hierarchical control architecture as it stands now. Hence no reprogramming of any of the control modules is necessary. The function of this higher level module is to transform the required path into a group of points that are applied to the current a high level control, which in turn transforms those into angle commands to the local controllers. Again, this is applicable to both design A and design B.

The advantage of hierarchical control lies in the local controllers. In this case a local controller is a very simple controller that uses the most straight forward methods. For example, at simple PID control works perfect for both designs with no need for any intelligence at the local level. This simplifies the controller design considerably, because the intelligence can be designed independently from the hardware control. Also, in design A's case, the local controllers do not have to compute the solutions for the constraints. These are handled in the top-level control. the disadvantage of hierarchical control is that it needs a communication network and a communication channel always open. Communication network faults might lead to total system failure.

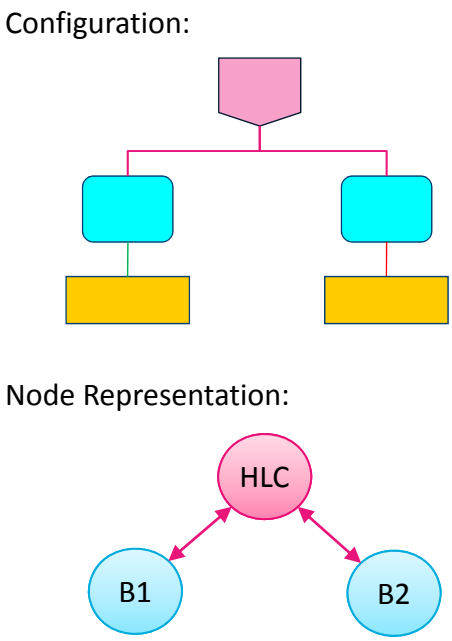


Figure 64: Hierarchical Control Architecture - Configuration Complexity

6.4.2.2 Amdahl's Law - Speedup

The block diagram in Figure 62 is redrawn using the meta-model representation discussed in section 5.6. The result is shown in Figure 63. The hierarchical control architecture has three independent control channels, one for each local controller with the associated motor, and another for link from the high level controller to the local controllers. It is assumed that each channel has an independent process. Also, since the control architecture has three controllers (one high level and two local), then it can be assumed that the control architecture has three processors. In addition, it can be assumed that two out of the three processes can be parallelized. Hence the speed up S is be given by

$$S = \frac{1}{(1-P) + \frac{P}{N}} = \frac{1}{\left(1 - \frac{2}{3}\right) + \frac{2/3}{3}} = 1.8$$

The speedup for hierarchical control is almost twice that of centralized control.

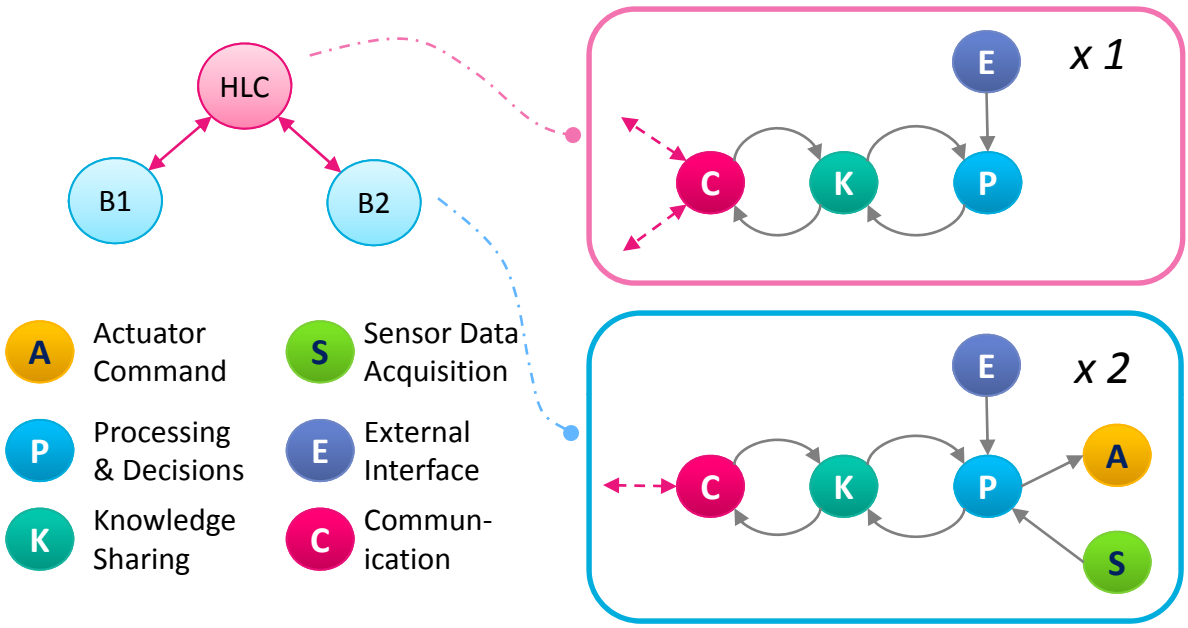


Figure 65: Hierarchical Control Architecture - Node Complexity

In other words, the control architecture will theoretically run twice as fast. This is true for both design A and design B.

6.4.2.3 Control Configuration Cyclomatic Complexity

Figure 64 shows the metamodel representation and the node representation of the hierarchical control architecture. It has three nodes, as explained previously, with two edges and two outputs (the two actuators). Hence the cyclomatic complexity of the hierarchical control architecture configuration is given by

$$M = E - N + 2P = 2 - 3 + 2 \times 2 = 3$$

The hierarchical control configuration complexity is three times as much as the centralized control configuration. However, it is still well within a manageable range.

6.4.2.4 Control Node Cyclomatic Complexity

Next, the cyclomatic complexity of the node structure is addressed. This refers to the modules within the node that were discussed in Section 5.7. Figure 65 shows

the internal node structure for the high level controller, and the two low level local controllers. The high level controller consists of an external interface receiving the required final point from the user, a processing and decision making module that receives the user input and feedback signals and makes system wide decisions. The decisions are then handed over to the knowledge sharing module, which decides on the destination of each piece of information. Finally, the communication module places the information on the network. The high-level controller has five edges, four modules and one output, hence the cyclomatic complexity of the high level controller is given by

$$M = E - N + 2P = 5 - 4 + 2 \times 1 = 3$$

Similarly the low-level controller has a communication module, a knowledge sharing module and a processing module. In addition, it has two modules associated with actuator in the sensor. Therefore, it has seven edges, six nodes, and two outputs. Hence the cyclomatic complexity of the low-level controller is given by

$$M = E - N + 2P = 5 - 4 + 2 \times 1 = 5$$

It is important to note that there are two lower-level controllers. This means that the cyclomatic complexity of 5 is per controller, not the overall complexity of both. It is still within the manageable range.

6.4.3 Distributed Control Architecture

The distributed control architecture consists two local controllers, each associated with one of the electric motors. Unlike the hierarchical control architecture, the distributed control architecture does not have a high level control. Instead, it has a coordinating controller in between the two local controllers, whose function is to make sure that necessary information is passed between them. The coordinating controller does not make system wide decisions, nor command a course of action. The required

point is passed to both local controllers. Each controller then transforms required point to a commanded bar angle, and ensures that the bar it controls rotates to this angle. As is the case with hierarchical control, the feedback loop is local to each local controller.

Distributed control is very common in loosely connected large-scale systems. Sometimes it is necessary that the coordinating controller chooses the appropriate states to pass between the local controllers so that it ensures stability and adequate performance.

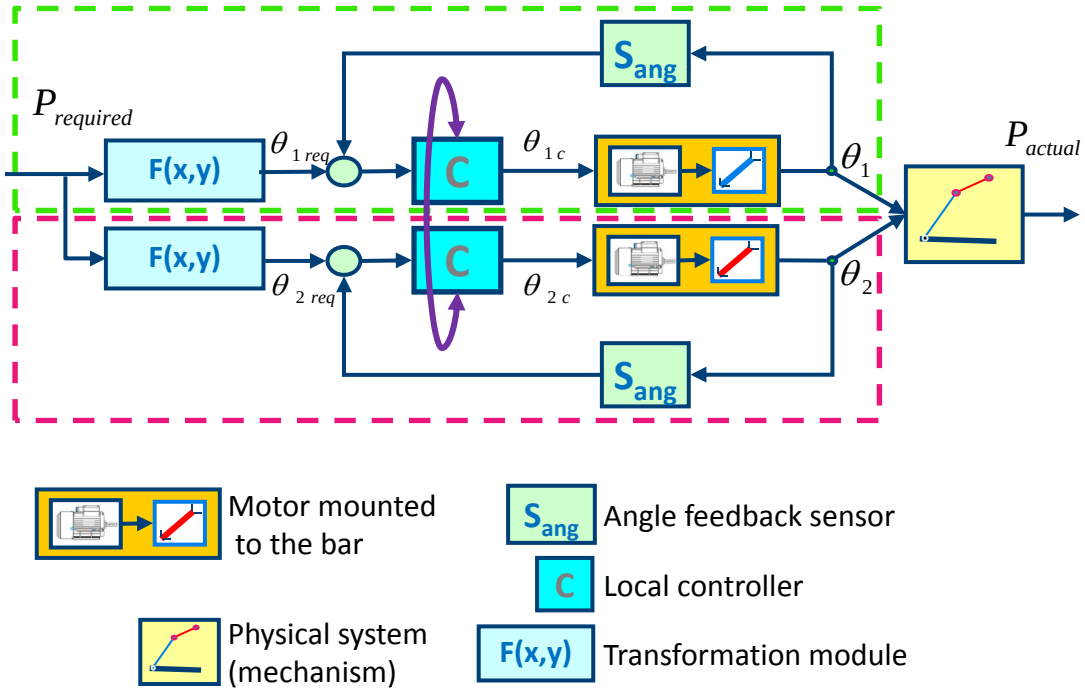


Figure 66: Distributed Control Architecture Block Diagram

6.4.3.1 Configuration Feasibility

The distributed control architecture block diagram is shown Figure 66. Each local controller takes the required final point as its input. It has a similar role to $F(x, y)$ discussed in the centralized controller architecture. It transforms the final point coordinated into a a pair of final bar angles, but only uses the relevant one in driving

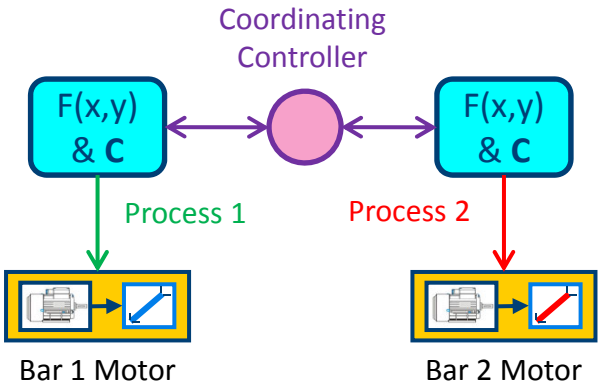


Figure 67: Distributed Control Architecture Configuration

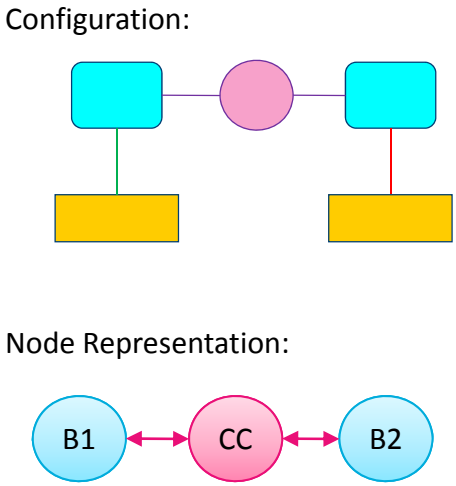


Figure 68: Distributed Control Architecture - Configuration Complexity

the bar it controls. The feedback is local, i.e. the bar angle is fed back to the low level controller. The two local controllers exchange the current bar angle through the coordinating control. In addition, the coordinating controller might have the responsibility of restricting the motion of either of the local controllers.

The position feedback of the point coordinates (x, y) does not work for similar reasons to the ones discussed in hierarchical control.

Design A has kinematics constraints that restricts the ranges of the bar angles. These constraints have to be accounted during the motion of the mechanism, or else

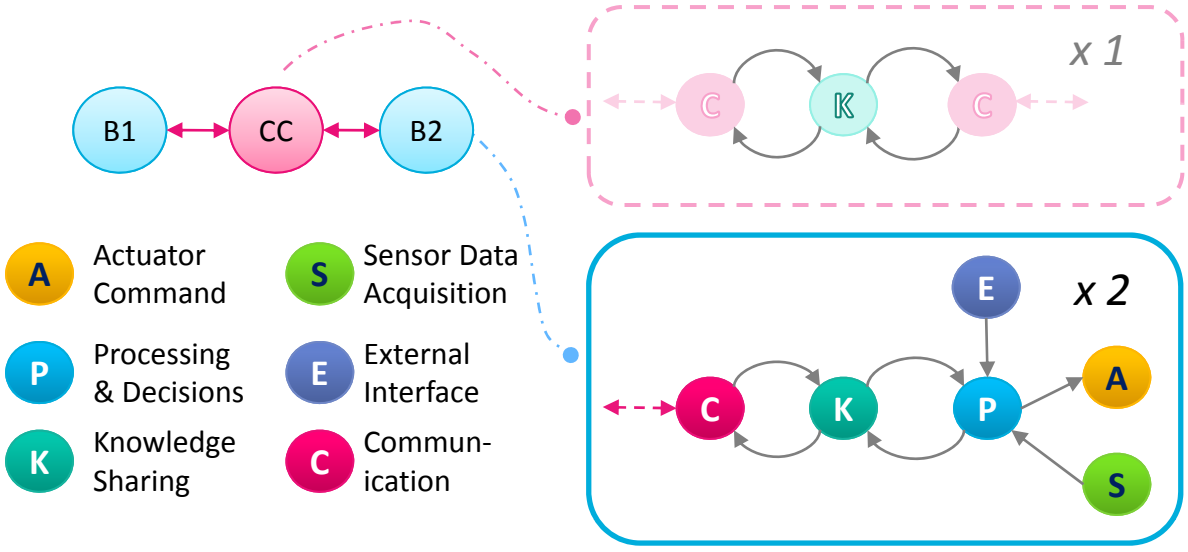


Figure 69: Distributed Control Architecture - Node Complexity

the mechanism may go out of bounds, or get into a deadlock. Therefore a coordinating controller is necessity to ensure the proper operation. the extended use case of following a certain path is also possible by changing the modules within the local controllers, and upgrading the coordinating control. However, this involves a higher degree of complexity. Due to the sub performance of design A in this configuration, it will not be considered for distributed control.

Design B is more graceful than design A. Since there are no constraints that include both the independent states of the mechanism, then each state can be controlled independently, without knowledge of the other state. This works for the main use case only, although this may result in a higher energy consumption as is shown in the inverse dynamics analysis. A coordinating controller in this case functions as a limiter on the states to save energy. As far as the extended use case is concerned, it is also feasibly using the coordinating control. Hence the role of the coordinating controller is to enhance the system performance, not to ensure that the system is feasibly as with design A. The coordinating controller will not be considered in the assessment of design B.

6.4.3.2 Amdahl's Law - Speedup

The block diagram in Figure 66 is redrawn using the meta-model representation discussed in section 5.6. The result is shown in Figure 67. The distributed control architecture has two independent control channels, one for each local controller with the associated motor. It is assumed that each channel has an independent process. Also, since the control architecture has two controllers, then it can be assumed that the control architecture has two processors. In addition, it can be assumed that one out of the two processes can be parallelized. Hence the speed up S is be given by

$$S = \frac{1}{(1-P) + \frac{P}{N}} = \frac{1}{\left(1 - \frac{1}{2}\right) + \frac{1/2}{2}} = 1.3$$

The theoretical speedup for distributed control is not much improvement over centralized control. It is halfway between centralized and hierarchical control architectures.

6.4.3.3 Control Configuration Cyclomatic Complexity

Figure 68 shows the metamodel representation and the node representation of the distributed control architecture. It has three nodes, as explained previously, with two edges and two outputs (the two actuators). Hence the cyclomatic complexity of the distributed control architecture configuration is given by

$$M = E - N + 2P = 2 - 3 + 2 \times 2 = 3$$

The distributed control configuration complexity is three times as much as the centralized control configuration and equal to hierarchical control architecture. However if the coordinating control is not considered, then the Cyclomatic complexity would be

$$M = E - N + 2P = 1 - 2 + 2 \times 2 = 3$$

which is the same result as before.

6.4.3.4 Control Node Cyclomatic Complexity

Next, the cyclomatic complexity of the node structure is calculated. This refers to the modules within the node that were discussed in Section 5.7. Figure 69 shows the internal node structure for the local controllers, and the coordinating controller. The coordinating controller consists of a knowledge sharing module which decides on the information shared between nodes. The two communication modules place the information on the network. The coordinating controller has four edges, three modules and two outputs, hence its cyclomatic complexity is given by

$$M = E - N + 2P = 4 - 3 + 2 \times 2 = 5$$

Similarly the local controller has a communication module, a knowledge sharing module and a processing module. In addition, it has two modules associated with actuator and the sensor. Therefore, it has seven edges, six nodes, and two outputs. Hence the cyclomatic complexity of the low-level controller is given by

$$M = E - N + 2P = 5 - 4 + 2 \times 1 = 5$$

It is important to note that there are two local controllers. This means that the cyclomatic complexity of 5 is per controller, not the overall complexity of both. It is still within the manageable range.

6.4.4 Decentralized Control Architecture

The decentralized control architecture consists of two local controllers with no communication between them. Each local controller works in isolation driving part of the system, in this case one of the twp bars. The required point is passed to both local controllers. Each controller then transforms required point to a commanded bar angle, and ensures that the bar it controls rotates to this angle. The feedback loop is local to each local controller.

Decentralized control only works for systems with uncoupled states.

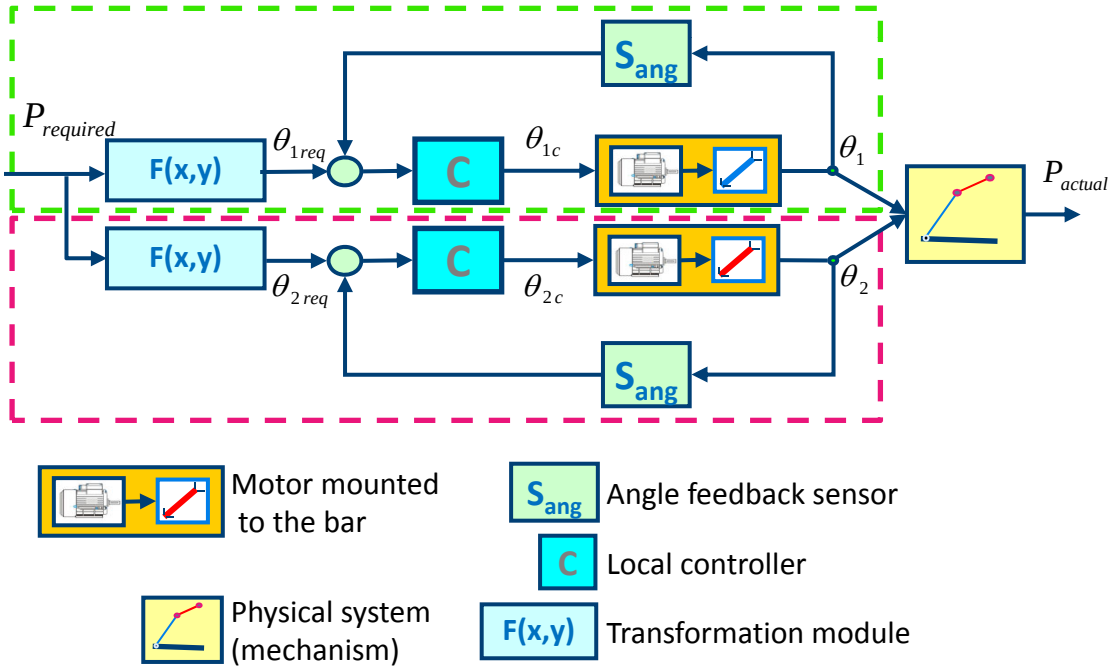


Figure 70: Decentralized Control Architecture Block Diagram

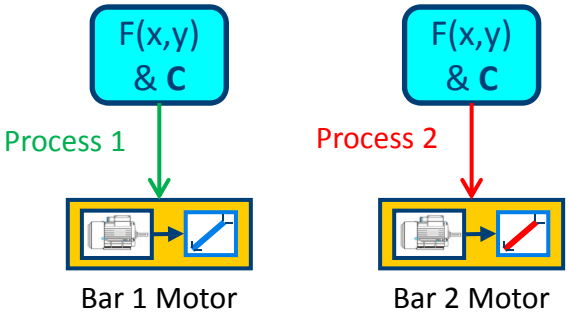


Figure 71: Decentralized Control Architecture Configuration

6.4.4.1 Configuration Feasibility

The decentralized control architecture block diagram is shown Figure 70. Each local controller takes the required final point as its input. It has a similar role to $F(x, y)$ discussed in the centralized controller architecture. It transforms the final point

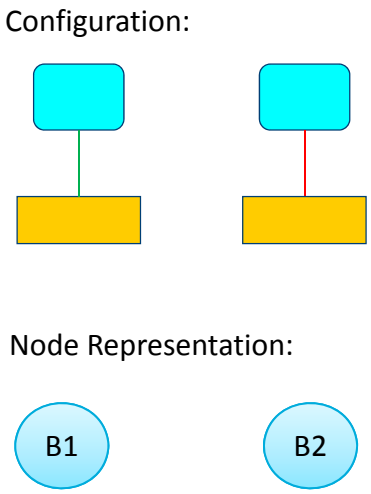


Figure 72: Decentralized Control Architecture - Configuration Complexity

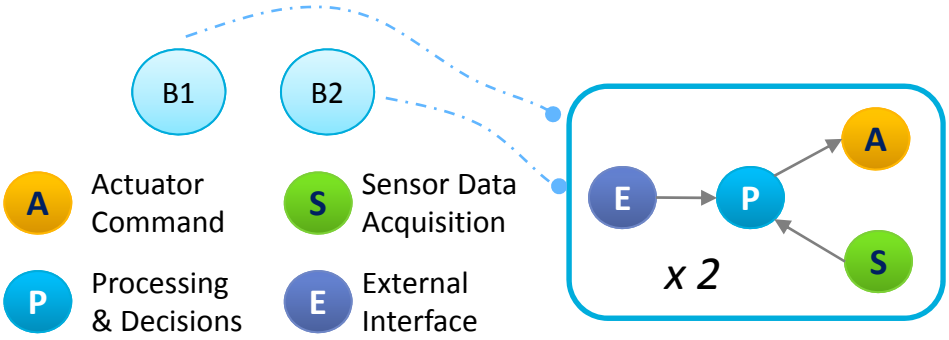


Figure 73: Decentralized Control Architecture - Node Complexity

coordinated into a a pair of final bar angles, but only uses the relevant one to drive the bar it controls. The feedback is local, i.e. the bar angle is fed back to the low level controller. There is no communication among the two local controllers.

The position feedback of the point coordinates (x, y) does not work for similar reasons to the ones discussed in hierarchical control.

Design A cannot be controlled using a decentralized control architecture, for the same reasons presented in the distributed control architecture discussion.

Design B has decoupled states with no constraints binding them. Each state

can be controlled independently, without knowledge of the other state. Hence the decentralized control is possible for the main use case as well as the extended use case.

6.4.4.2 Amdahl's Law - Speedup

The block diagram in Figure 70 is redrawn using the meta-model representation discussed in section 5.6. The result is shown in Figure 71. The decentralized control architecture has two independent control channels, one for each local controller with the associated motor. It is assumed that each channel has an independent process. Also, since the control architecture has two controllers, then it can be assumed that the control architecture has two processors. In addition, it can be assumed both processes can be parallelized. Hence the speed up S is be given by

$$S = \frac{1}{(1-P) + \frac{P}{N}} = \frac{1}{\left(1 - \frac{1}{1}\right) + \frac{1/1}{2}} = 2$$

which is slightly more than hierarchical control.

6.4.4.3 Control Configuration Cyclomatic Complexity

Figure 72 shows the metamodel representation and the node representation of the distributed control architecture. It has two nodes, as explained previously, with no edges and two outputs (the two actuators). Hence the cyclomatic complexity of the decentralized control architecture configuration is given by

$$M = E - N + 2P = 0 - 2 + 2 \times 2 = 2$$

which is slightly less than hierarchical control.

6.4.4.4 Control Node Cyclomatic Complexity

The cyclomatic complexity of the node structure is calculated. Figure 73 shows the internal node structure for the two local controllers. Each has a processing module,

two modules associated with actuator and the sensor, and the external interface. Therefore, it has three edges, four nodes, and one output. Hence the cyclomatic complexity of the local controller is given by

$$M = E - N + 2P = 3 - 4 + 2 \times 1 = 1$$

This is the least cyclomatic complexity thus far.

6.5 Control Architecture Demonstration

So far, two main tasks were addressed: the interaction between the physical system and the control architecture has been presented in section 6.2, followed by the control architecture metamodel representation in section 6.3. The analysis suite, as well as the metamodel representation shed some light on how the control architecture should look like in terms of structure, and applicable methods. A demonstration of an arbitrary selected control architecture is presented in this section to further extend the conclusions drawn from the two previous tasks.

The objective of the mechanism is to move the tip point from an initial location to a final location. An additional requirement is to move the point and along a pre-specified path. The control architecture can be implemented in one of two ways. The first option is using a hierarchical control architecture, such that the top-level control node decides on the path points, while lower-level control nodes drive the actuators. the second option is using centralized control, in which all the decisions regarding attacks as well as driving the hardware are made within a single control node.

The centralized control architecture is arbitrarily chosen for design B as a platform to perform further studies. There are two reasons for this choice. Firstly, it is assumed that the control engineer can design a hierarchical control architecture that is comparable in performance to a centralized control architecture. The second reason relates to the inherent need of hierarchical control for a communication network. Although this might be important for such analyses, it is not as effective

when it comes to the special case system that we use for this demonstration. The system is all mechanical, and all its components are co-located in the same vicinity. Therefore the time constant for such a system is orders of magnitude higher than the time constant for the electric and communication networks. As the communication network does not affect the outcome much. The centralized control architecture is chosen so that networking effects are decoupled from the system performance.

It is worth noting that for a large-scale, spatially distributed (over a large geographic area) system networking effects become profound and cannot be neglected. Also, bandwidth may become a concern. Nonetheless, this is not the case for the mechanism under consideration.

6.5.1 Controller Design Overview

The input to the centralized controller is a list of waypoints that the mechanism is it required to go through. The outputs are two commanded torques to the two electric motors driving the two bars. The controller chooses the first waypoint, compares it to the mechanism's current position, and based on the difference between the two coordinates, computes a control signal that represents the commanded torque to the motor. Since there are two motors, the controller has to compute a torque for each. Each torque is calculated using a simple PID controller on a separate channel. Once the mechanism reaches the waypoint, a signal is sent back to the controller to switch to the next waypoint in the list.

The controller has four main modules. The first module is the sub-controller which is responsible for comparing the current position with the commanded waypoint, and switching to the next one on the list once the current mechanism position reaches the commanded waypoint (within a tolerance of 0.01). It is implemented using an s-function in Matlab (see appendix A). The second module takes the current waypoint coordinates as an input and computes the corresponding bar angles using the system's

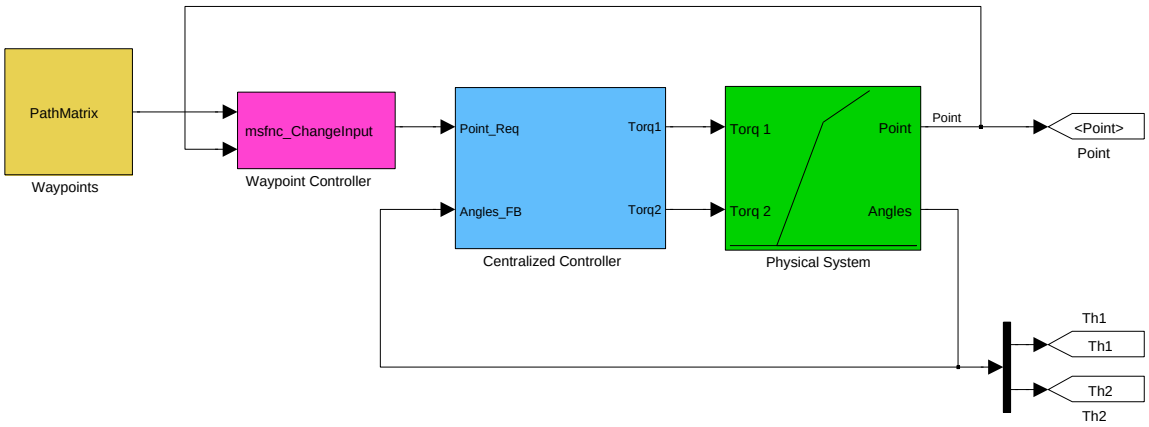


Figure 74: Centralized Control Architecture Implementation

kinematics equations. This module is implemented using a Simulink block diagram. the third and fourth modules are the two PID controllers, each taking the commanded angle as an input and computes the commanded torque to the motor. the PID controller proportional, integral, differential gains are 20, 10, and 4 respectively. The Simulink block diagram is shown in Figure 74.

6.5.2 Controller Operation - Sensitivity Analysis

To analyze the controller operational, it is assumed that the mechanism moves around a predefined path. The path is in the form of a square, therefore has four waypoints. The mechanism's initial position places the tip point outside the square. Hence it has to move to the first waypoint to get on the path. The square path is well within the reachability of the mechanism. Also, the above analysis showed that the error is almost uniform making the error in any of the waypoints equal for all practical purposes.

In the first simulation, the mechanism is commanded to move around the square path, divided into four segments. Each segment is a defined by an initial and a final waypoint. The controller computes the error between the current location and required final waypoint along the path segment, and chooses the appropriate command

torque signal. Note that the path is not controlled in between the two waypoints defining the path segment; the controller takes into consideration the segment's final waypoint only.

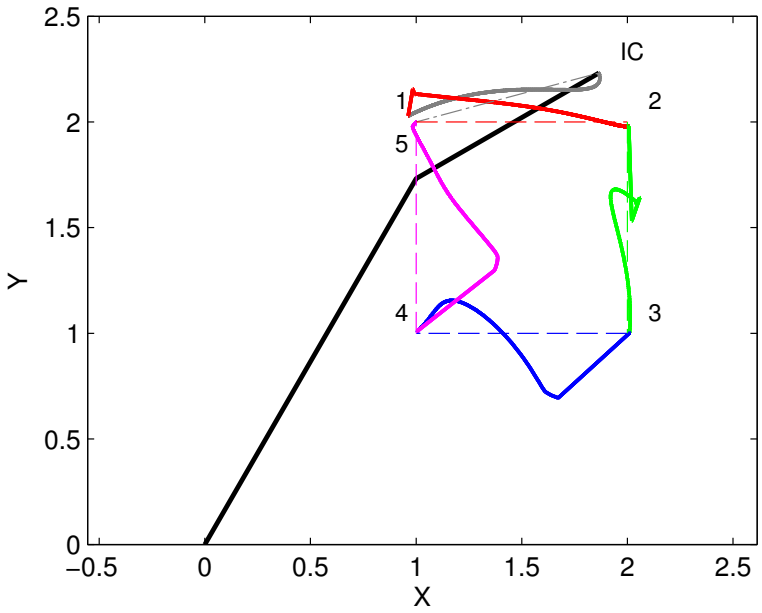


Figure 75: Trajectory - Segment Defined by Two Boundary Waypoints

Figure 75 shows the mechanism trajectory and the nominal path. The path waypoints are numbered from 1 to 5 (with waypoints 1 and 5 being the same). The mechanism trajectory starts from its initial position, moves to the first waypoint, then to point 2, followed by waypoint 3, then 4, and finally back to 1. The trajectory is unconstrained in between the waypoints, as is the case in the control architecture analyses suite discussed earlier. The mechanism undergoes an unconstrained trajectory between the segment boundary waypoints, with no apparent pattern. The four segments of the path have a different structures. For example, segment 1-2 is more or less straight, while segment 2-3 has a loop. In spite of being an advantage when compared to the earlier mechanism A, the lack of coupling between the independent mechanism states results in this almost sporadic behavior. There is no guarantee where the mechanism is going to be in between waypoints, unless simulations are run

priori. This result verifies the inverse dynamics conclusion obtained earlier.

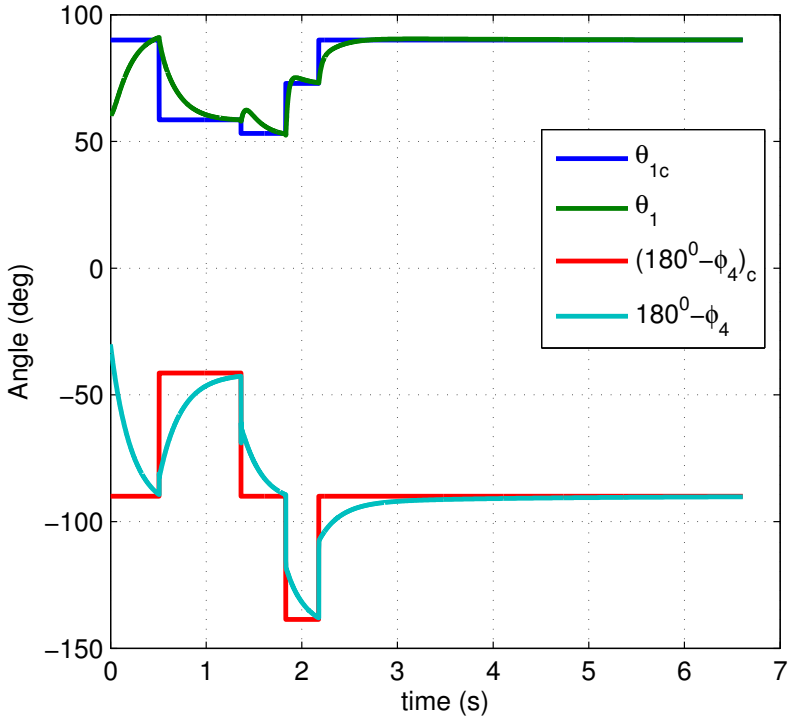


Figure 76: Bar Angles - Segment Defined by Two Boundary Waypoints

The error tolerance of the position vector is arbitrary set at 0.01 ft. The controller assesses the position error by computing the difference between the current tip point position vector and the waypoint position vector. Figure 77 shows the position error at each point along the path. As soon as the controller detects that the error is less than the tolerance limits, it assumes that the mechanism reached the sought waypoint and it moves to the next one on the list. It uses this error measurement as a basis to compute the torque commands to the motors. The error is maximum at the beginning of each path segment, resulting in a higher torque command, which in turn initially drives the mechanism faster. The higher torque command is the main reason why the mechanism completes the path in about 5 seconds. Note the four error peaks corresponding to the four waypoints.

Considering the two boundary waypoints is an acceptable approach only if the designer cares less about the intermediate mechanism behavior. However, this is not

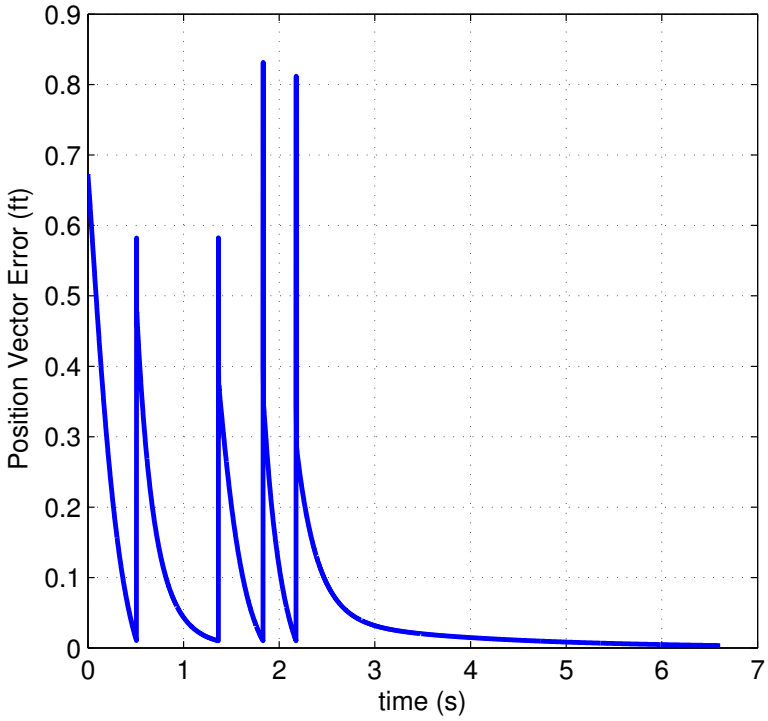


Figure 77: Position Vector Error - Segment Defined by Two Boundary Waypoints

the case in general. Such mechanisms are usually required to avoid obstacles during motion or move on a certain path, not merely move between the initial and final points. An alternate approach is to add intermediate or internal waypoints along each path segment. Figure 78 shows the resulting mechanism tip point trajectory if four internal waypoints are added to each path segment, amounting to a total of 20 waypoints along the path. The trajectory better follows the path (if compared to the previous case) because the controller drives the system from on internal waypoint to another. The mechanism has the same almost sopradic behavior in between internal waypoints, so it remains "in the vicinity" of the path rather than "on the path".

Note that the same PID controller is used, together with the same tolerance limits, and the same path boundary waypoints.

The internal waypoints are more clear in Figure 79. The commanded angles are changed based on the current waypoint, and the motor controller tries to drive the motors to this angle. The actual bar angles follow the commanded angles well enough

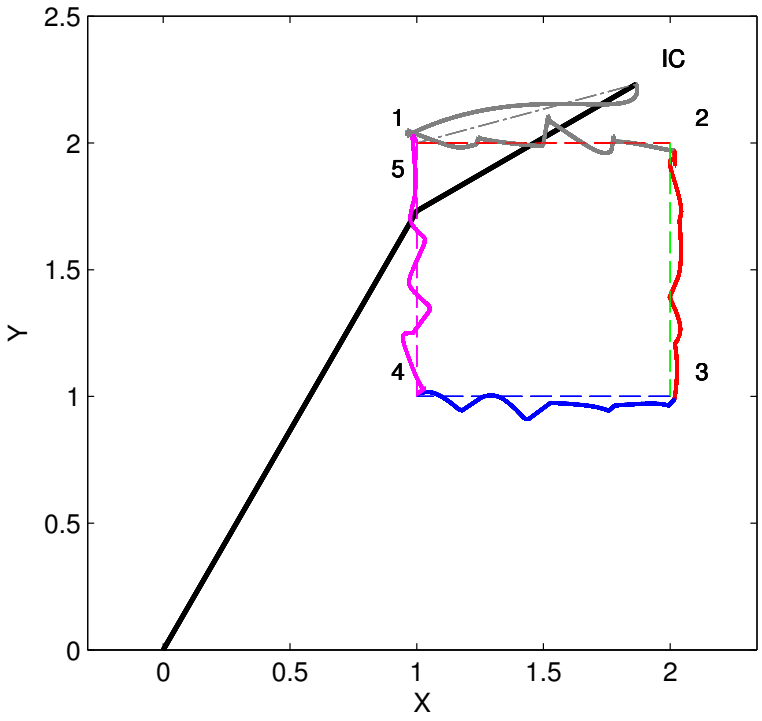


Figure 78: Trajectory - Segment Defined by Four Internal Waypoints

to put the mechanism in the vicinity of the path. With the exception of the mechanism initial condition, the error in Figure 80 hardly goes over 0.2 ft (since the waypoints are now closer). This results in motor commanded torques of low magnitudes, driving the mechanism slower than before. The mechanism completes the required path in about 15 seconds, three times as much as the previous case.

The number of internal waypoints is increased to nine, bringing the total number of waypoints to 40. Figure 81 plots the mechanism's tip point trajectory for this case. The trajectory closely follows the path on some segments but deviates to its vicinity on other segments.

A pattern is now starting to emerge. The further the path segment from the origin, the better the trajectory follows the segment. Also, the intuitive result that if the path segment abruptly changes direction, the trajectory is more likely to locally diverge off the path (as in waypoint 2 and 4). The control engineer, during the detailed design phase, might consider adding more waypoints the closer the path gets

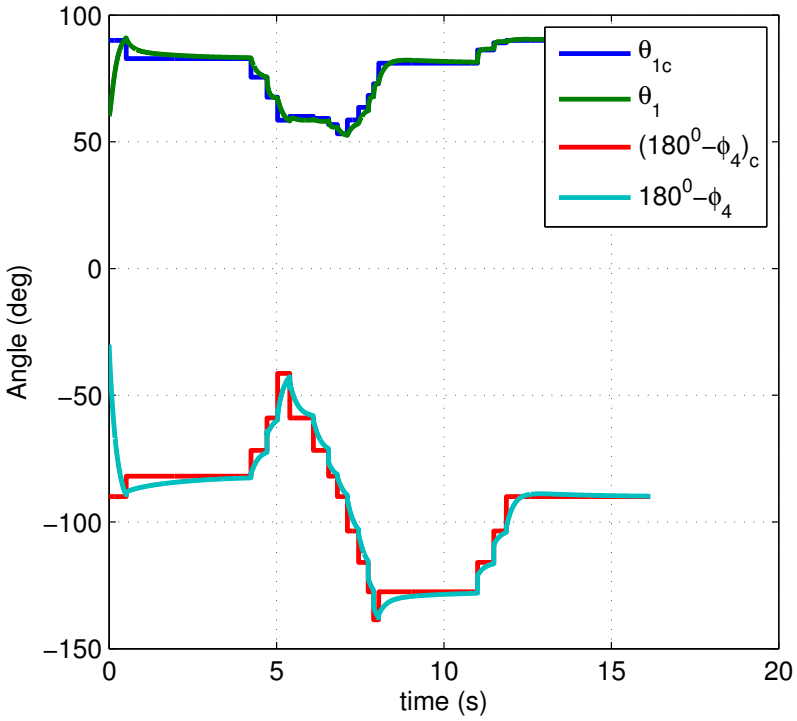


Figure 79: Bar Angles - Segment Defined by Four Internal Waypoints

to the origin, or in segments with abrupt path changes.

Figure 82 presents another important deduction. The angle control on the lower bar motor is more accurate than the one on the upper bar motor. This implies two results. First, the upper bar controller design has to be more stringent, i.e. faster with less steady state error. Using the same controller for the upper and lower bars is not a good design decision. In fact, an independent PID controller for the upper bar may be abandoned for a more sophisticated controller type. Second, the PID control method for the lower bar is sufficient to satisfy the design requirements.

As with previous cases, the lower error, shown in Figure 83, results in a lower magnitude torque. The error hardly goes over 0.1 ft. The mechanism is slower than before, completing the path in about 18 seconds. Therefore, designing a fast and accurate mechanism are two conflicting design requirements.

Figures 84, 85 and 86 show the results for 19 internal waypoints on each path segment. This amounts to a total of 80 waypoints along the path. The mechanism

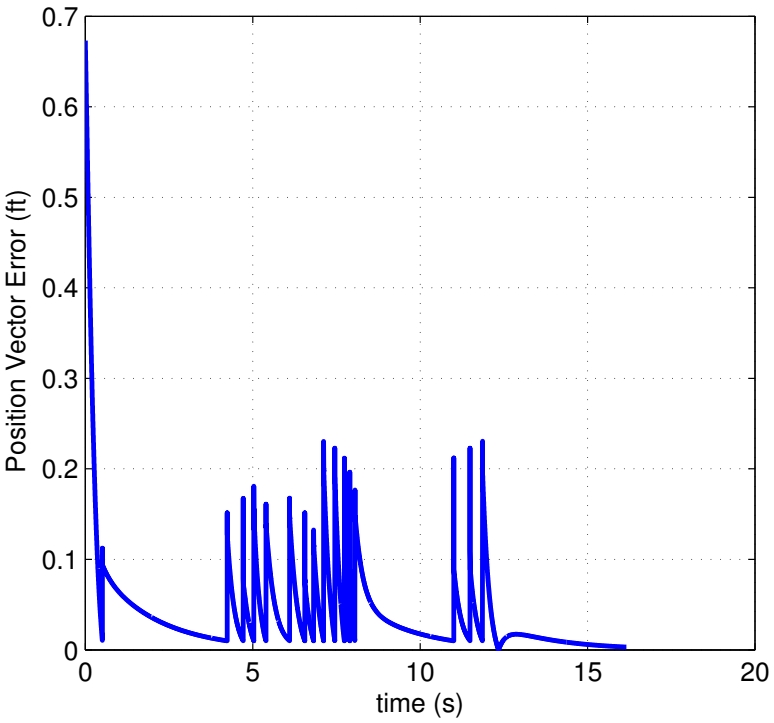


Figure 80: Position Vector Error - Segment Defined by Four Internal Waypoints

closely follows the path for almost all its trajectory. The error hardly goes over 0.05 ft. However, it takes the mechanism about 22 seconds to complete the path.

The control architecture demonstration provided more detail on the control architecture design, and the possible trade offs. Increasing the number of waypoints, or alternately increasing the controller sampling, results in a more accurate system, but slows down the performance. The two conflicting design requirements drive the system towards a compromise.

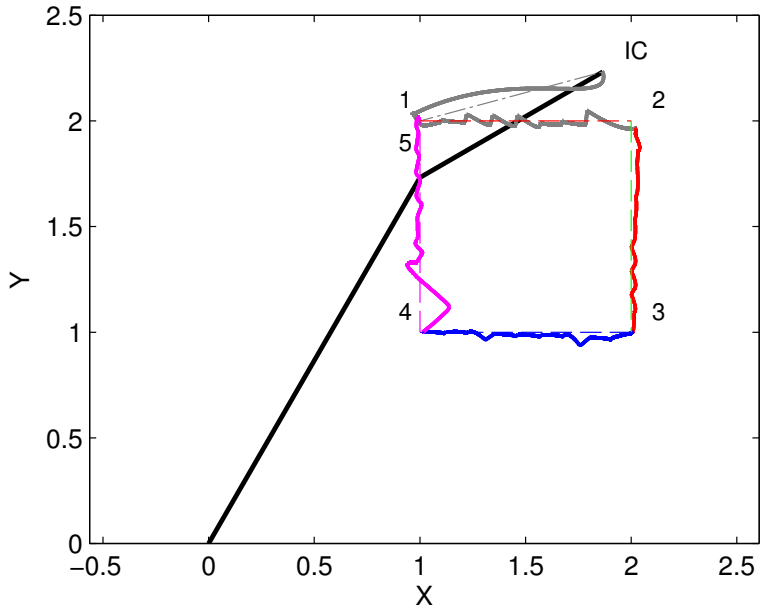


Figure 81: Trajectory - Segment Defined by Nine Internal Waypoints

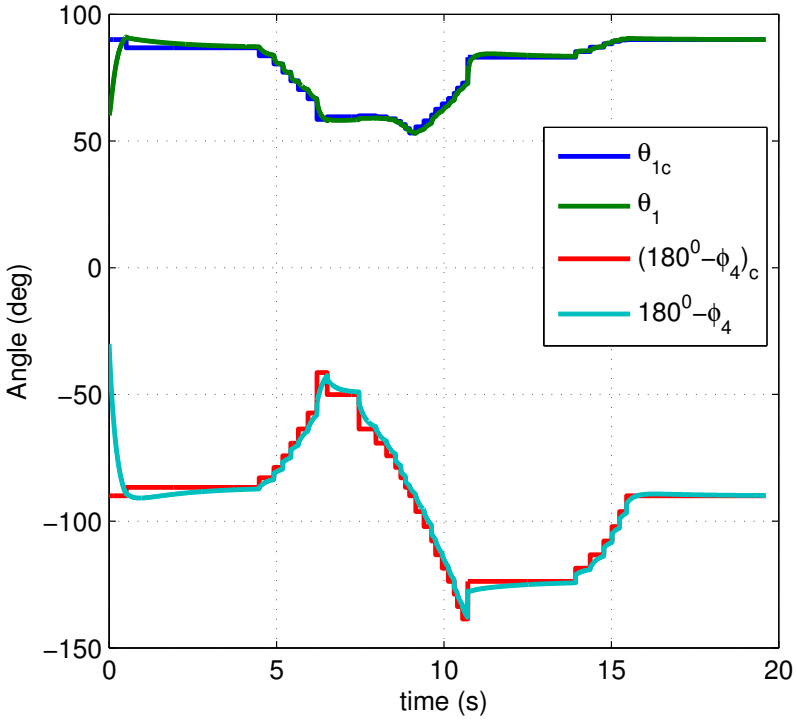


Figure 82: Bar Angles - Segment Defined by Nine Internal Waypoints

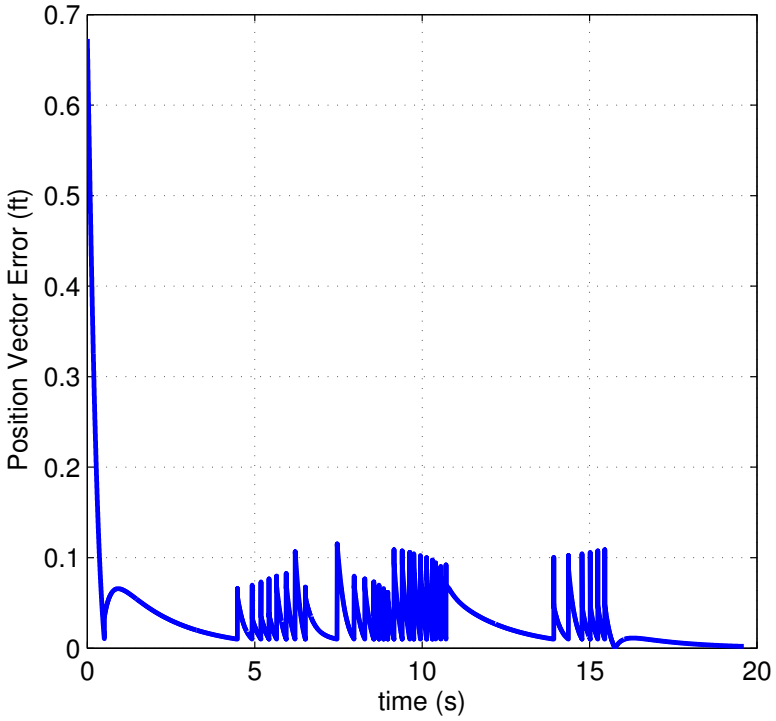


Figure 83: Position Vector Error - Segment Defined by Nine Internal Waypoints

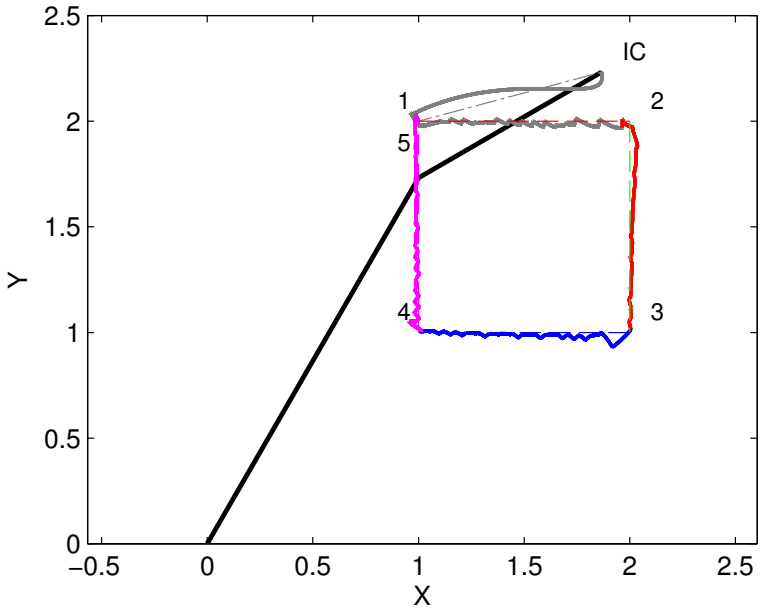


Figure 84: Trajectory - Segment Defined by Nineteen Internal Waypoints

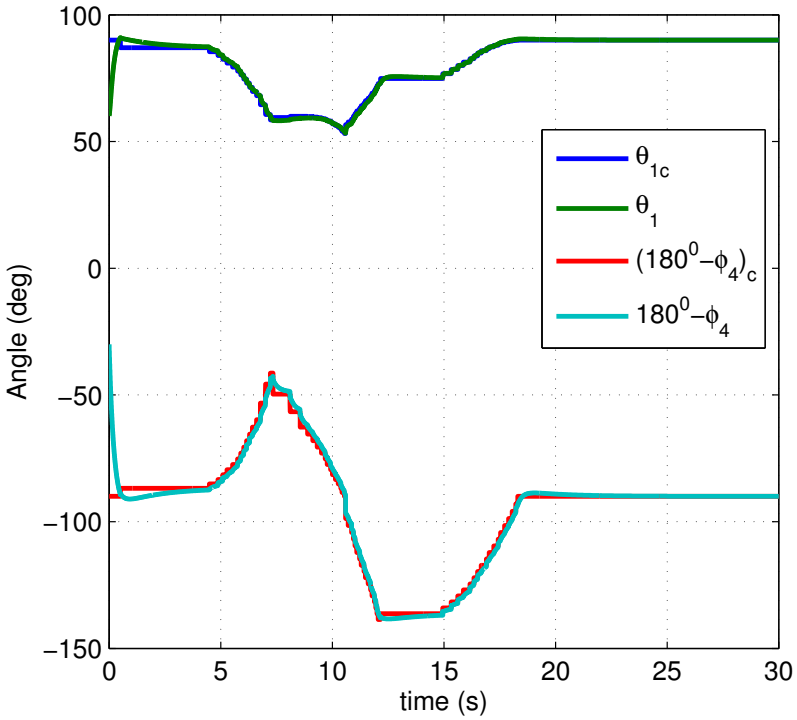


Figure 85: Bar Angles - Segment Defined by Nineteen Internal Waypoints

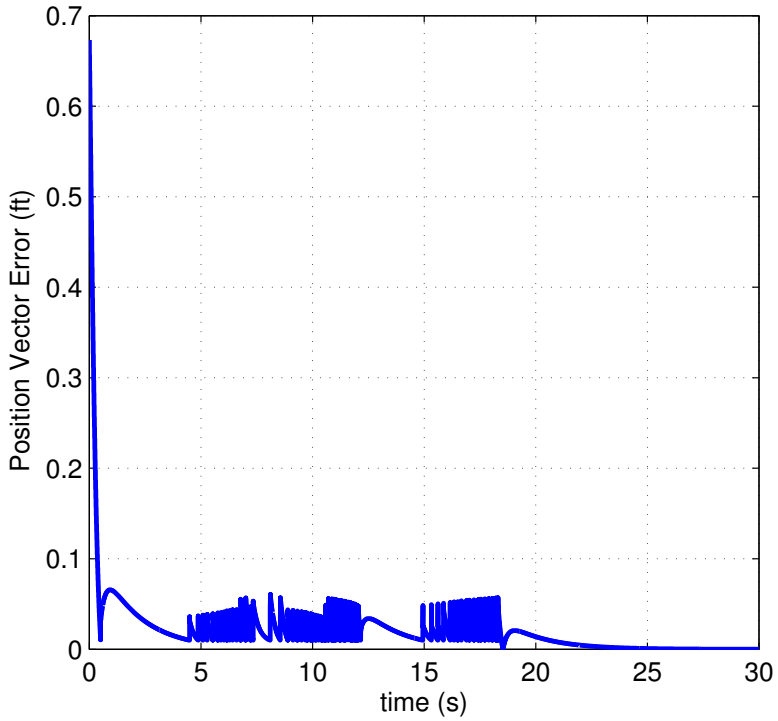


Figure 86: Position Vector Error - Segment Defined by Nineteen Internal Waypoints

CHAPTER VII

APPLICATION: CONTROL ARCHITECTURE DESIGN OF A MULTI-NETWORK COMPLEX SYSTEM

7.1 Introduction

Next generation naval ship design and operation comprises and increased demand for affordability, ship ship survivability and mission effectiveness [4] [86]. The Office of Naval Research proposed an *Integrated Engineering Plant* concept to as a solution to achieve such design requirements [53]. A naval ship is composed of a collection of highly complex interdependent systems, including electrical systems, propulsion systems, thermal systems, fluid systems and weapon systems among others. The integrated engineering plant concept challenges the traditional walls separating various subsystems, and attempts at removal of these barriers such that ship subsystems can share resources, information management systems and be unified in an integrated control architecture [142].

Nevertheless, traditional design methodologies, in which the ship hull is designed, followed by the independent design of different subsystems, and eventually leading to the design of one integrated control architecture, is rendered ineffective for such requirements. The *Integrated Reconfigurable Intelligent Systems*(IRIS) is an initiative proposed by the *Aerospace Systems Design Laboratory* at the Georgia Institute of Technology as a solution to the Integrated Engineering Plant problem to satisfy the requirements of the next generation naval ship [85]. One of the three major features of the IRIS solution is intelligence, which is effective across the whole system of systems. This intelligent control architecture enables the sharing and reassignment of

resources, the mission phase specific system reconfiguration, the raised level of survivability, the maintaining of the ship capability (or slight degradation thereof) under adverse conditions and a more optimal performance of the overall ship systems.

The control architecture analysis suite described in this research is applied to an IRIS like system. The objective is to determine an appropriate level of automation at each level of the IRIS subsystems, how the subsystem intelligent controllers interact, and their hierarchical distribution. As explained in Section 2.5, analytical methods fall short when tackling large scale complex systems. A simulation based engineering approach is in order, through utilizing behavioral models.

7.1.1 Behavioral model

A representative scaled down ship model is given in [112]. The model is further described in more detail in [133], which extends the model as a test bed for different control systems. The simulation testbed has been utilized in several research endeavors such as [121] which attempts at using information theory in controller design. The research presented in [113] explores the system's ability to reconfigure its resource delivery network using the same test bed. Two major ship subsystems are modeled in the test bed, namely a fluid network and the associated DC electric network, in addition to the necessary simulation engine drivers. Simulink (with embedded Matlab functions) was primarily used to program the two main networks. No communication network was modeled, however extensibility accommodations (in terms of necessary component states and physical properties) are available for adding a communication system.

The test bed* is utilized as a basis for building an extended Framework for Control Architecture Testing and Evaluation (FCATE). A thermal system is modeled and

*The author would like to thank Dr. Frank Ferrese, Naval Surface Warfare Center, Carderock Division, for providing the *Control System Testbed* [133] simulation model, and allowing its use for the purpose of this research. The testbed was developed by Nathan Rolander among others at Johns Hopkins University.

added to the basic framework. Modifications are applied to the electric network such that it models the behavior of the components rather than the transient state values. FCATE is used to test several control architectures targeted at reconfiguring the system in case of failure of some of its components. Several control architectures are implemented, tested and compared.

7.2 System Behavioral Model Description

7.2.1 Fluid Network

The chilled water cooling network is shown in Figure 87, which has most of the characteristics of a naval ship chilled water distribution system [133]. The fluid network is supplied by four plants (pumps), designated with "**P**", that supply pressurized chilled water flow to six service loads, designated with "**S**". Cooling the service loads is the main function of the chilled water system. Chilled water flows from the plants along supply lines (internal loop in Figure 87) to the service loads, and back on return lines (outer loop). Flow meters are placed along several branches of the return, designated with "**F**". Certain parts of the fluid network can be isolated such that chilled water supply can be rerouted along alternate routes. This is accomplished through setting the appropriate state of different valves in the system (opened or closed). Valves are designated with "**V**".

The fluid network has four connected zones. These zones coincide with the electric zones described in Section 7.2.2. Each zone is separated from the adjacent zone by two isolation valves (shown with a textured background in Figure 87), one on the supply line and another on the return line. Not all zones are connected to each other: a cyclic connectivity is implemented. Zone 1 is connected to Zone 2 and Zone 3, but not to Zone 4; Zone 2 is connected to Zone 1 and Zone 4, but not to Zone 3, and so on. The isolation valves are controlled by the zone controller, and are energized by the zone power connection. Hence, to isolate a particular zone, it has to close its

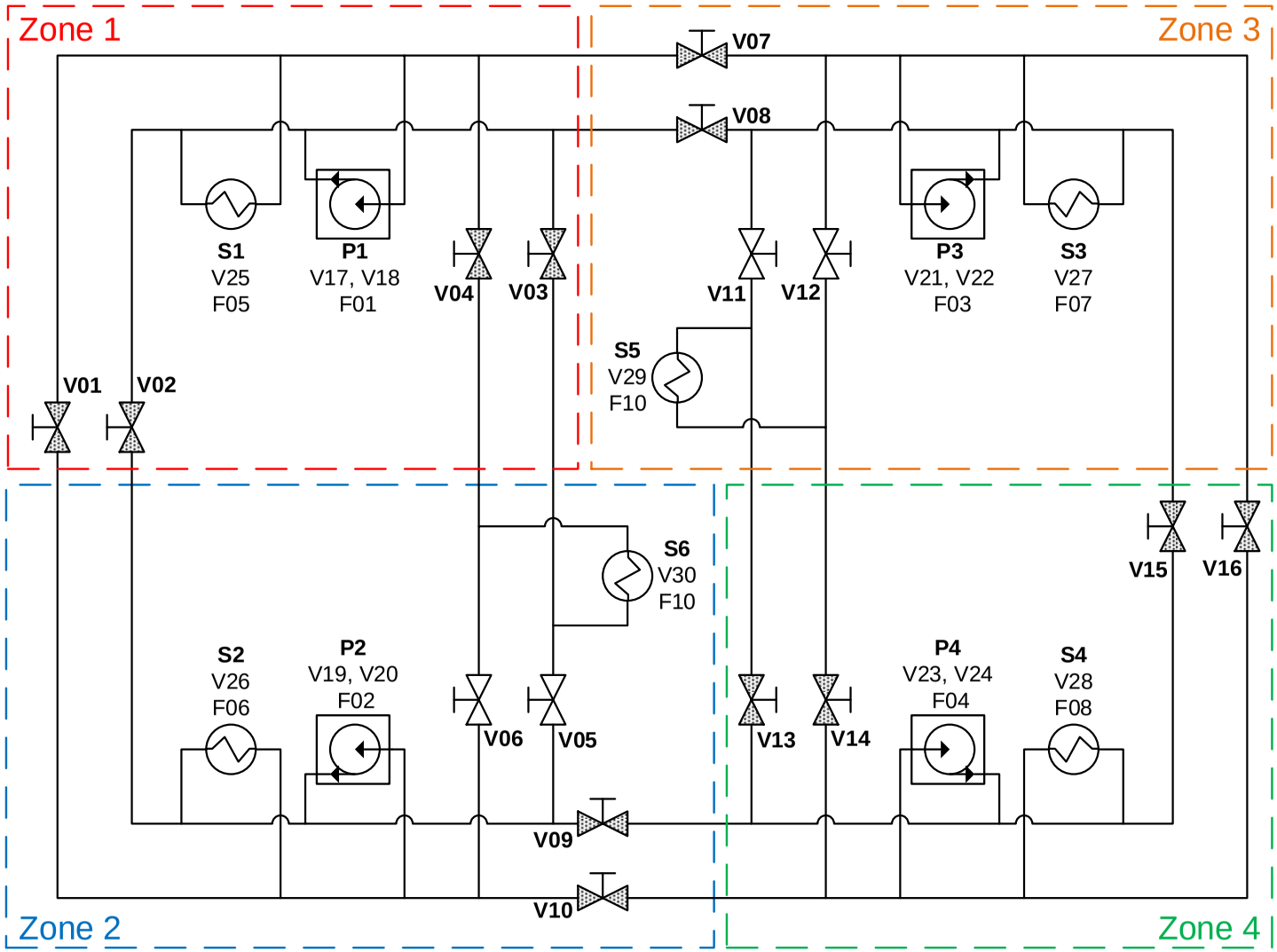


Figure 87: Fluid Network

isolation valve pair and request the adjacent zone to do the same with the other pair of valves. For example if Zone 1 is to be isolated, Zone 1 has to close V01 and V02 (which are controlled and energized by Zone 1), and Zone 3 has to close V07 and V08.

Figure 88 depicts a schematic of the fluid network plant. Each plant consists of pump, two valves to isolate the pump, a pressure gauge to monitor the pump pressure and a flow meter to measure the pump volumetric flow rate. The pair of valves are used to isolate the pump in case of failure, but not for flow control. They are energized by the zonal electric connection. Valves consume electric energy only when the spindle moves (i.e. when the valve either opens or closes), but not when the valve maintains its previous state. Valve power consumption is fixed. The volumetric flow rate is used by the control architecture as a feedback signal to assess the pump performance. The pressure drop across the flow meter is assumed negligible. On the other hand, the pressure gauge signal is not utilized by the control architecture, yet is available for future use.

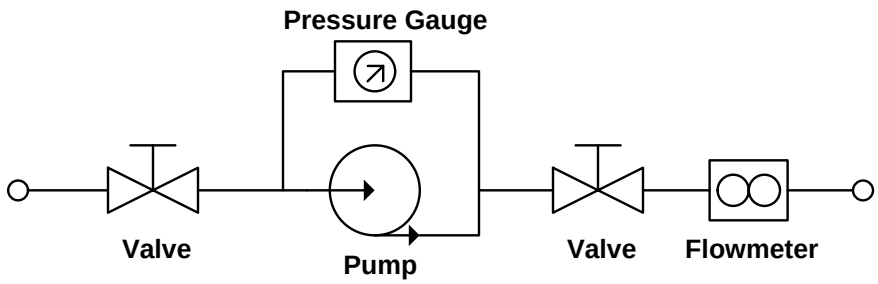


Figure 88: Fluid Network Plant

The pump is powered by an electric motor that is also energized by the zonal electrical connection. The pump power consumption behavior is characterized by a pressure versus current relationship shown in Figure 89. The pump operates at the set pressure of 1405 KPa (20 psi), which can be externally adjusted. It is assumed

that the pump has an embedded controller that is able to maintain the pressure setting regardless of the operating conditions. The electric power consumption is calculated by determining the current using Figure 89, which is used in conjunction with the electric bus potential to calculate the pump impedance (resistance). The pump impedance and current values are propagated up the electric bus supplying the zone. The flow meter and valves power consumption are also calculated in the same way.

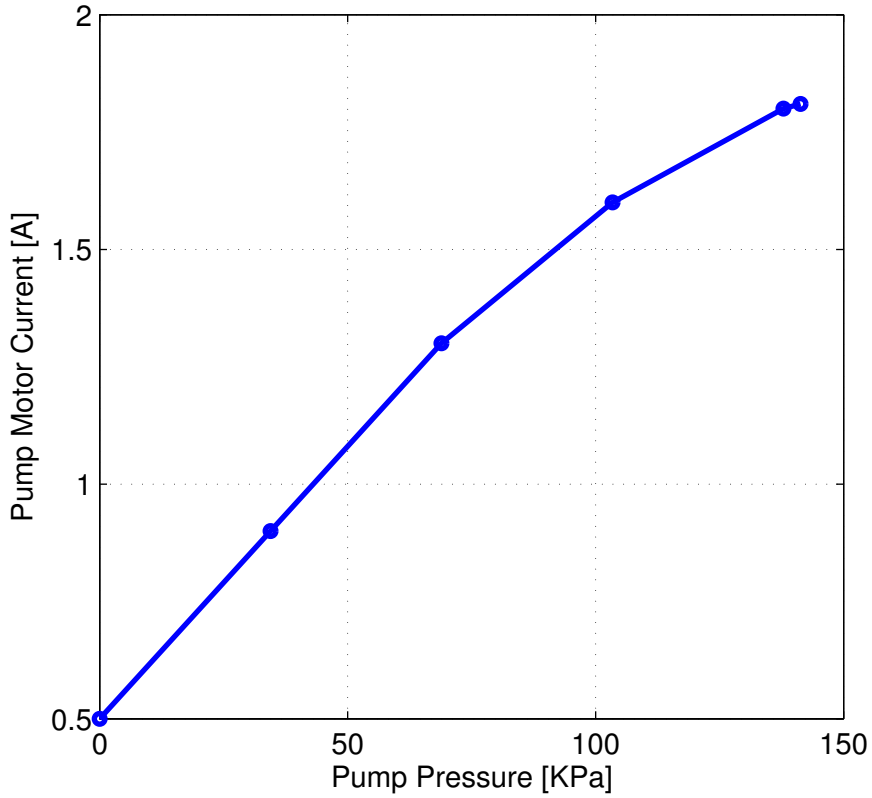


Figure 89: Pump Electric Power Characterisitics

A schematic of the service load is shown in Figure 90. A valve controls the chilled water flow rate entering the service. Unlike the pump isolation valves and zone isolation valves, the service valve is not an on/off valve, but rather a flow control valve. A simple ball valve type is chosen. Chilled water passes through the valve to enter a compact heat exchanger cooling coil (addressed in 7.2.3) at which heat

transfer occurs between the hot service components high temperature and the chilled water low temperature.

The cooling coil fluid resistance is modeled by a square orifice, such that the flow rate through the coil is proportional to the pressure drop. The square orifice has a preset area and discharge coefficient that cannot be changed during a simulation run. The pressure drop is measured across the cooling coil, however it is not used by the control architecture. A flow meter monitors the volumetric flow rate through the service and transmits this signal to the controller. The service, valve and flow meter are powered through the zonal electric connection. Similar to the plant components, the service load components' power consumption is calculated by calculating the individual impedance of components.

The chilled water network contains six service loads. Two of the six loads are designated as "vital loads": they have to be maintained within operable limits to avoid system failure. The rest of the services are designated "non-vital loads": failure of these loads degrades the performance of the system but does not lead to system failure. S1, S2, S3 and S4 are non-vital loads while S5 and S6 are vital loads. Zone 1 contains S1, Zone 4 contains S4, Zone 2 contains S2 and S6 while Zone 3 contains S3 and S5. This implies that it is vital to ensure the operation of Zones 2 & 3 to avoid system failure.

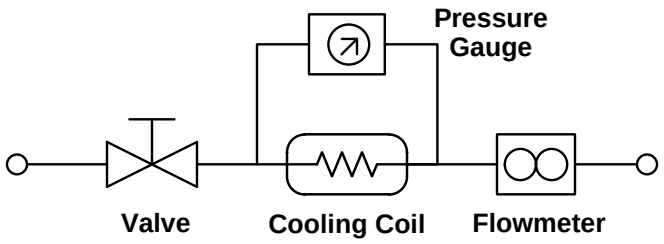


Figure 90: Fluid Network Service (Load)

The main task of the control architecture is to ensure that the service loads'

temperatures are always maintained within certain operable limits. The system is designed to supply all six services during normal operation. However, in case of damage or limited resources, the fluid network control architecture decides on which services have higher cooling priority, route chilled water to these services, and ensure their temperatures are within operable limits. The control architecture

7.2.2 Electric Network

The electric network represents a scaled down version of a typical ship electric system. It serves as the main power provider for the energized valves, flow meters, pumps, service loads, and other system components. The network is energized by two 12V power supplies. There are two main buses on the electric network, which model the port and starboard buses on a naval ship. The two power supplies are linked to the two buses through four relays. This configuration enables each power supply to provide power to one or both buses depending on the relays' states. The high level electric network is shown in Figure 91.

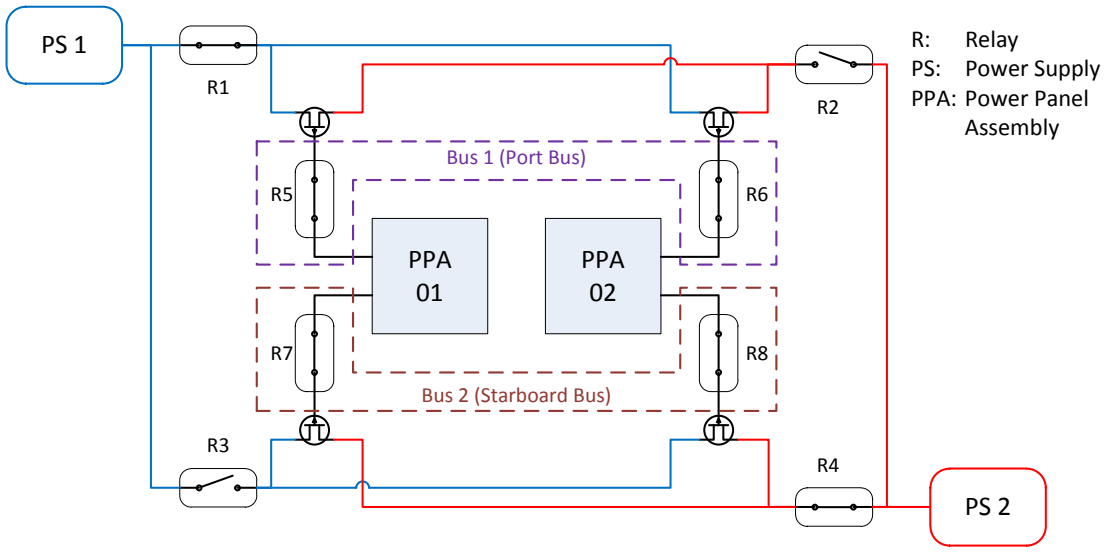


Figure 91: Electric Network - High Level

The two buses are connected to two power panel assemblies using four relays. Each

power panel assembly can be powered by both buses by coupling their connection through a junction point (refer to Figure 92). A junction point takes two upstream electric connections as inputs and outputs one electric connection to a downstream component. The output is a function of the physical properties of the two input electric signals. Several functions can be used to model the electric junction, such as a balance between the two connections, the connection supplying less loads, or other. The function can be arbitrary chosen priori to simulation. The intelligence/control architecture can be modified to control thus function. For example, the controller might decide to draw electric power from a specific connection that is more reliable or with a lower importance. The current implemented function outputs the electric signal from the bus with the higher voltage:

$$S_3 = f\left(S_1, S_2\right) = \max\left(V_1, V_2\right) \tag{20}$$

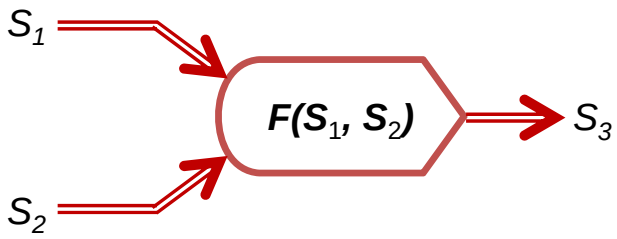


Figure 92: Electric Junction

Both Buses are connected to two Power Panel Assemblies (PPAs) using four relays. This relay array enables the the buses to independently (and redundantly) power the two power panels assemblies, hence providing a fully reconfigurable electric system. The PPAs' details is shown in Figure 94. Each power panel assembly consists of four Power Panels (PPs), such that each panel is linked through a junction to two relays, and ultimately to the two buses. The junction function is given by 20, similar to the junctions between the power supplies and buses. In total, the electric system has

twenty four relays.

The system has four electric zones. Each zone is energized through two power panels, one from each power panel assembly. The relay logic is shown in Figure 93 while the panel physical connectivity is shown in Figure 94.

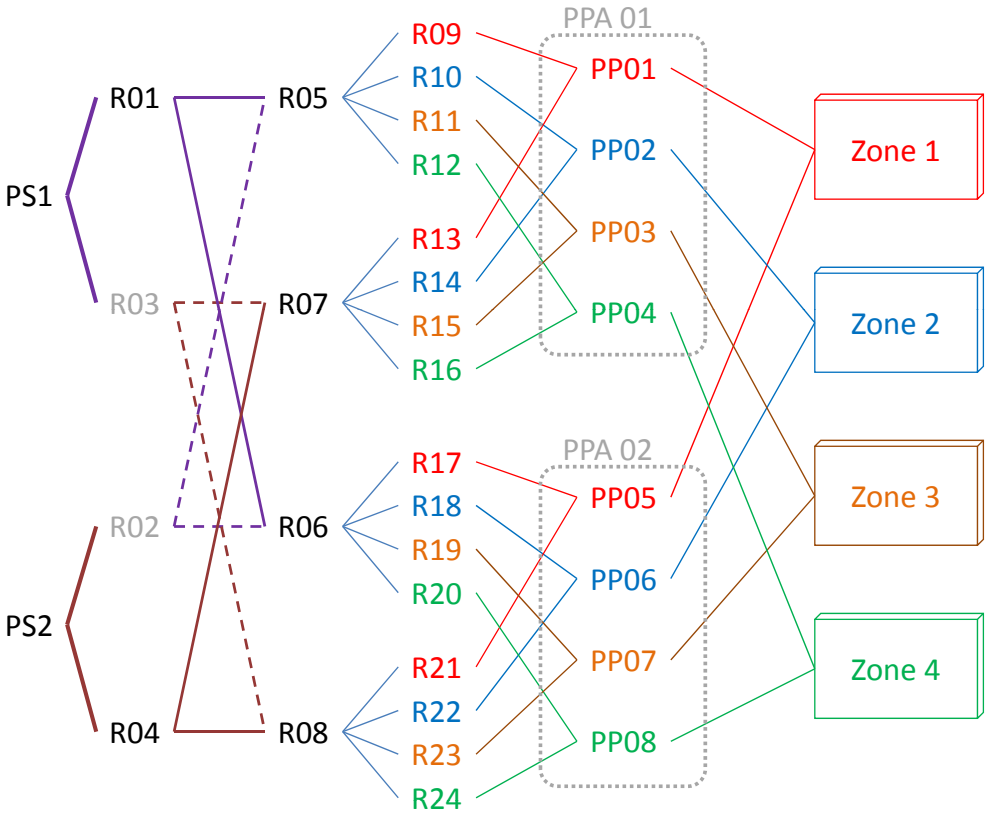


Figure 93: Electric Network - Relay Logic

7.2.3 Thermal System

Every service load heats up during operation, therefore requires adequate cooling. The air surrounding the service also heats up accordingly. It is assumed that the service temperature is the same as the air temperature (an equilibrium is reached). Cooling the service is done indirectly by cooling the surrounding air. Chilled water from the fluid network enters the heat exchangers and passes through a finned coil. Forced air flow is drawn from the air surrounding the service and passed over the

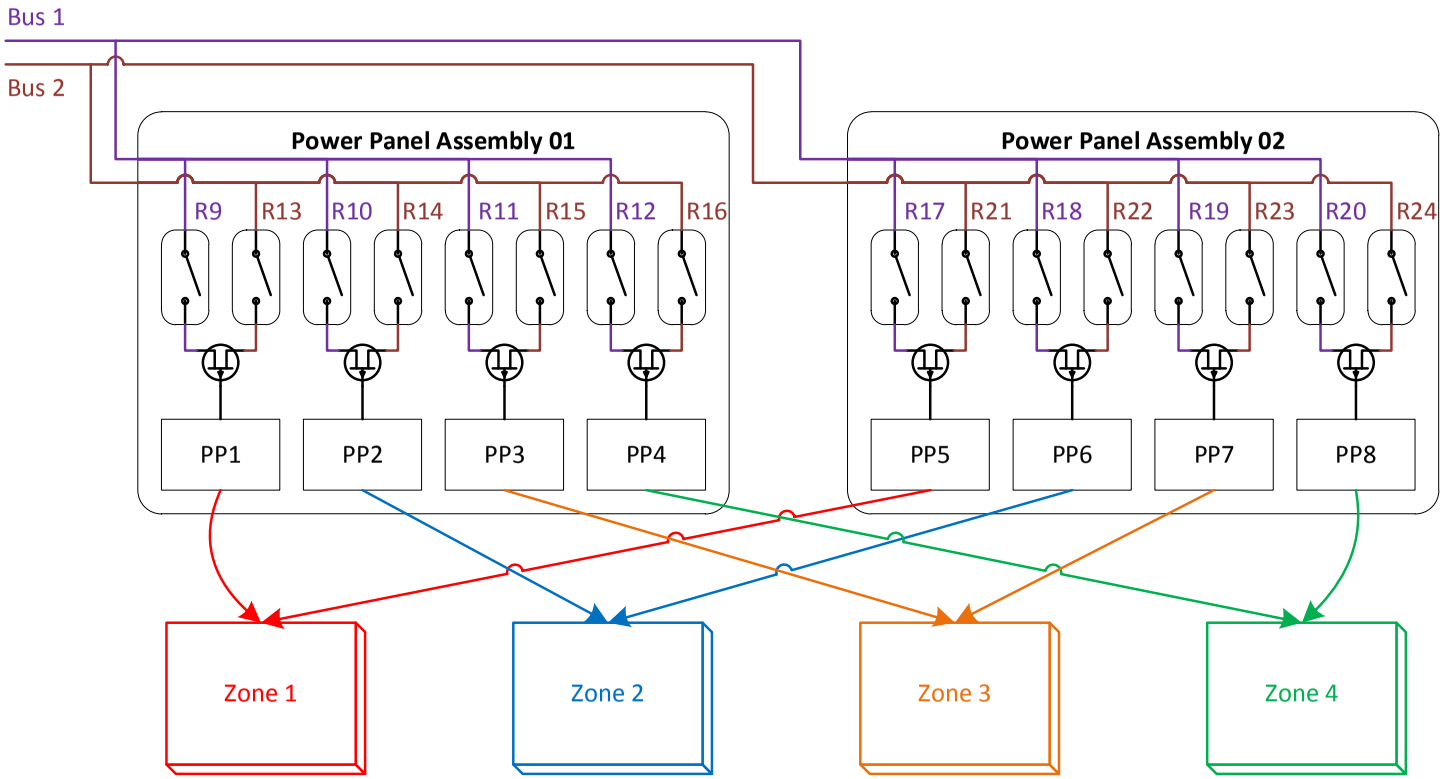


Figure 94: Electric Network - Power Panel Assemblies Details

cooling coils so that it may lose heat, and back to the service. The schematic of the compact heat exchanger is shown in Figure 95. Note that air is blown through the duct using a fan to enter the exchanger so that the more efficient forced convection occurs.

The thermal system consists of six compact heat exchangers, each attached to a service load. The heat exchanger air fan is powered by the same zonal connection that powers the service. Failure of the service does not imply failure of the exchanger, however failure of the zonal electric connection results powering down both the service and the heat exchanger.

The amount of heat (per unit time) a service produces is treated as an external input to the simulation model, in the form of a schedule. Compact heat exchanger design is usually based on the maximum amount of heat that the heat exchanger handles. Design procedures are outlined in the literature, for example [124], for various heat exchanger types. Other approaches to heat exchanger design are based on a graphical approach such as in [63]. Most heat exchanger design algorithms are industry owned, making them not readily available, hence [105] attempts at proposing an alternative design approach. [20] presents two methods of design and analysis of heat exchangers. The first method depends on the use of the log mean temperature difference. For a parallel flow type heat exchanger, the heat transfer across the exchanger can be given by:

$$q = U\,A\,\Delta T_{lm} \tag{21}$$

where

$$\Delta T_{lm} = \frac{\Delta T_2 - \Delta T_1}{\ln\left(\Delta T_2 / \Delta T_1\right)} \tag{22}$$

such that:

q : Total rate of heat transfer between the cold and hot fluids

U : Overall heat transfer coefficient

A : Heat transfer surface area

ΔT_{lm} : Log-mean temperature difference

ΔT_1 : Temperature difference between the two fluids at input

ΔT_2 : Temperature difference between the two fluids at output

Counter flow heat exchangers have similar relationships, however ΔT_1 is defined at the temperature difference between the two fluids at the "hot" fluid input, and ΔT_2 is defined at the temperature difference between the two fluids at the "hot" fluid output.

The log-mean temperature difference method is simple to use if the fluid temperatures are known at the inputs, and are readily at the output (such as from an energy balance, or know boundary conditions). If the inlet temperatures are the only knowns, a computationally expensive iterative approach is necessary to determine the outlet temperatures. In addition, this method is suitable for direct and counter flow heat exchangers, but not for more complex heat exchanger types. Hence, a different method is introduced for modeling the compact heat exchanger behavior, namely the Number of Transfer Units (NTU) method.

The heat service simulation model is shown in Figure 96. Next, the NTU method is discussed, followed by outline of the numerical approach to calculating the service temperature, and a brief presentation of the air circulating fan ans shown in Figure 95.

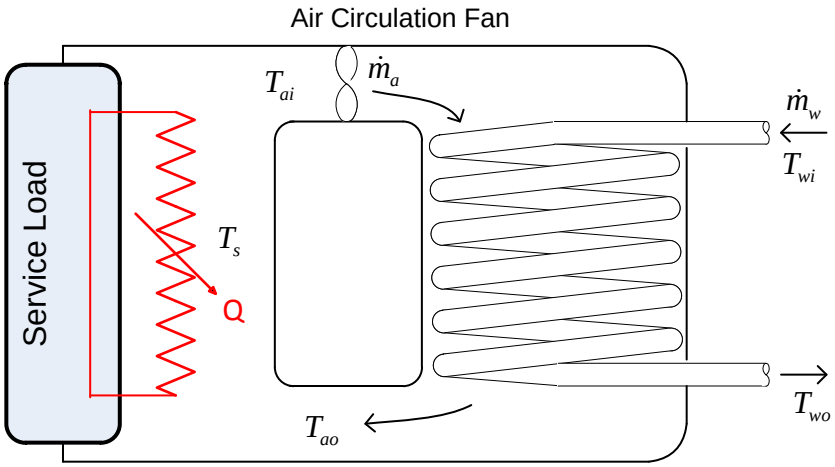


Figure 95: Compact Heat Exchanger

7.2.3.1 The Effectiveness NTU Method

The NTU method is based on defining the effectiveness of the heat exchanger. This enables the analysis of different types of heat exchangers using mathematical relations. To define the effectiveness of a heat exchanger, the maximum possible heat transfer rate (occurring for a counter flow heat exchanger of infinite length) has to be determined. Defining:

$$C_c \triangleq \dot{m}_w C_{Pw} \tag{23}$$

and

$$C_h \triangleq \dot{m}_a C_{Pa} \tag{24}$$

such that:

$\dot{m}_w$: Chilled water mass flow rate

$\dot{m}_a$: Air mass flow rate

C_{Pw} : Chilled water specific heat (assumed constant)

C_{Pa} : Air specific heat (assumed constant)

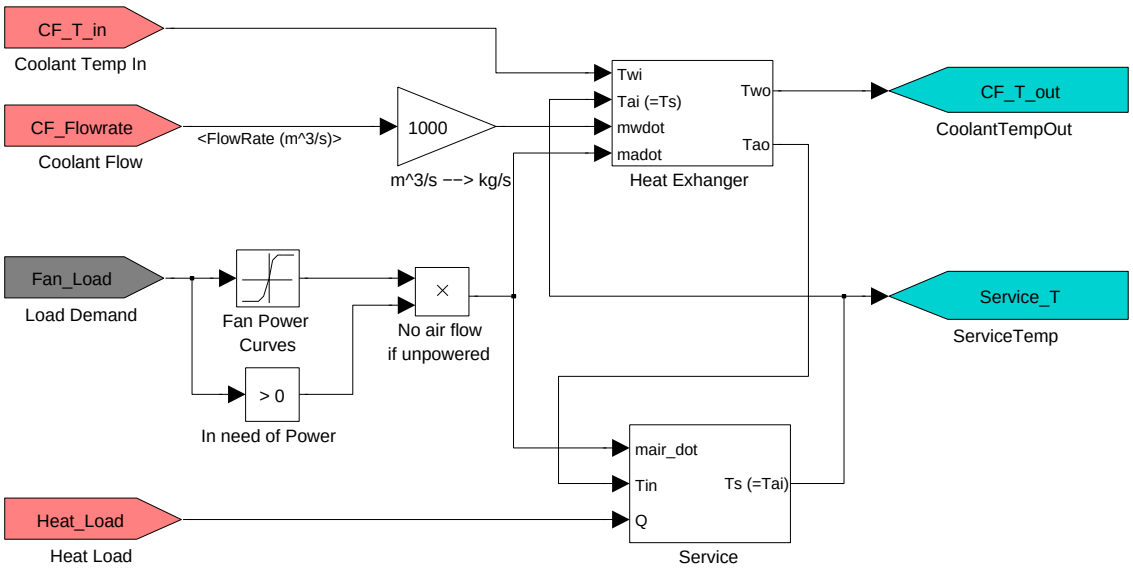


Figure 96: Thermal Service Simulation Diagram

If the chilled water experiences the larger temperature change between the inlet and outlet(as compared to the air flow), the maximum heat transfer rate, $q_{\max}$ can be calculated from

$$C_c < C_h : \quad q_{\max} = C_c(T_{ai} - T_{wi}) \tag{25}$$

If the air flow experiences the larger temperature difference, the maximum heat transfer rate can be calculated from

$$C_h < C_c : \quad q_{\max} = C_h(T_{ai} - T_{wi}) \tag{26}$$

Defining $C_{\min}$ and $C_{\max}$ as

$$C_{\min} \triangleq \min(C_c, C_h) \tag{27}$$

$$C_{\max} \triangleq \max(C_c, C_h) \tag{28}$$

The maximum heat transfer can be expressed in the form

$$q_{\max} = C_{\min}(T_{ai} - T_{wi}) \tag{29}$$

The effectiveness of the heat exchanger, ε, is defined as the ratio of the actual heat transfer, q, to the the maximum heat transfer, $q_{\max}$

$$\varepsilon \triangleq \frac{q}{q_{\max}} \tag{30}$$

The actual heat transfer of each fluid can be calculated from their enthalpy relations. Hence it is possible to calculate ε from

$$\varepsilon = \frac{C_h(T_{ai} - T_{ao})}{C_{\min}(T_{ai} - T_{wi})} \tag{31}$$

or

$$\varepsilon = \frac{C_c(T_{co} - T_{wi})}{C_{\min}(T_{ai} - T_{wi})} \tag{32}$$

By definition, the effectiveness has to be a value in the range $0 \leq \varepsilon \leq 1$. If ε is known, together with the inlet conditions of both fluids, the actual heat transfer can be calculated from

$$q = \varepsilon C_{\min}\left(T_{ai} - T_{wi}\right) \tag{33}$$

For any type of heat exchanger, it can be shown that [77]

$$\varepsilon = f\left(\mathrm{NTU}, C_r\right) \tag{34}$$

such that

$$C_r = \frac{C_{\min}}{C_{\max}} \tag{35}$$

The NTU is a dimensionless parameter that is widely used to characterize heat exchanger behavior. It is defined by:

$$NTU = \frac{U\,A}{C_{\min}} \tag{36}$$

The function in Equation 34 depends on the type of heat exchanger. It is assumed that the thermal system's heat exchangers is of the cross flow single pass type, with both flows unmixed. Thus the heat exchanger effectiveness relationship is given by [77]

$$\varepsilon = 1 - \exp\left[\left(\frac{1}{C_r}\right)\, NTU^{0.22}\, \left\{\exp\left(-C_r\, NTU^{0.78}\right) - 1\right\}\right] \tag{37}$$

Relationship 37 is plotted for several values of C_r in Figure 97. All heat exchanger types (for any C_r) have approximately the same effectiveness for NTU$\lessapprox 0.2$. Maximum values of effectiveness is associated with $C_r = 0$, while minimum effectiveness is associated with $C_r = 1$. Care should be taken in the case in which $C_r = 0$ to avoid a division by zero. Other relationships can be given for other types of heat exchangers, and present the same trends.

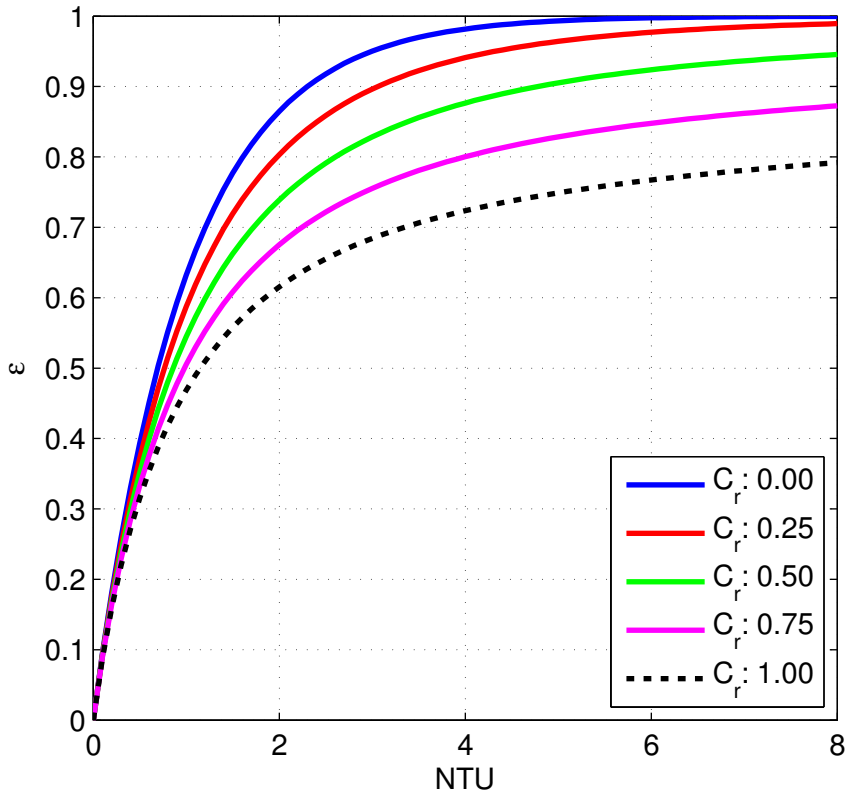


Figure 97: NTU Vs Effectiveness - Cross Flow Single Pass - Both Fluids Unmixed

To summarize the NTU method: C_c, C_h, $C_{\min}$, $C_{\max}$ and C_r are calculated from 23, 24, 27, 28 and 35. NTU is then computed from 36. Either equation 37 or Figure 97 is used to determined to the value of ε. Note that changing the analysis to include a different heat exchanger type only requires a change in this equation. The maximum heat transfer rate, $q_{\max}$ is then calculated from 29, followed by the actual heat transfer, q, from 30. It is assumed that the heat exchanger has negligible heat

loss to the surrounding, hence all the heat lost by the hot air is transfered to the chilled water. The actual heat transfer is equal to

$$q = \dot{m}_a \, C_{Pa} \, (T_{ai} - T_{ao}) \tag{38}$$

for the air, and

$$q = \dot{m}_w \, C_{Pw} \, (T_{wo} - T_{wi}) \tag{39}$$

for the chilled water. Given T_{wi} and T_{ai}, and using equations 38 and 39, the chilled water output temperature, T_{wo}, and the air output temperature, T_{ao}, are computed. The air input temperature, T_{ai}, is assumed to be the same as the service temperature, T_s. The Simulink diagram for the heat exchanger model is shown in Figure 98.

7.2.3.2 Service Temperature Calculation

The air within the service load cavity is blown through a duct to flow over the chilled water cooling coil. The air temperature falls and re-enters the service cavity to mix with the hot temperature air resulting in an overall cooling effect. The final temperature of the service is assumed to be equal to the average air temperature in the cavity, T_s. A control volume is assumed within the service air cavity to conduct thermal analysis. The energy entering the control volume (in the form of cooled air flow and heat generated by the service) is balanced by the energy stored in the control volume and the energy leaving the control volume (in th form of hot air), i.e. an energy balance analysis is performed. The energy entering the control volume, Q_{in}, is given by

$$Q_{in} = \dot{m}_{a-in} \, C_{Pa-in} \, T_{a-in} + Q \tag{40}$$

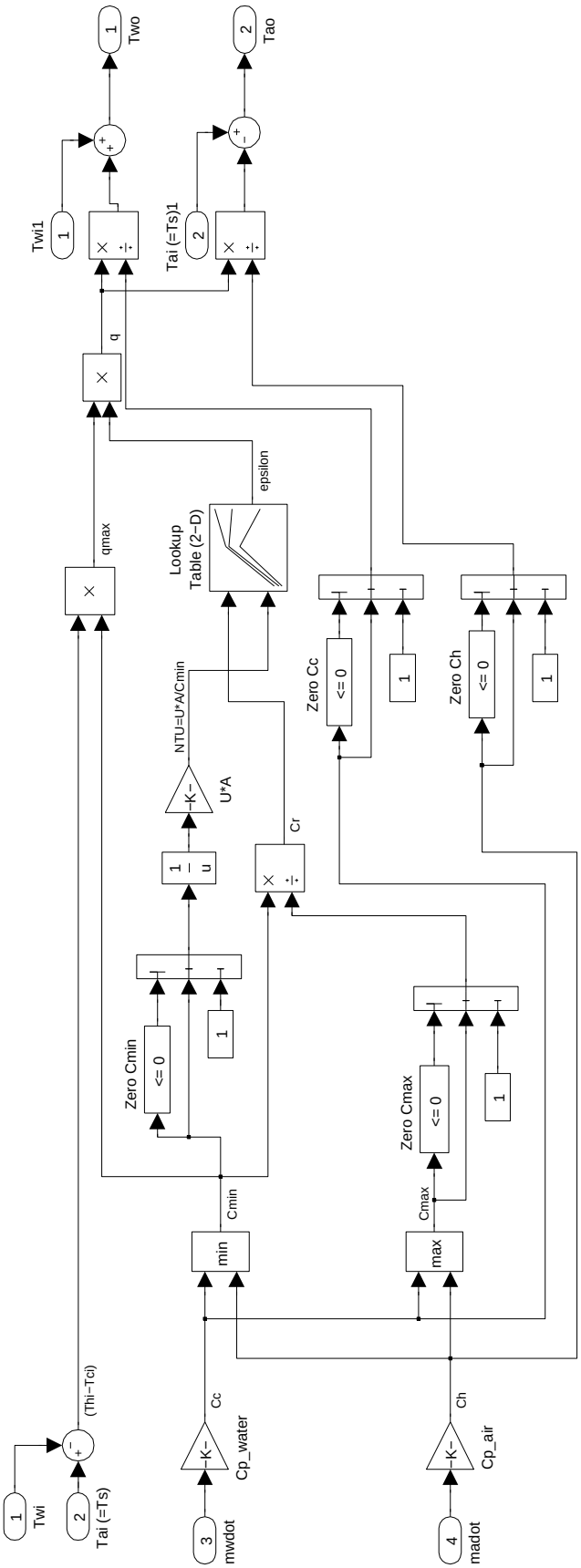


Figure 98: Heat Exchanger Simulink Model

such that

$\dot{m}_{a-in}$: Air mass flow rate entering the service (exiting the heat exchanger)

C_{Pa-n} : Air specific heat at constant pressure

T_{a-in} : Air temperature entering the service (exiting the heat exchanger)

Q_{in} : Thermal energy rate added to the service

The energy exiting in the control volume, Q_{out} is given by

$$Q_{out} = \dot{m}_{a-out}\, C_{Pa-out}\, T_{a-out} \tag{41}$$

such that

$\dot{m}_{a-out}$: Air mass flow rate exiting the service (entering the heat exchanger)

C_{Pa-out} : Air specific heat at constant pressure

T_{a-out} : Air temperature exiting the service (entering the heat exchanger)

It is assumed that the the air temperature does not change much to cause a change in the air specific heat constant. Hence

$$C_{Pa-in} = C_{Pa-out} = C_{Pa}$$

Also, the average temperature of the service is equal to the outlet air temperature T_{a-out}, which in turn is the same as the heat exchanger input temperature, T_{in}. In addition, the service inlet temperature is equal to the heat exchanger exit temperature. Therefore

$$T_s \;=\; T_{a-out} \;=\; T_{ai} \quad \text{and} \quad T_{a-in} \;= T_{ao}$$

It is also assumed that no compression takes place making the control volume input and output mass flow rates, and the heat exchanger input and output air mass flow rates equal.

$$\dot{m}_{a-in} \;=\; \dot{m}_{a-out} \;=\; \dot{m}_a \tag{42}$$

The stored energy in the control volume, Q_{stored}, can be expressed as

$$Q_{stored} \;=\; \rho V C_{Va} \frac{dT_{a-out}}{dt} \;=\; \rho V C_{Va} \frac{dT_s}{dt} \tag{43}$$

such that

$$\rho \;:\; \text{Air density}$$

$$V \;:\; \text{Service air cavity volume}$$

$$C_{Va} \;:\; \text{Air specific heat at constant volume}$$

Combining equations 40, 41 and 43 yields

$$\begin{aligned} Q_{in} &= Q_{out} + Q_{stored} \\ \dot{m}_a C_{Pa} T_{ao} + Q &= \dot{m}_a C_{Pa} T_s + \rho V C_{Va} \frac{dT_s}{dt} \end{aligned} \tag{44}$$

Equation 44 is integrated in time to compute a value for T_s at each time step, using inputs from the heat exchanger from the previous time step. The above equations are solved / integrated in a Simulink model, shown in Figure 99.

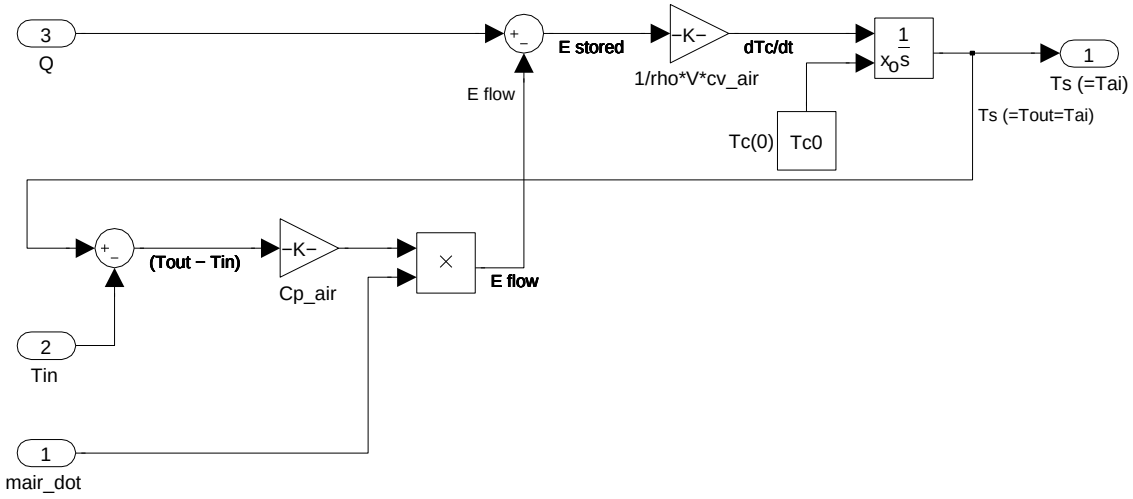


Figure 99: Service Cavity Simulation Model

7.2.3.3 Air Circulating Fan Model

A fan (mounted inside the service duct) forces the air to circulate between the service cavity and above the water cooling coils. The fan is powered through the same power

source providing for the service. A simple behavioral numerical model that relates the fan load demand [Watts] to the air mass flow rate $\dot{m}_a$ $[kg/s]$ is assumed. The model is linear, but can be replaced by any off-the-shelf fan characteristics. The linear numerical model is shown in Figure 100.

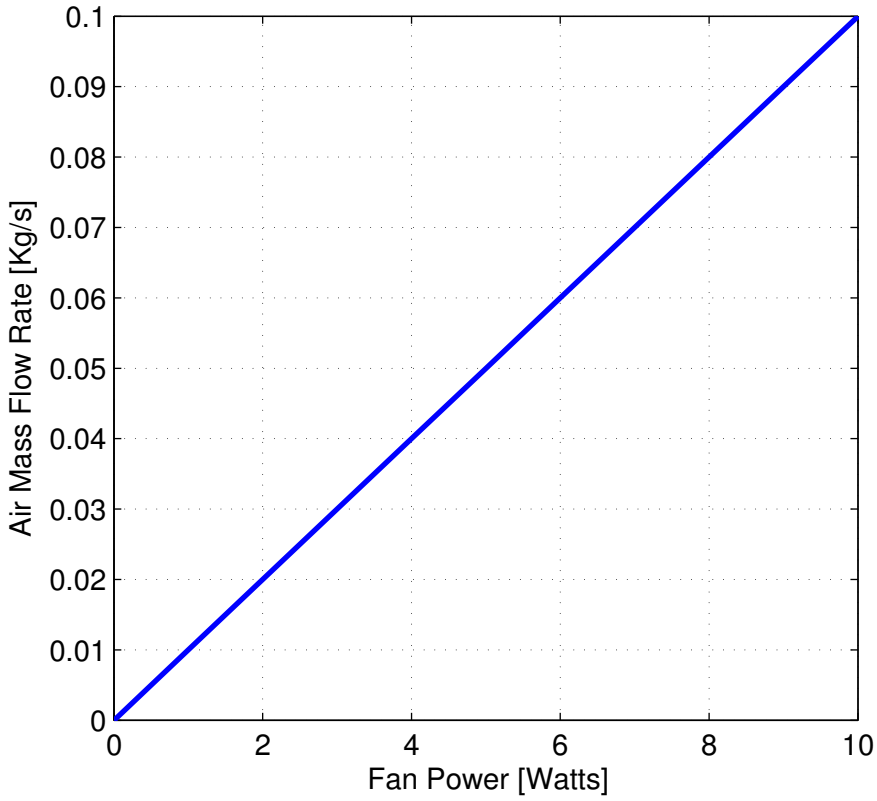


Figure 100: Air Circulating Fan Model

7.3 Naval Vessel Control Architecture Analysis Suite

The previous section described a naval ship simplified simulation model. The model can be categorized as a low-mid fidelity behavioral model that adequately describes the behavior of the systems from a functional perspective. The main physics based relationships among system states and parameters are embedded in the model.

In the current section, the analysis suite described in 4.7 is applied to the behavioral model. The objective is to reveal more information on the limits imposed by the control architecture choice on the physical system's behavior, and the effect of

the physical system's design parameters on the control architecture feasible options. This objective is accomplished by conducting four types of analyses, namely: state reachability, inverse dynamics, stability and capability potential analyses.

However, basic controller implementation is required to explore the interaction between the control architecture and the physical system. To limit the scale of the design problem, it is assumed that the overall integrated system (physical and control architecture) has two controllable inputs:

1. The cooling fluid (water) flow rate. Four pumps control the amount of cooling fluid circulating through cooling network.

2. A compact heat exchanger is attached to each thermal service. The heat exchanger assembly consists of a cooling coil connected to the cooling network, ducts, and a cooling fan circulating air between the service and the cooling coils. The operating power to the fan is a controllable input, such that higher power results in more air being circulated and therefore better cooling.

Therefore, a local switching controller is implemented on each thermal service. The controller switches the cooling fan on and off based on the current service temperature. The controller is a bang-bang controller, meaning that it does not command a control action proportional in value to the service temperature feedback signal. Instead, it commands either an "'on"' or an "'off"' signal to the cooling fan. The maximum fan power counts as a design parameter, set in the beginning of the simulation. Figure 101 shows the operation of the fan controller. The controller is engaged if service temperature rises above 315^oK, and disengages when the temperature is less than 310^oK. As a final assumption, unstable system performance results if the service temperature exceeds 320^oK.

The cooling fluid pump capacity is another design parameter. Ideally, another controller should control the pump speed and output. Nevertheless, the concentration

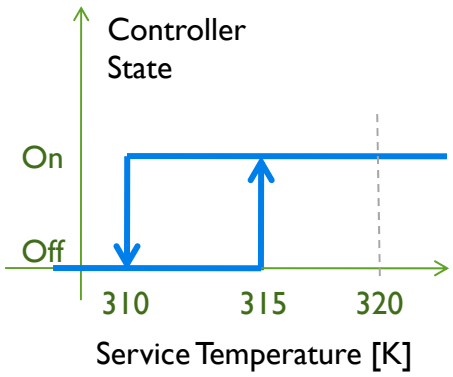


Figure 101: Air Circulating Fan Controller

in the context of this research is the service temperature. Hence only the *behavior* of such a controller is modeled, such that the "'virtual pump controller"' is responsible for maintaining a given flow from a specific pump. The pump capacity is preset in the beginning of the simulation and is assumed constant and maintainable throughout.

7.3.1 State Reachability

The main function of the cooling network is to maintain the thermal service temperature within a certain acceptable range. The first and foremost design analysis is to assess the level of achievement of this function. The operating conditions as well as the physical systems and control architecture design choices together affect the system's final state.

Reachability analysis requires that an ideal "'best case"' controller is assumed. Therefore, the cooling fan is operated at maximum power of 10 W. Four different cooling flow magnitudes through a service load are explored, namely 0.2, 0.6, 1.0 and 1.4×10^{-5} m^3/s. Note that these are not necessarily equal to pump capacities, since one pump can be used to supply different service loads. Figures 102, 103, 104, 105 and 106 show the time histories of the service temperature for heat loads of 50, 100, 150, 200 and 300 W respectively. The limits for acceptable operation ($310^oK \leq T_s \leq 315^oK$) and the stability limit ($T_s \leq 320^oK$) are also shown on the

figures.

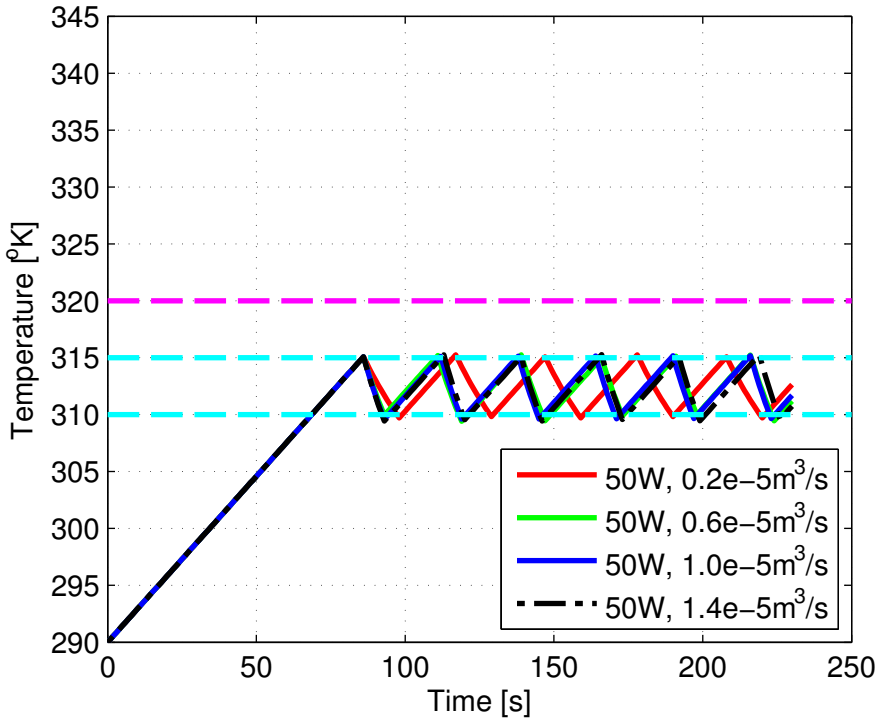


Figure 102: Service Temperature Response - 50W Heat Load

Lower heat loads (50 and 100 W) are always manageable by the cooling system, for all cooling flow magnitudes considered. However, extremely low cooling flow result in a slower reduction in service temperature. In addition, for lower heat loads, it is observed that increasing the cooling flow beyond a certain limit hardly causes any change in the service temperature response. For example, the temperature time history for 0.6, 1.0 and 1.4×10^{-5} m^3/s cooling flows in the 50 W case are almost identical (refer to Figure 102).

Increasing the heat load to 150 W (Figure 104) results in acceptable performance, but only for higher cooling flows. The lowest cooling flow value of 0.2×10^{-5} m^3/s results in a constant temperature outside the acceptable limits, but less than the stability limit. In this case, the controller is always engaged, sending an "'On"' signal to the fan to operate at maximum power, but the cooling is not enough to drive the temperature back to within the acceptable limits. It is up to the designer to assess

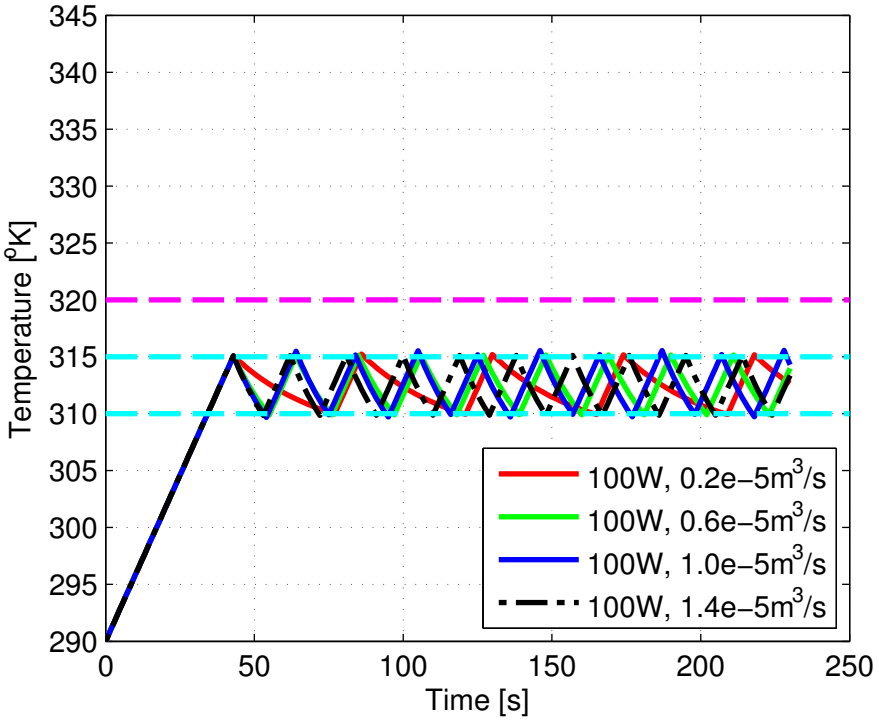


Figure 103: Service Temperature Response - 100W Heat Load

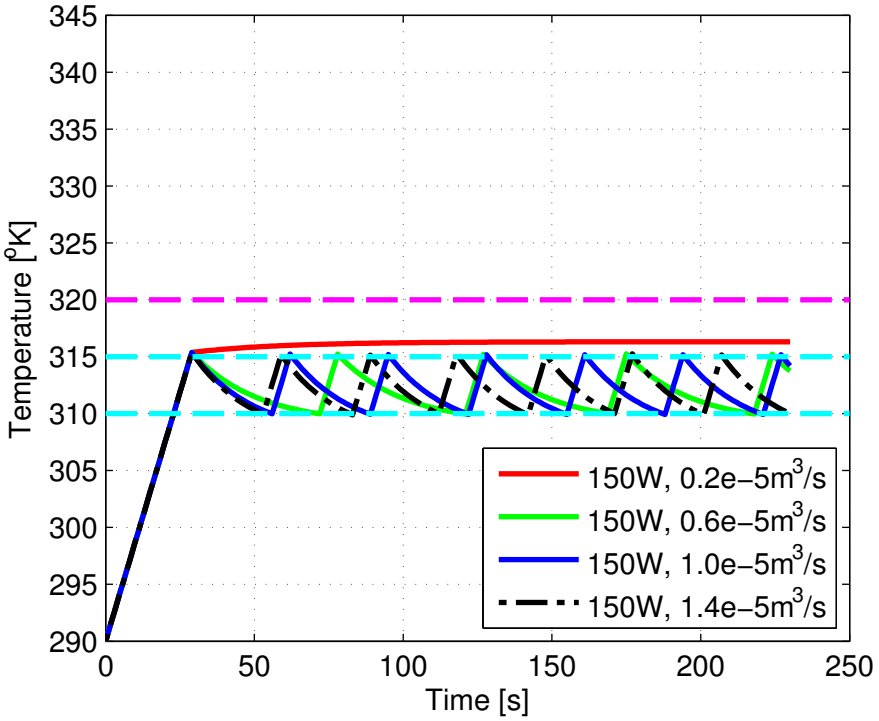


Figure 104: Service Temperature Response - 150W Heat Load

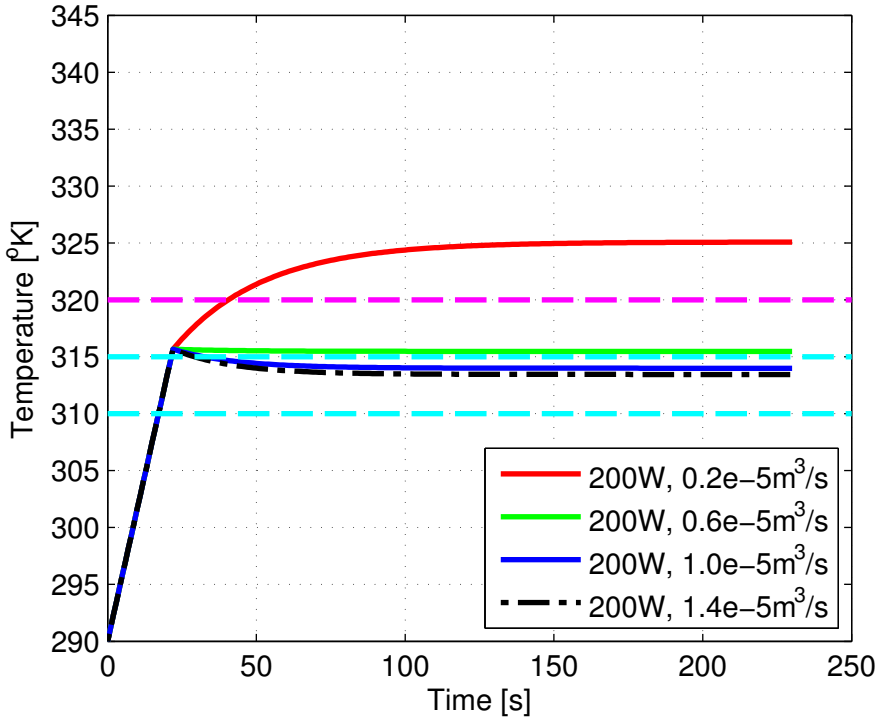


Figure 105: Service Temperature Response - 200W Heat Load

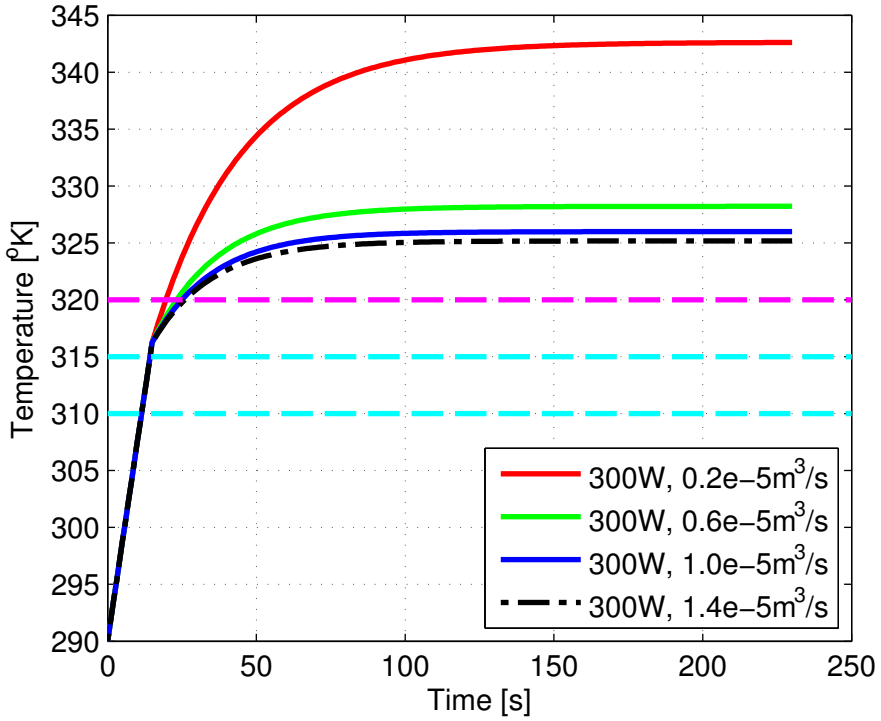


Figure 106: Service Temperature Response - 300W Heat Load

the acceptability of such cases.

A similar situation occurs when the heat load is increased to 200 W (Figure 105). Higher cooling flow rates are enough to maintain a constant temperature, but not enough to decrease it. At 300 W (Figure 106), the cooling flow, within the assumed range, is not enough to keep the temperature within the stable limit.

It is important to reemphasize that reachability analysis was conducted assuming the control architecture is operating at its best/maximum capacity, i.e. an air fan power of 10 Watts. Accordingly, if any improvement in state reachability is sought, the physical system's design parameter choices have to be readdressed.

7.3.2 Inverse Dynamics

The second type of analysis aims at assessing the resources needed to drive the system to its required final state. In the general case, the resources are equivalent to the generalized driving forces. However, in the case of a controller switching an air circulating fan, the resources can be viewed in the context of the fan energy consumption.

The fan operates at a given constant power specified at the beginning of the simulation. This power is assumed as a parameter in the inverse dynamics analysis. The fan power ranges between 1 and 10 Watts.

The fan energy consumption is a direct integration of the fan power only if the fan is switched on for the full duration of the simulation. However, the presence of a hysteresis band (temperature lower and upper limits for acceptable performance) affects the state of the fan, resulting in lower energy being consumed in some cases.

Figure 107 shows the different cases of 0.2, 0.6, 1.0 and 1.4×10^{-5} m^3/s cooling flows combined with 50, 100, 150, 200 and 300 W heat loads. As expected, the energy consumption increases with the increase in heat load for the same cooling flow. Also, the energy consumption decreases with the increase in cooling flow for the same heat load. Although the curves are very close to being linear, nonlinearity occurs due to

the controller switching the fan states. The lower the heat load, the more times the fan switches states, hence the higher the nonlinearity.

Another observation from Figure 107 is related to how the curves for different cooling flows are organized for a given heat load. The different curves, although slightly nonlinear (for example the 50 W curves), collectively form an almost linear band. Therefore, these curves can be approximated by a linear regression curve within acceptable accuracy. This prompts a simpler controller design, which only takes into consideration the heat load and ignores the cooling fluid magnitude in contrast to taking both into account. The exact controller design cannot be finalized unless the system parameters are. Nevertheless, the result that the local controller can be a *single input single output* controller is an important reduction in controller complexity.

In case of the 200 and 300 W heat loads, the fan switches to the "'On"' state and remains at this state. Hence, irrelevant of the cooling flow, the energy consumption remains the same. Therefore all curves corresponding to different cooling flows for the same heat load coincide, and are linear. This is a typical case where the controller reached its saturation or maximum capacity, driving the system as hard as it can, but the physical limitations of the system prohibits improved performance. In fact, in case of the 300 W heat load, it cannot stabilize the system for any fan power command and cooling flow magnitude.

Although fan switching results in nonlinear behavior, it is a desired feature since it saves energy. Considering the maximum fan operation power of 10 W, Figure 108 shows the domain of the switching controller. Cases in which the final service temperature at the end of the simulation is between 310^oK and 315^oK, and the controller is powerful enough to drive the temperature down, generally occur for lower heat loads and higher cooling flows. The significant result here is that a rough estimate of the range of controller *desirable* operation (switching range) as related to

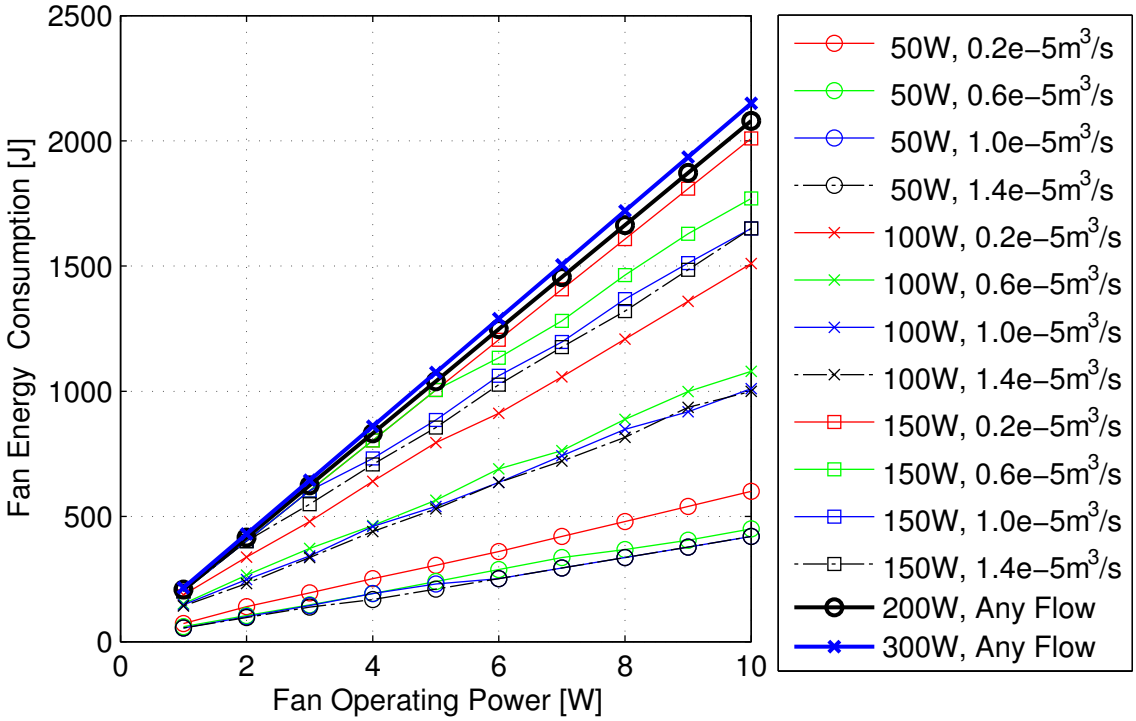


Figure 107: Fan Power Vs. Energy Consumption

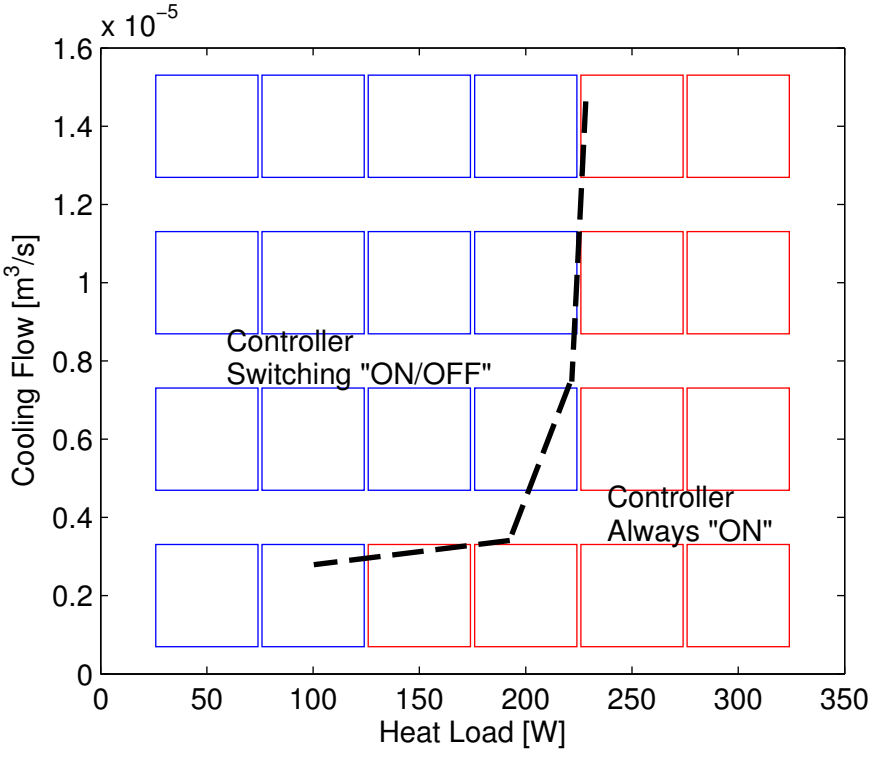


Figure 108: Fan Controller Switching Cases - $10W$ Fan Power

system's operational parameters (cooling flow and heat load) is predicted. The range is defined by the dashed line in Figure 108.

It is important to note that not all cases shown in Figure 107 are stable cases. Also, not all cases shown in Figure 108 for $T_{sf} > 315^oK$ are unstable cases. This is the subject of the following section.

7.3.3 Stability Considerations

To simplify the analysis, the stability condition is assumed to be $T_s < 320^oK$ for stable operation. If the service temperature exceeds this limit, either in its steady state value or during a temporary overshoot, the system is assumed unstable.

Note that traditional definitions of system stability (such as phase or gain margins) do not serve the purpose of this analysis. Traditional stability metrics check if a state becomes unbounded. However, the service temperature in our case never does. However, it is assumed that the temperature increase beyond a certain point results in a series of cascading failure events which eventually drives the whole system unstable. States do become unbounded, but these states are not part of the behavioral model presented. Therefore it suffices, in this context, to describe stability in terms of a temperature limit.

Figure 109 presents the data in Figure 107 from a different perspective. Each point in Figure 109 corresponds to specific system operational parameter set (heat load, fan power, cooling flow magnitude). However, the focus of Figure 109 is on the whether a particular parameter set results in stable (acceptable) operation. It is clear from the figure that the unstable parameter sets (represented by red squares) form an upper boundary on the system's operations parameter set space. The boundary is dominated by the heat load value and the fan operating power.

The stability boundary is more of a gray area than a sharply defined edge. Stable and unstable operating points almost coincide in the vicinity of the boundary region.

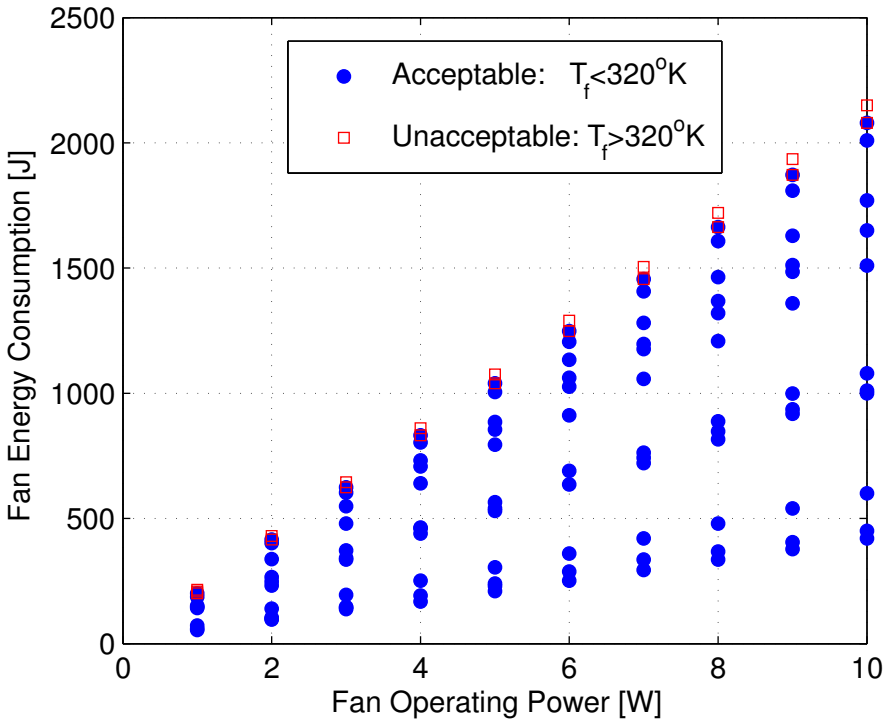


Figure 109: Fan Power Vs. Energy Consumption - Stability

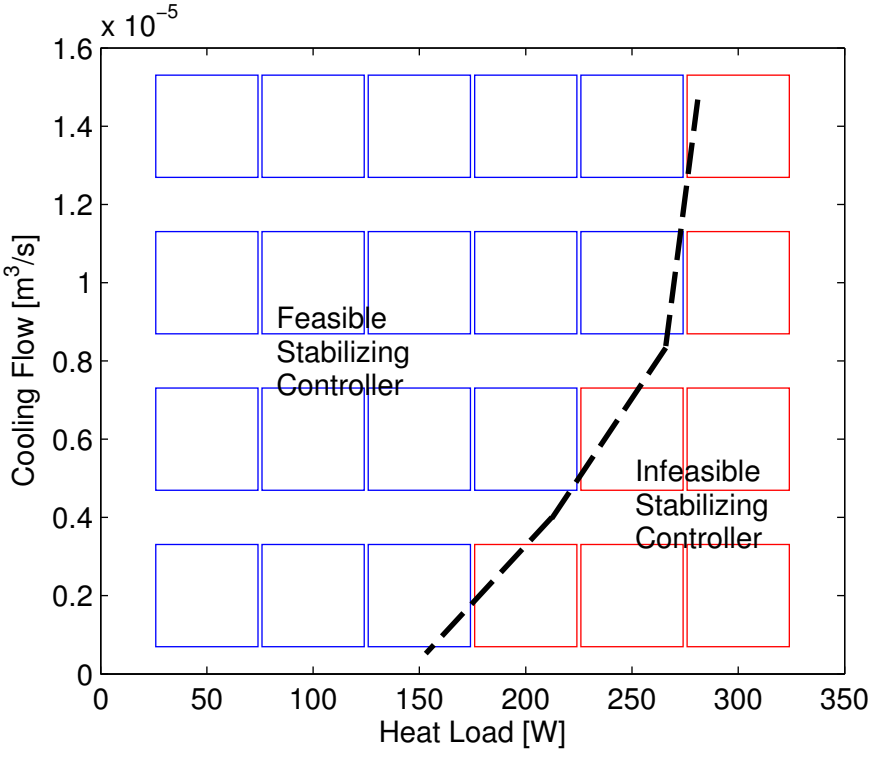


Figure 110: Stabilizing Fan Controller Feasibility Region - $10W$ Fan Power

For example, the operation parameter set consisting of low cooling flow associated with the 150 W heat load is very close to the any flow with higher heat load. This presents itself as a local controller design challenge in the detailed design phase, since the controller has to avoid a loosely defined stability boundary.

Figure 110 shows the operational parameters set for a given fan power, in this case the maximum power of 10 W (the left edge on Figure 109). The feasibility region is defined as the parameter sets that result in a final service temperature of less than 320^oK. The less the heat load and the more the cooling flow, the more likely that the parameter set is in the feasible stabilizing controller region. Note that this feasible region shrinks as the fan power is decreased. Also, comparing Figures 108 and 110, the unstable cases set is a subset of the set of cases in which the fan is always on.

7.3.4 Capability and Performance Potential

Three types of performance criteria are presented in this analysis. The first criterion deals with the system's desired performance, but not necessarily its stability. In the presence of a continuous heat load, the desired service temperature behavior is that it is maintained between 310^oK and 315^oK. The local controller responsibility is to provide for this functionality. Figure 111 shows the region that satisfies the desired performance. Note that this region is smaller than the feasibility region shown in 109, since there are cases where the steady state service temperature is less than the stability limit of 320^oK, but more than the 315^oK upper limit of the desired performance region.

The second and third performance criteria are more on the traditional control design side, namely the rise time and settling time. To conduct this analysis, an initial service temperature of 290^oK is assumed. The local switching controller is hard wired into continuously switching "'on"' the fan, hence driving the system on a linear path. The maximum fan power is varied between 1 W and 10 W; the cooling

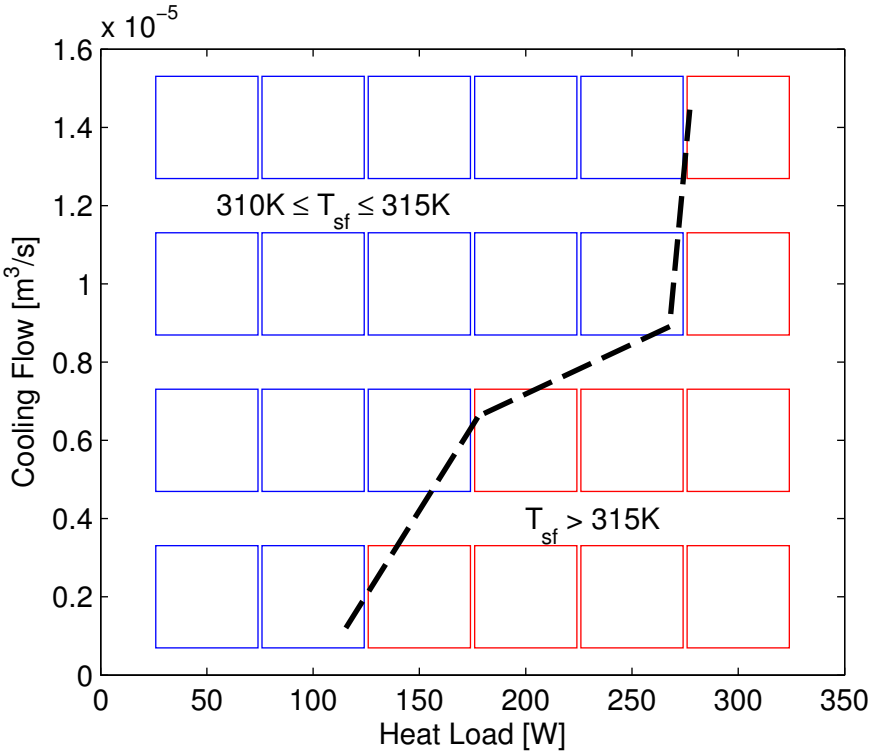


Figure 111: Desired Performance Region - 10W Fan Power

flow is varied between 0.1×10^{-5} m^3/s and 1.6×10^{-5} m^3/s; while the heat load is varied between 50 W and 200 W. Note that the cases of 250 W and 300 W heat loads are not included since these are unstable cases.

The contour plot relating the rise time, cooling flow and fan power for the 50 W heat load case is shown in Figure 112. Similarly, Figures 113, 114 and 115 are plotted for the rest of the heat load cases. As the heat load decreases, the lower rise time region expands. For example, compare the region size for a 40 seconds rise time between the 50 W and 150 W plots.

Another important result presented in the Figures 114 and 115 is the stable regions. The contour plots in both figures are chopped and only the regions with stable performance are shown.

Figures 116, 117, 118 and 119 show the settling time results, and exhibit the same trends as the rise time results. The best achievable performance is a 40 seconds rise

time and an 80 seconds settling time. Since the controller is hard wired as previously explained, then this is the best it can attain. Better performance is only possible by improving the physical system design.

7.4 Naval Vessel Control Architecture Meta-model

The main question asked in Chapter 1 is answered in this section. Which is better: using a control architecture that is broken down based on a spatial decomposition, or a control architecture, structured around functional subsystems? Two architectures would be considered here. The first is a spatial decomposition approach in which each zone is a stand alone unit within the control architecture, such that the zone controller controls all three subsystems.

The second considered architecture is based on a functional decomposition. There are three main controllers in a hierarchy: a thermal system controller, a fluid network controller, and an electric network control. This architecture is a hybrid between the hierarchical and distributed structures.

7.4.1 Control Node Complexity

As explained in Chapter 5, a node is a sub controller within the architecture. Since the hardware is already designed in current design problem, then there is not much the designer can do regarding reduction of the complexity of the hardware components. In other words, a component, such as a valve, has a pre-specified controller that cannot be changed. The objective of this section is to present the internal node structure for each of lower-level sub controllers that are directly associated hardware.

Cyclomatic complexity is used as a guide in the comparison of different architecture options. It should be emphasized that such a metric primarily applies to the software domain. Hence it directly applies to the control node assessment, since a control node almost always boils down to a software program. However, using

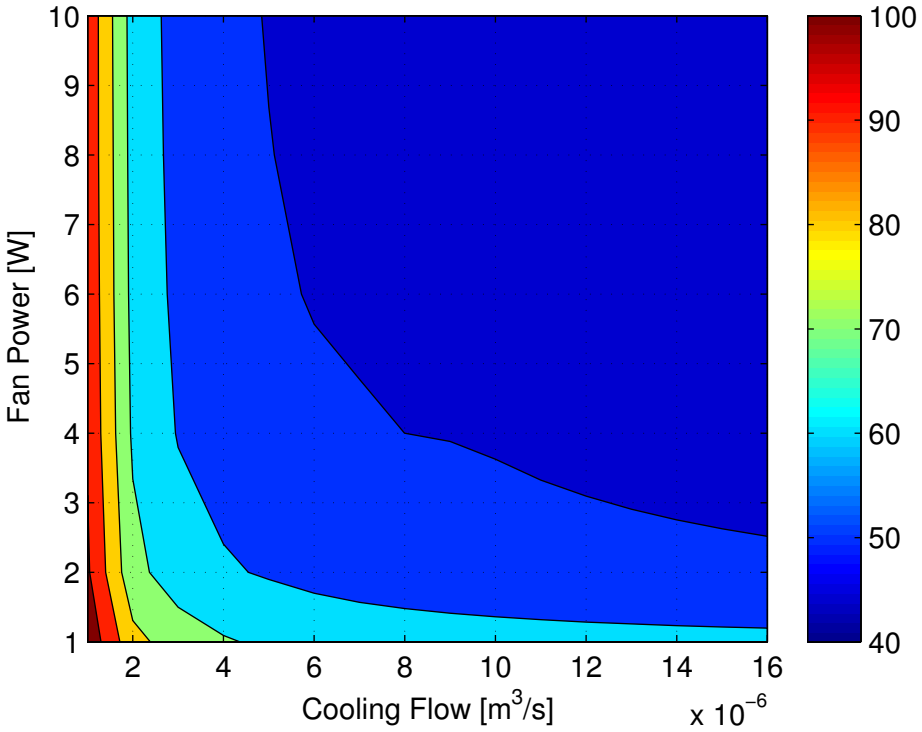


Figure 112: Rise Time[s] - 50W Heat Load

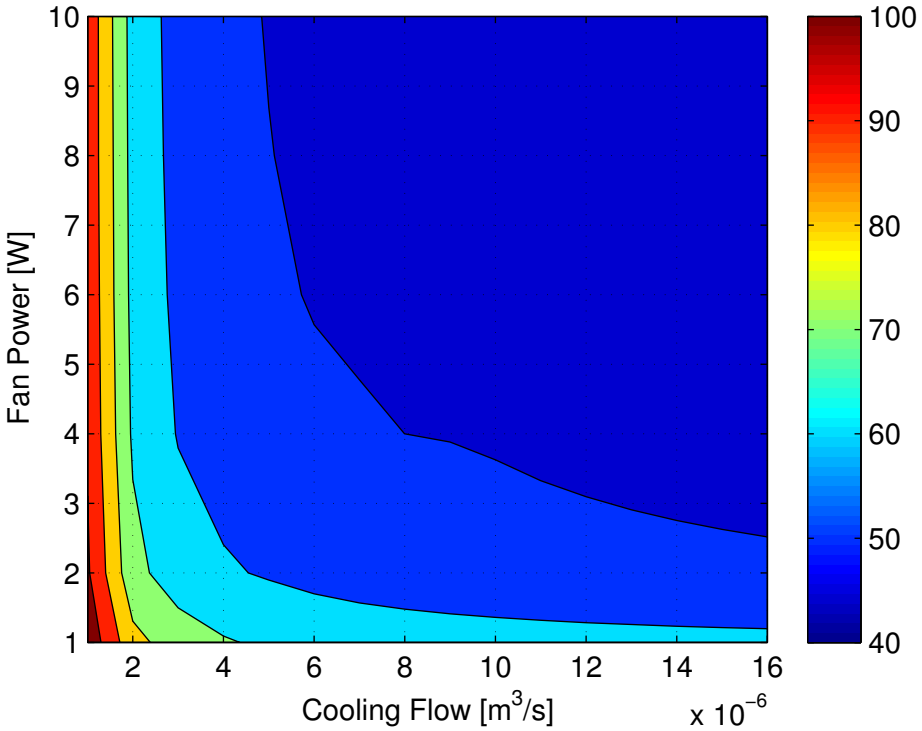


Figure 113: Rise Time[s] - 100W Heat Load

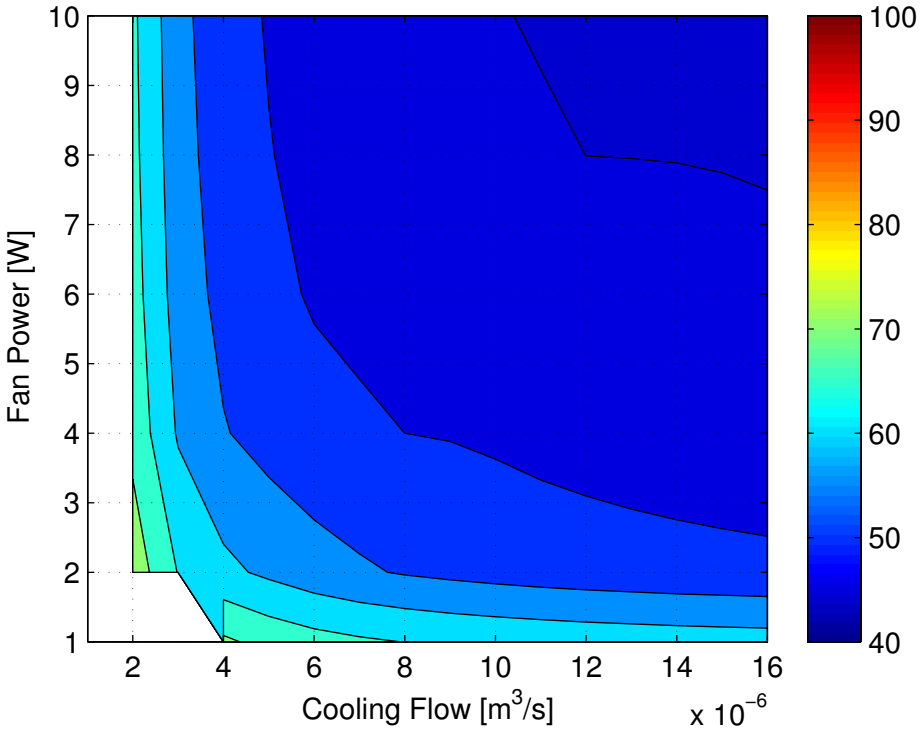


Figure 114: Rise Time[s] - 150W Heat Load

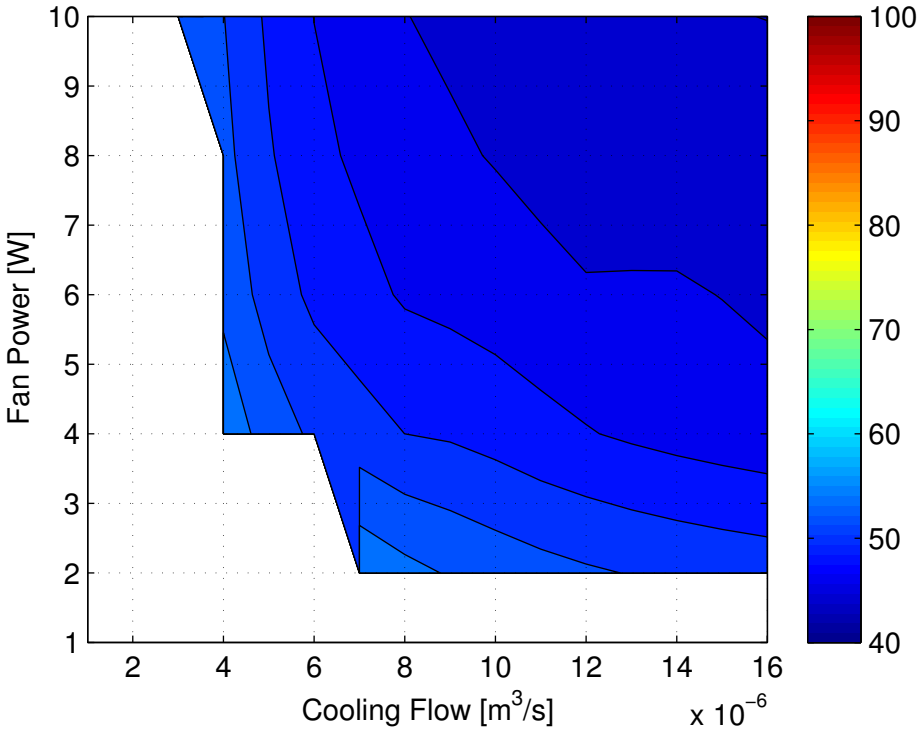


Figure 115: Rise Time[s] - 2000W Heat Load

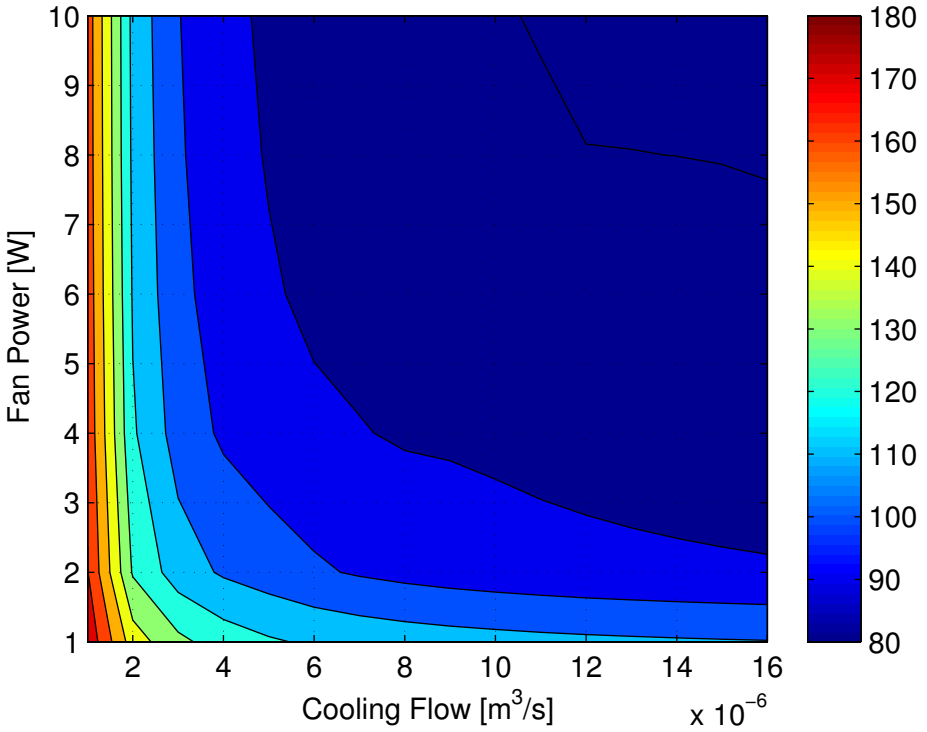


Figure 116: Settling Time[s] - 50W Heat Load

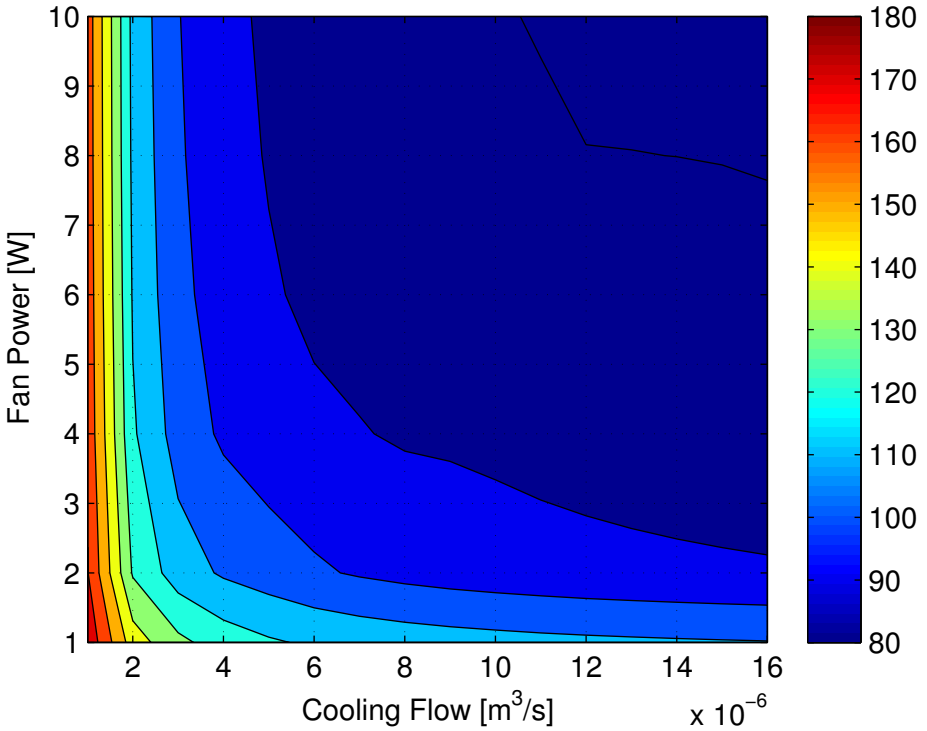


Figure 117: Settling Time[s] - 100W Heat Load

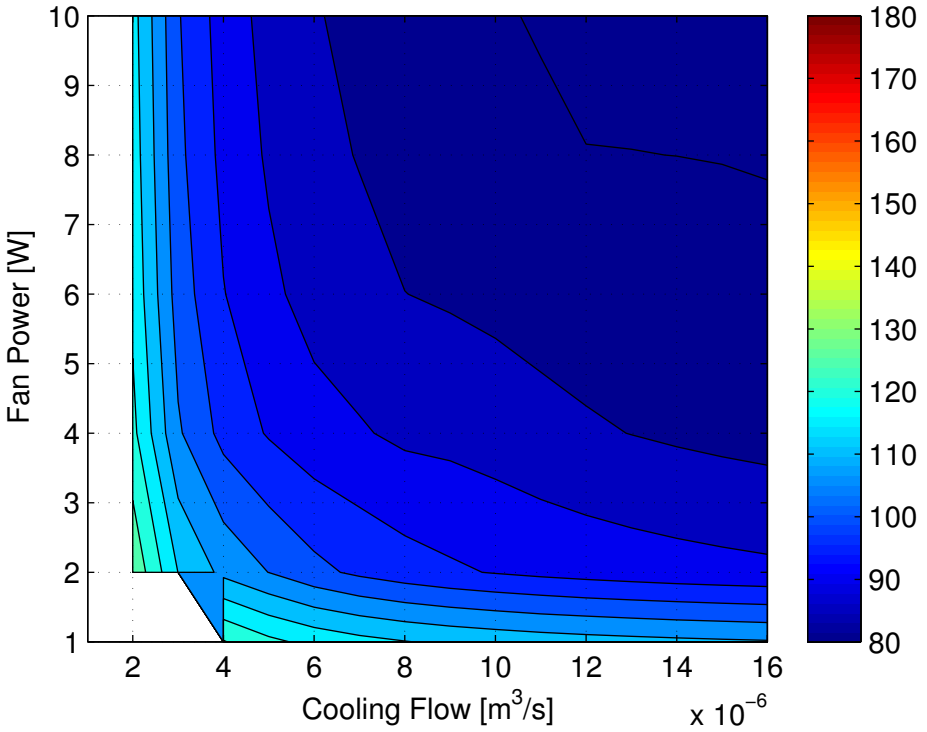


Figure 118: Settling Time[s] - 150W Heat Load

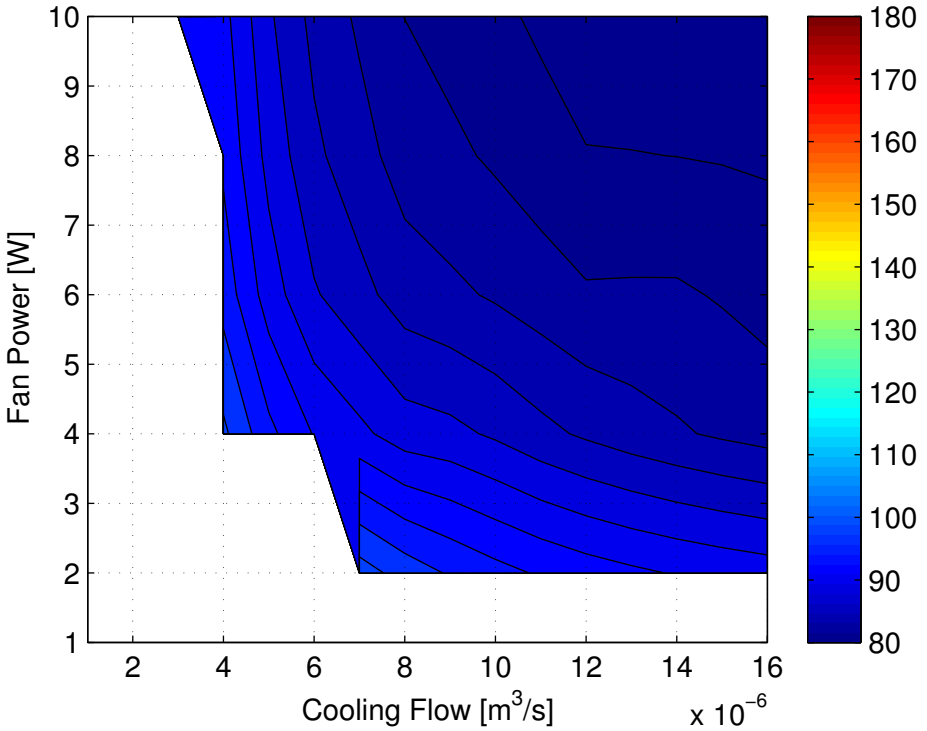


Figure 119: Settling Time[s] - 2000W Heat Load

cyclomatic complexity in the context of control configuration implies slight extrapolation. Nevertheless, it counts as an indicator to the control architecture's difficulty of implementation.

In addition to cyclomatic complexity, a qualitative description of the architectural structural complexity is given, and is linked to the physical system's reliability. Note that the extraction of the information presented in this section (on the control architecture and its interaction with the physical system) does not depend on the presence of a system's final design.

7.4.1.1 Fluid Network Isolation Valve

The valve consists of a processing module, an external interface module, an actuator that moves the valve stem, a sensor that measures the position of the valve stem, and a communication module. The valve meta-model is shown in Figure 120. The cyclomatic complexity is calculated as

$$M(V) = E - N + 2P = 5 - 5 + 2 \times 2 = 4$$

There are 12 isolation valves in the system. Each of the rest of the 18 valves is a part of another component.

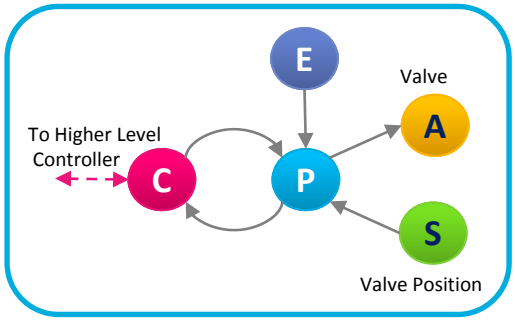


Figure 120: Fluid Network Isolation Valve Metamodel

7.4.1.2 Fluid Network Service

The service consists of a processing module, an external interface module, an actuator that moves the valve associated with the service, a sensor that measures the service flow rate, and a communication module. The service meta-model is shown in Figure 121. The cyclomatic complexity is computed as

$$M(S) = E - N + 2P = 5 - 5 + 2 \times 2 = 4$$

There are six services in the system, two of which are vital services.

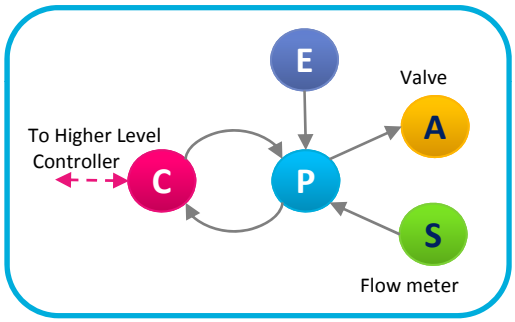


Figure 121: Fluid Network Service Metamodel

7.4.1.3 Fluid Network Plant

The plant consists of a processing module, an external interface module, an actuator that moves the two pump valves and turns on the pump, a sensor that measures the service flow rate, and a communication module. The plant meta-model is shown in Figure 122. The cyclomatic complexity is given by

$$M(P) = E - N + 2P = 5 - 5 + 2 \times 2 = 4$$

It is worth noting that the plant also contains a pressure sensor. Figure 122 shows the pressure sensor added to the node. However this does not change the cyclomatic complexity of the node since the number of edges increases by one as well as the number of modules. There are four plants in the system.

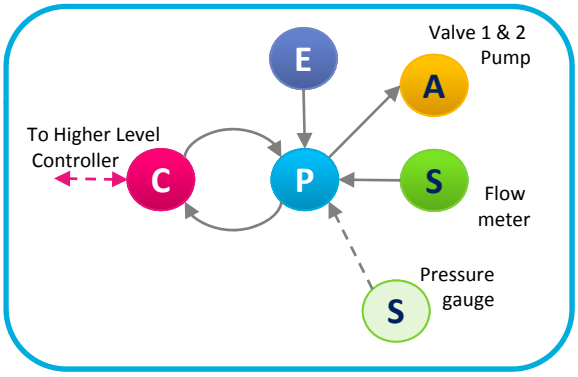


Figure 122: Fluid Network Plant Metamodel

7.4.1.4 Power Panel Assembly

The power panel assembly consists of a processing module, a knowledge extraction module, the communication module, and an external interface module. The power panels assemblies supply the zones with electric power through individual electric power panels. Hence each power panel assembly is assumed to have four actuators, corresponding to the four power panels, and for sensors corresponding to the power panel current sensors. The power panel assembly meta-model is shown in Figure 123. The cyclomatic complexity is computed as

$$M(A) = E - N + 2P = 13 - 12 + 2 \times 5 = 9$$

The electric system has two power panel assemblies.

7.4.1.5 Thermal System Heat Service

The thermal service consists of a processing module, an external interface module, an actuator that drives the air circulating fan, a temperature sensor that measures th temperature of the survice, and a communication module. The thermal service meta-model is shown in Figure 124. The cyclomatic complexity is calculated as

$$M(T) = E - N + 2P = 5 - 5 + 2 \times 2 = 4$$

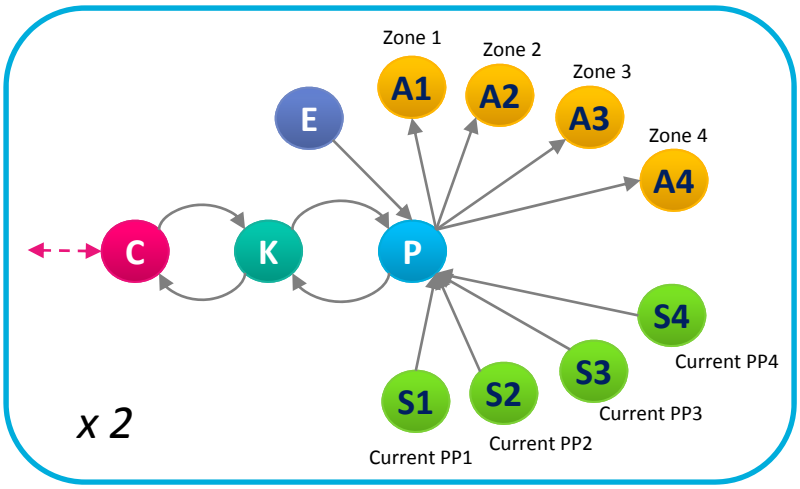


Figure 123: Electric Network Power Panel Assembly Metamodel

There are 6 thermal services in the system, each associated with one fluid network service.

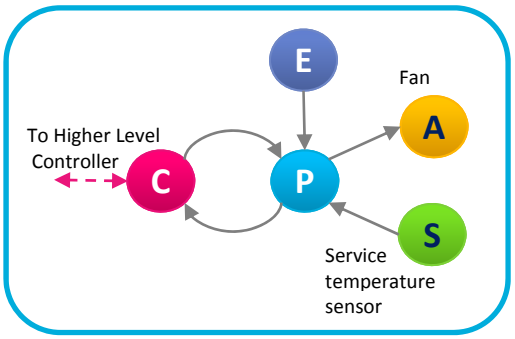


Figure 124: Thermal Network Service Metamodel

7.4.2 Control Architecture Configuration

7.4.2.1 Function Federation Architecture

In this type of architecture, the system is decomposed based on functionality. A subsystem is defined, in the context of the functional federation, a group of components that collectively achieve a certain function. The system has three major subsystems: The fluid network subsystem, the electric network subsystem, and the

thermal subsystem. the objective is to put them in a control hierarchy that minimizes complexity, allowing for better control architecture performance, testing, scalability, and reliability.

Although it is possible to calculate the complexity of the full structure of the control architecture, it does not provide much information. The control architecture is not designed all at once. Usually it is designed in phases and in modules. Hence, the complexity of individual layers within the hierarchy, i.e. the way control architectures are designed, is what is important and provides much information on the quality of a control architecture versus another.

Although the functional federation architecture will be shown for the three subsystems, more focus will be placed on the fluid network subsystem since it has more components and features. The total system architecture is shown in Figure 125.

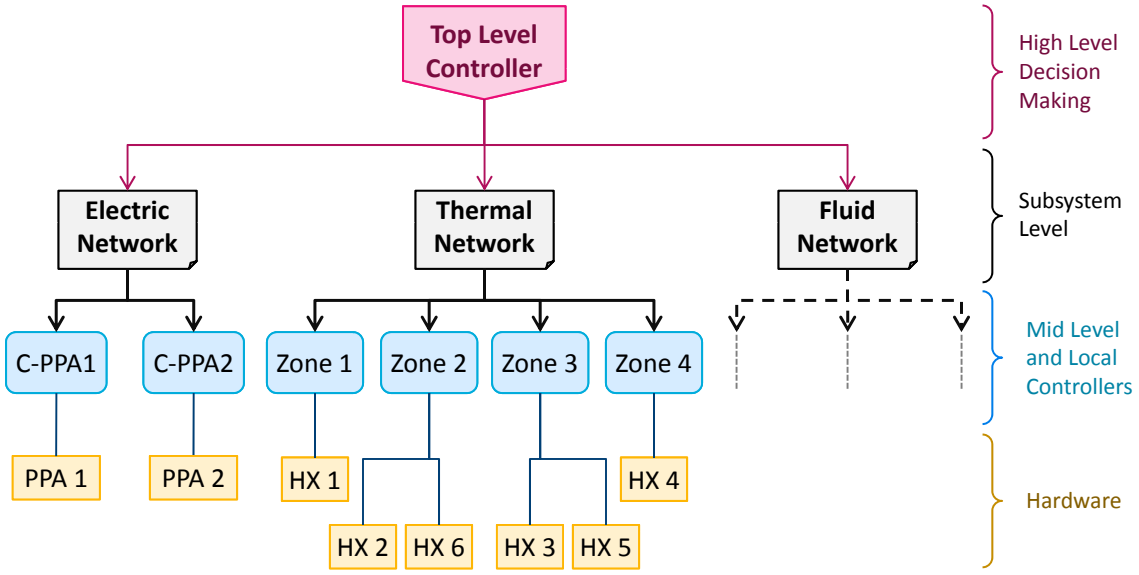


Figure 125: System Functional Federation Control Architecture

7.4.2.2 Electric Network Control Architecture

The electric power network has two power panel assembly controllers that link directly to the hardware. It is assumed that power generation and electric bus routing control

architecture are separate, and there function is to provide power for the power panels. The electric network subsystem controller, shown in Figure 125, ensures that power is routed from the buses to the destination hardware.

The electric power network control architecture is of the hierarchical type. The hierarchical and distributed architectures are very similar in behavior since the electric system has only two sub-controllers. Substituting the hierarchy with single centralized controller increases the complexity considerably.

The cyclomatic complexity of the power assembly controller was computed to be 9. This is somewhat a high complexity, since the rule of thumb is that software with complexity higher than 10 should be a partitioned into modules. Breaking down the power panel assembly controller (controlling four zones) into four separate sub controllers, each controlling one zone reduces overall complexity from nine to a complexity of four for each sub controller. However, this increases the cyclomatic complexity of the control hierarchy, since instead of one module, the hierarchy is now composed of four modules.

A decision on the part of the designer needs to be made whether lower complexity on the lower level accompanied by higher overall structural complexity is more preferred than the opposite. Usually, the higher overall complexity resulting from using more lower-level controllers can be reduced by further breakdown in two more layers of control. This is verified in large-scale control architectures common practices where control systems are almost always of a multilayered nature. Hence the preference is to you use simpler lower-level controllers with more layers in the hierarchy.

7.4.2.3 Thermal Network Control Architecture

The choice architecture for the thermal network control is a 2-layer control hierarchy. The first layer is the subsystem control layer, the second is the zonal layer. Each zone module controls the heat exchangers (only the thermal network components, since it

is a functional federation) within the zone.

7.4.2.4 Fluid Network - Functional Federation Control Architecture

The fluid network architecture is part of the overall hierarchical system architecture. Since it has more components and features than the aforementioned networks, it is shown on separate figures. Note that the valves that are not embedded in neither the plants nor the services are isolation valves, and are only used for reconfiguring the network. Hence these valves have the same function and can be grouped in one module. In addition, these valves work in pairs (one on the supply line and another on the return line), i.e. the pair of valves open and close simultaneously. Therefore, each pair is treated as a single control signal destination. This approach reduces the architecture complexity considerably.

One of the more common control architectures is what is referred to in Chapter 5 as the functional federation architecture. The control architecture presented in this section is a strict functional federation, in which all components are grouped according to their functions, starting from the top, the system level, followed by the subsystem level, and finally down to the hardware level. Applying this architecture to the fluid network results in the configuration shown in Figure 126.

The fluid network control architecture is further decomposed into three subsystems: 1) the plants controller, which includes all pump assemblies, their associated valves and sensors, and 2) the services controller, together with their flow control valves, and 3) the reconfiguration controller subsystem, which includes all the isolation valves.

Functionally, the architecture makes sense; separating the resources from the consumers, and both from the backup reconfiguration system. However from a practical stand point, it is not a preferred option. Calculating the cyclomatic complexity of

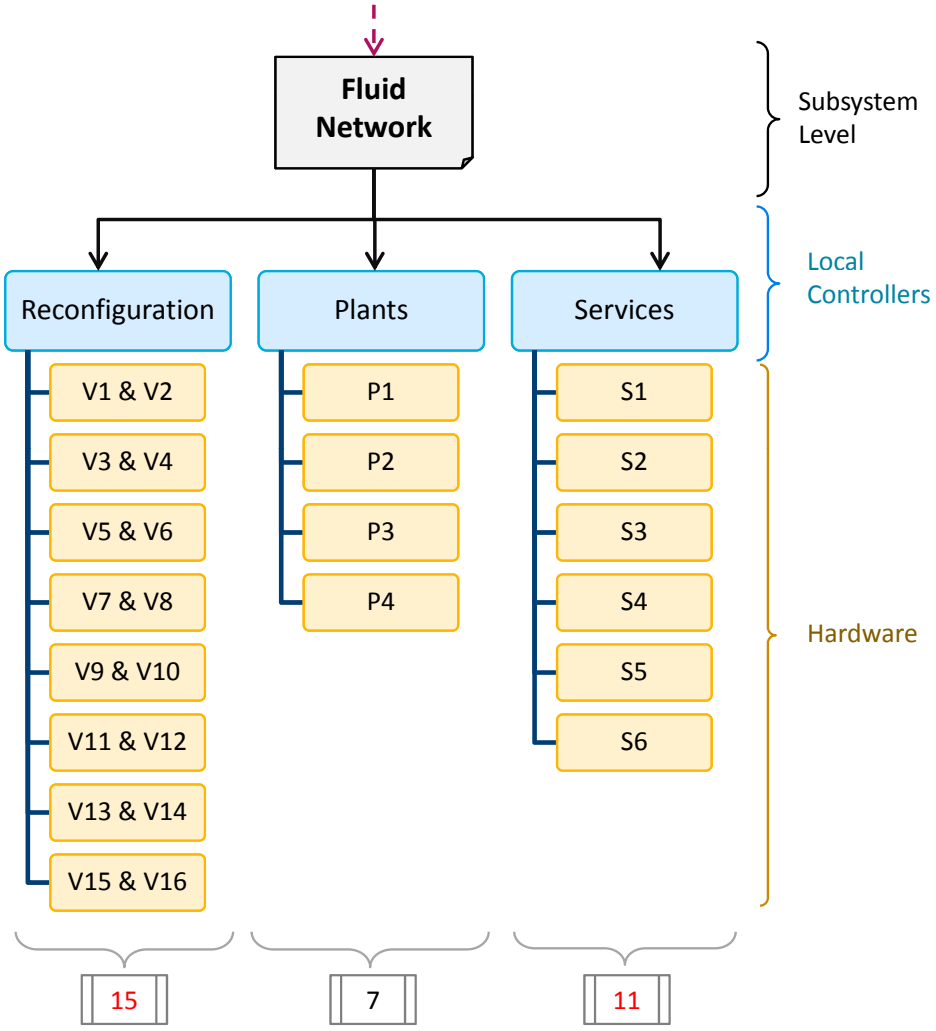


Figure 126: Strict Functional Federation Control Architecture

the three control architecture subsystems:

$$\text{Reconfiguration: } M = 8 - 9 + 2 \times 8 = 15$$

$$\text{Plants: } M = 4 - 5 + 2 \times 4 = 7$$

$$\text{Services: } M = 6 - 7 + 2 \times 6 = 11$$

In addition, the overall fluid network controller complexity is given by

$$\text{Fluid Network (Strict): } M = 3 - 4 + 2 \times 3 = 5$$

A cyclomatic complexity of 5 or 7 is acceptable. In fact, any complexity less that 10 is acceptable. The higher the complexity, the harder to test the system and the more

the system is prone to error. Even if the services complexity of 11 is accepted, the reconfiguration subsystem complexity of 15 is an unacceptable value. However, the fluid network higher level cyclomatic complexity is at an extremely graceful 5.

Another perspective on the controller complexity is related to the internal structure of the controller within the configuration. The reconfiguration controller has eight pairs of valves, each with (a minimum of) two states: "'open"' and "'close"'. The controller has to make eight decisions for the eight valve pairs, known as the decision set, depending on the external environment system drivers. In other words, the controller has to choose a decision set from among a group of the order of 2^8 different decision sets to achieve proper reconfiguration. (Note that a few decision sets are not realistic, such as all open, since some services are supplied by two lines, but this does not change the order of magnitude of the decision set choices). Such a controller is a challenging controller to design and implement.

The next step is to use the meta-model to explore different architecture options, such that the systems with higher complexity are partitioned or redistributed.

7.4.2.5 Fluid Network - Hybrid Federation Control Architecture

Following the system's functional decomposition into three major subsystems (thermal, electrical and fluid), the architecture is further decomposed based on a spatial decomposition. This leads to four subsystem controllers corresponding to the fluid network zones. Each zone controller is linked to the fluid network zone components: the plant, the service(s), and the two pairs of isolation valves. Note that only the fluid network components here, since the architecture started with a functional decomposition. The control architecture is shown in Figure 127.

Increasing the number of the fluid network subsystems to 4 results in an increase in the complexity of the network. This is calculated as

$$\text{Fluid Network (Hybrid): } M = 4 - 5 + 2 \times 4 = 7$$

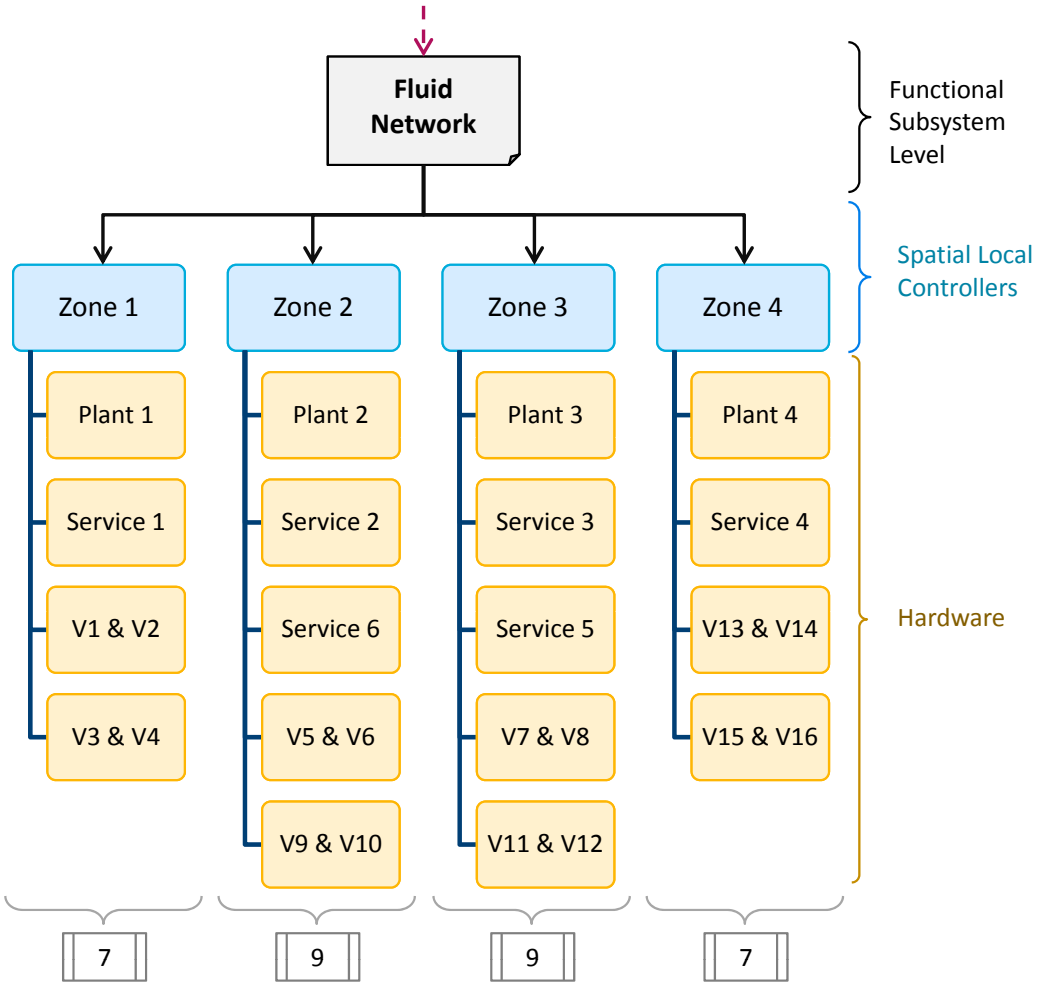


Figure 127: Hybrid Functional - Spatial Federation Control Architecture

which is not a drastic increase. As a matter of fact, a cyclomatic complexity of about 7 is more on the normal side for a software rich with features. However, the main advantage of this architecture is evident when calculating the cyclomatic complexities of the zone controllers

$$\text{Zone 1: } M = 4 - 5 + 2 \times 4 = 7$$

$$\text{Zone 2: } M = 5 - 6 + 2 \times 5 = 9$$

$$\text{Zone 3: } M = 5 - 6 + 2 \times 5 = 9$$

$$\text{Zone 4: } M = 4 - 5 + 2 \times 4 = 7$$

The slight increase of the fluid network top level control layer complexity of 7 was accompanied by a drastic decrease of the zonal control layers complexity to 7s and 9s. Designing a 9 (or a definitely 7) cyclomatic complexity piece of software to act as the zonal controller is a very feasible task, together with testing, validation, among other software design tasks. Unless there is another design driver that requires the strict functional federation, the hybrid federation is a clear winner here.

The structural complexity of such a configuration can also be viewed from a controller design requirement. In case of Zones 2 and 3, assuming the plant, two services, and two configuration valve pairs have two levels of decisions each. Hence the controller has to choose a decision set from among 2^5 decision sets (2^4 in case of Zones 1 and 4), definitely lower than the 2^8 from the previous architecture option. Designing a controller for the hybrid configuration is less complex than one for a functional federation configuration.

The failure of one of the zone plants results in the requirement that an adjacent zone supplies the services in both zones. For example, if Plant 1 fails, Plant 2 in Zone 2 supplies both Services 1 and 2. This increases the structural complexity of the controller, since it has to choose a decision from among 2^7 decision choices (3 services, 4 pairs of valves). If Plant 4 in Zone 4 also fails, the decision pool increases to an order of magnitude to 2^10, an unacceptable case. Hence, for the hybrid configuration to be as feasible as it is attractive, a systems design requirement has to be placed on plant reliability. Three plants cannot fail simultaneously, or else the control architecture will fail. This can be achieved with proper selection of highly reliable pumps.

7.4.2.6 Fluid Network - A Variant Hybrid Federation Control Architecture

The previous hybrid control architecture is a viable and attractive design option. However, the strict functional architecture has the advantage of logically decomposing the system into its main functions. Observing the components on a zonal level,

it is clear that each zone possesses the same functions as the fluid network level, namely: the resource, the consumer, and the reconfiguration system. Therefore if the components within the zone are grouped based on their function, and a local controller is designed for each group, then the best of both worlds is achieved: logical / functional decomposition and reduced complexity.

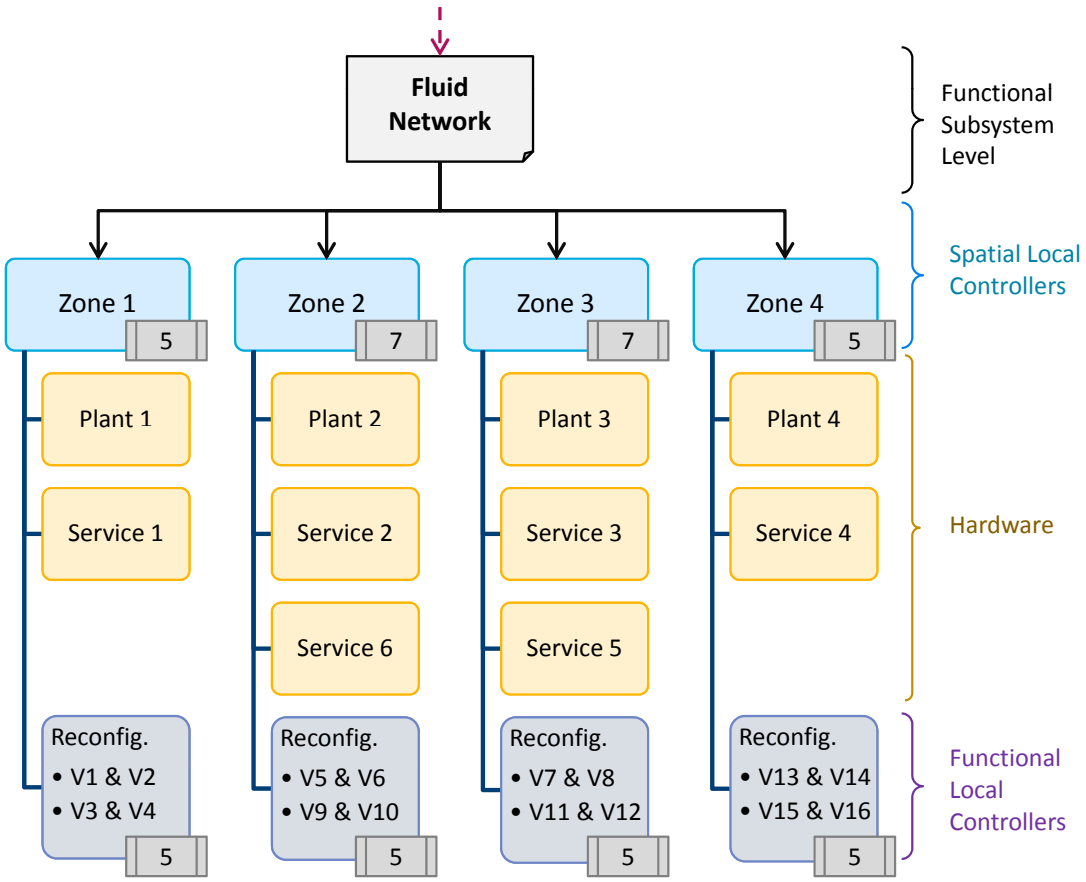


Figure 128: Variant Hybrid Functional - Spatial Federation Control Architecture

The variant hybrid architecture is shown in Figure 128. Each zone contains one plant and one service, each having its own controller as discussed in the previous section. On the other hand, the two valve pairs can be grouped into one local reconfiguration subsystem with its own controller. The fluid network top level controller

complexity is calculated as

$$\text{Fluid Network (Variant): } M = 4 - 5 + 2 \times 4 = 7$$

exactly the same as with the hybrid architecture, since no subsystems were added to the top level. The zonal complexity is given by

$$\text{Zone 1: } M = 3 - 4 + 2 \times 3 = 5$$

$$\text{Zone 2: } M = 4 - 5 + 2 \times 4 = 7$$

$$\text{Zone 3: } M = 4 - 5 + 2 \times 4 = 7$$

$$\text{Zone 4: } M = 3 - 4 + 2 \times 3 = 5$$

which is another reduction in the complexity as compared to the hybrid architecture; an even better solution. Moreover, the cyclomatic complexity of the newly added reconfiguration local subsystem is given by

$$\text{Fluid Network (Variant): } M = 5 - 5 + 2 \times 2 = 4$$

A similar case related to the number of decisions choices a control architecture can to choose from can be made here. Zones 1 and 4 have to choose from among 2^2 decisions, while Zones 2 and 3 have to choose from among 2^3 decisions. The reconfiguration controller has to choose from 2^2 decision choices. A drastic reduction in the controller algorithm design and implementation complexity is achieved by the variant hybrid architecture.

However the variant hybrid architecture has a catch. The four zone's reconfiguration controllers have to operate in synchronization. Information has to be exchanged between these controllers and taken into consideration when a decision set is chosen. Network communication requirements increase. This is specially true in the case of failure of one of the plants, since the two reconfiguration controllers (from the supply and starved zones) have to coordinate their actions.

The case of failure of one of the plants in the current architecture is similar to the case with the previous architecture. More than 2 pumps failing results in an over complex controller, that quickly loses its advantage over the fully functional federation architecture.

The fact is that the more layers added, and the more large subsystems are partitioned, the less cyclomatic complexity, and the less structural complexity. In addition, it is easier and more reliable to scale up the system by adding more components (with their controllers), or upgrade the system by swapping subsystems (with their less complicated, easier to design controllers). But this comes at a price. More components usually means less reliability, higher cost and more communication band width. This decision is highly dependant on the particular system and discipline. Therefore, it is beyond the scope of this dissertation. The decision is left for the designer depending on the design problem at hand.

CHAPTER VIII

CONCLUDING REMARKS

Large-scale distributed intelligent systems are starting to gain dominance in many applications. Systems such as advanced vehicles, (aircraft, naval vessels, spacecraft, etc.), networked systems (electric grid, fluid networks, wireless networks, etc.), hybrid systems (composed of dissimilar entities with a common goal) are some examples among a multitude of system of systems. Other applications include spatially distributed systems such as the Smart Grid, or functionally distributed systems such as network-centric taskforces. Large-scale intelligent systems are characterized by nonlinear behavior patterns that can only be predicted through simulation based engineering. In addition, the autonomy, intelligence, and reconfiguration capabilities required by certain systems introduce obstacles, therefore adding another layer of complexity. There is no standard way of representing control architectures. Consequently, many control architectures cannot be evaluated. Variations in modeling approaches, simulation scenarios, metrics definition, among other factors present a challenge in the fair comparison of control architectures.

8.1 Research Objectives Review

In spite of their importance, large-scale intelligent systems are not designed using standard processes. They do not have a well established design methodology that addresses the system's behavior design requirements in general, and the intelligence requirements in particular. Large-scale intelligent systems are regarded as regular complex systems during design, with the intelligence treated as an afterthought.

As a result, the current state of the art relies on the serial process of designing the physical system followed by the design of the intelligent control architecture. The

physical system design is usually finalized (or is in the detailed design phase) before the intelligent control architecture design is addressed. This process limits the control architecture to a handful of choices that are based on the physical system structure. Furthermore, the capability of the control architecture is limited by the physical system configuration choice. The physical system might not be able to achieve the required performance level no matter how well the control architecture is designed.

The goal of this research is to develop the necessary tools for a design methodology specific to large-scale intelligent systems. The design methodology has to be applicable during the conceptual design phase, during which the physical system's parameters and structure are not yet finalized. During the conceptual design phase, minimum financial investment has been made as far as the physical system is concerned. It is usually represented on paper or by simulation models, while costly hardware prototypes are seldom built.

In addition, the design methodology has to provide more information to the system designer during the conceptual design phase, information that may alter the final concept selection if only the physical system is considered. Typically, the design methodology considers the system's behavior characteristics as well as the physical system's constraints, in addition to the required functionality and limitations on computation. The product of applying the design methodology is a matching physical system and control architecture, coupled with feasibility, capability and evaluation analyses.

To accomplish this goal, the dissertation aims at achieving two main objectives. The first objective is the development of means (or tools) by which the interaction between the physical system and the control architecture can be studied, assessed, and analyzed. The second objective is to develop a common framework of control architecture representation, hence enabling the fair and consistent comparison of different architectures for the same physical system.

8.1.1 Objective I: Interaction Between the Physical System and the Control Architecture

The interaction between the physical system and the control architecture can be further decomposed into three sub-objectives. The first sub-objective is the interaction within the design process tasks. The effect of the physical system on the control architecture, as well as the best achievable performance by a control architecture on a given physical system comprise the second and third sub-objectives respectively.

In order to achieve the first sub-objective, the research presented in this dissertation proposes a concurrent approach to design large-scale intelligent systems. The physical system and the control architecture design processes are merged into a single hybrid design process with two parallel paths. Essential information pertaining to both the control architecture of the physical system is exchanged at specific milestones in the hybrid design process.

Yet, control architectures have no standard design process in the literature. The main tasks involved in designing control architectures are analyzed. In addition, drawing similarities between control architecture design and complex system design led to the proposition of a (semi-)standard control architecture design process. The proposed control architecture design process as well as the standard complex system conceptual design process forms the backbone of the hybrid intelligent systems design process.

Three design loops are identified in the hybrid design process. The first loop is the physical system conceptual designed loop. The second loop is the control architecture design loop in which a matching controller structure and algorithm is found. The third loop defines the interaction between the control architecture design and the physical system design processes. It is the focus of this dissertation.

The second and third sub-objectives are closely related to the third design loop.

The choice of a physical system configuration in the conceptual design phase (disregarding the control architecture) confines the control architecture design space to a few choices that match the physical system. On the other hand, if a physical system is not well designed there is not much the control architecture can do. Hence, no matter how good the control architecture is, the overall system's performance is limited by the choice of the physical system.

To assess the effect of the physical system on the control architecture and vice versa, a suite of analyses is developed. This suite has four main analyses categories. The first is the state reachability analysis, which checks whether the system is able to achieve the final required state or not. The second analysis category is the inverse dynamics analysis which relies on a behavioral model. Its purpose is to assess the required generalized forces that drive the system from its initial state to its required final state. The third type of analysis is the stability analysis which relates to the identification of instability regions within the system's domain of operation. Stability analysis can take a qualitative approach, if not enough information is available in the conceptual design phase. The fourth and last category of analysis is the capability potential of the system. It answers the question of how well the system can perform during operation.

The interaction between the physical system and the control architecture is explained in Chapter 4. A demonstration problem involving the design of a two degree of freedom mechanism is presented in Chapter 6. The analysis suite is applied to the design of this mechanism. Results show that if the control architecture consideration is not included early on in the conceptual design phase, there is a chance that the designer might choose a physical system configuration that highly increases the complexity of the controller. It also shows that the choice of specific control architectures limits the performance of the system. Hence, the first objective of this research is achieved.

8.1.2 Objective II: Control Architecture Meta-model

Intelligent control architectures design involves three major decisions: control configuration, control methods, and controller parameter selection. It is shown that the first two decisions can be addressed in the conceptual design phase. To accomplish this, it is necessary to analyze control architectures in categories. Presented analyses demonstrate that the traditional classification of controllers depends on the computational technique the designer uses during controller design. It is shown that this specification is not suitable for control architectures conceptual design. Hence, control architectures are reclassified into a novel taxonomy based on the control architecture configurations and methods.

A control architecture's meta-model is developed using the novel taxonomy, emphasizing the distinction between control configurations and control methods. As far as control configurations are concerned, two controlled architecture structures are proposed: a spatial federation and a functional federation. Any control architecture structure can be represented by one or the other or by a hybrid between both.

Controller methods are represented in the form of a generalized control node. The modules contained within the control node act as generic placeholders for software modules in the controllers. The choice of modules within a control node, and the algorithms used within the control node defines the control architecture method. The collection of control nodes in a specific structure defines the control architecture configuration. The advantage of such approach is evident when control methods are mixed within the same control architecture.

Henceforth, the standard representation of a wide spectrum of control architecture frameworks is possible. Since the control architecture meta-model draws similarities with software architectures, standard software metrics can be applied to different control architecture meta-models. Such an approach enables consistent and fair comparison of control architectures applied to the same physical system.

The mechanism design problem is addressed to demonstrate the utilization of the control architecture meta-model. Four different control architectures spanning the control architecture design space are successfully represented using the meta-model. Furthermore, the evaluation of each of the control architectures using software metrics is shown to be possible.

8.2 Methodology Limitations

The intelligent systems design methodology proposed in this dissertation has two main elements, namely the hybrid design process with the accompanied analyses suite, and the control architecture meta-model for both control configurations and methods. Information is lost if a subset of the above analyses are not performed on the system. Most of the above analyses require a behavioral simulation model of the system, accompanied by appropriate scenarios.

Consequently, if the physical system under consideration cannot be represented by a behavioral model, the above methodology cannot be applied. A behavioral model requires that the characteristics of the system are known, or that the system drivers are understood. The constraints on their system are part of its behavior, whether these constraints come from the physical limitations or the functional limitations.

Some systems rely on their transient behavior to maintain their functionality. Behavioral models, by definition, cannot predict transient characteristics. Hence, systems such as power generation systems cannot be addressed by this methodology. Also, the behavioral model requires continuous dynamics in contrast to discrete event systems. This rules out systems that rely on connectivity more than energy or mass transfer, such as the Internet or wireless networks.

8.3 Summary of Contributions

8.3.1 Intelligent Systems Hybrid Design Process

- Standard controller design process. The literature contains multiple controller design approaches with different tasks in each. The standard controller design process integrates all tasks into a general standard design process.

- Hybrid and concurrent conceptual design methodology for intelligent large-scale systems. The complex system design process and the control architecture design process are merged into hybrid design process with two parallel paths.

- Main control architecture design tasks breakdown. Three main tasks (or decisions) are required for control architecture design: control configuration, method, and parameters selection. It is determined that the first two tasks are doable in the conceptual design phase.

- Identification of information pertaining to the control architecture and potentially available for the designer during the conceptual design phase, including capability potential and constraints.

  - Definition of a suite of four analyses that characterize the control architecture behavior. The analyses are reachability analysis, inverse dynamics analysis, stability analysis, and capability potential analysis.

  - Effect of physical system on control architecture design with respect to configuration and methods.

  - Limitations the control architecture places on system performance.

8.3.2 Control Architecture Meta-model

- Control architecture configuration meta-model describes the structure of the control architecture, its components, their functionality, and information exchange. Almost all control architecture configurations representation are possible using this meta-model.
  - Spatial federation representation such that the control architecture is structured using a station decomposition.
  - Functional federation representation in which the control architecture is structured based on the functionality of its components.
  - Hybrid federation representation which has features from both the spatial and the functional federations.
- Controller novel taxonomy. The traditional controller classification was substituted for a proposed innovative control architecture taxonomy on two levels: control configurations and controller methods.
- Control methods meta-model describes the control algorithms used within every part of the control architecture.
  - Generic control node meta-model is the elementary building block of the control methods meta-model. It is a collection of connected genetic software module's with predefined functionality of each.
  - Ability to represent any control method by manipulating the software modules within the node.
- Effective analogy between control architectures and software architecture. Since most control architectures are built of software modules, software design metrics can be applied to control architectures.

8.4 Recommendations for Future Research

The dissertation presents a set of tools and analyses packaged into a design methodology for intelligent systems. This is a seldom addressed topic in the literature and has only been gaining momentum recently. Most of the research is still in its infancy phases, and this dissertation is not an exception. There are many open research problems that need to be addressed using methods from control engineering and systems engineering.

One such problem is the automation of the concurrent hybrid design process of intelligent systems. The iterations on the internal design that links the physical architecture and the control architecture and design processes is currently manual. Iterations, changing system parameters and configurations, testing new methods and algorithms, are some tasks within this design loop. Unfortunately, they are not automated. The design space cannot be systematically spanned. Part of the future research should be directed towards automation of the design process.

In addition, little effort has been focused on the differences between the three design loops. The characteristics of each design loop and its influence on the system, whether the physical system or the control architecture, may shed a light on how to automate the design process.

Only specific control methods have been implemented using the controller methods meta-model. Control architectures involving control methods such as model predictive control, neural network control, fuzzy logic control, and so on, need to be represented by the control architecture meta-model. There is a chance that the control nodes structure might change to accommodate different control methods.

The expansion of control architecture meta-model to accommodate more software design metrics is another improvement. Metrics that specifically measure the structural complexity of the control architecture configuration is an important advancement to the control architecture meta-model. Perhaps an analogy can be made

with other engineering fields similar to the one made with software engineering.

Control architecture reconfiguration is another potential research path. Traditionally, reconfigurability assessment has been limited. Although not directly applicable to control architectures, the research presented in [38] and [54] describe a way of assessing the "'reconfiguration ease"' of a particular architecture using existing system's modularity measures. Modularity metrics are based on a design structure matrix, which can be adopted for control architectures as well.

It will be interesting to further explore the potential synergy between the control architecture meta-model and AADL. The control architecture meta-model is specific to controls, yet draws similarities with software architectures, while AADL is an established meta-modeling language focusing on hardware components and the communication among them. This presents a compelling case that AADL and the control architecture meta-model compliment each other.

APPENDIX A

MECHANISM DESIGN PROBLEM COMPUTER CODE

A.1 Four Link Geometry

```
function [D,th1_d_min, th1_d_max, th2_d_min, th2_d_max] = ...
    FiveLinkGeometry(R1, R2, R3, R4)

D = R3 * cos(30*pi/180) + R4 * cos(30*pi/180);
b = D / 2;
a = sqrt(R1^2-b^2);

th1_d_min = 180/pi * atan(a/b);
th2_d_max = 180 - th1_d_min;

d = sqrt( (R3 * cos(30*pi/180))^2 + R1^2 );
r3 = acos( (R1^2+d^2-R3^2) / (2*R1*d) );
eta = atan(R1/b);

th1_d_max = 180/pi * (eta + r3);
th2_d_min = 180 - th1_d_max;

th1_d_min = ceil(th1_d_min);
th1_d_max = floor(th1_d_max);
th2_d_min = ceil(th2_d_min);
th2_d_max = floor(th2_d_max);
```

```
end
```

A.2 Four Link Kinematics

```
function [phi3_d, phi4_d, th3_d, th4_d, x, y] =
    FiveLinkAngles(th1_d,th2_d)

%D = 7.4641; R1 = 4; R2 = 4; R3 = 2; R4 = 2;
% D = 3.4641; R1 = 4; R2 = 4; R3 = 2; R4 = 2;
R1 = 4; R2 = 4; R3 = 2; R4 = 2;
[D,th1_d_min, th1_d_max, th2_d_min, th2_d_max]
   =FiveLinkGeometry(R1, R2, R3, R4);

th1 = th1_d*pi/180;
th2 = th2_d*pi/180;

M1=R2; M3=D; M4=R1; M5=R1; M6=D^2+R2^2+R1^2;
M10=2*D*R2; M11=2*D*R1; M12=2*R1*R2;
M19=0.5* (D^2+R2^2+R4^2-R3^2+R1^2);
M20=D*R2; M21=D*R1; M22=R1*R2;

K1 = M1 + M3*cos(th2) - M4*cos(th2-th1);
K2 = sqrt( M6 + M10*cos(th2) - M11*cos(th1) -
    M12*cos(th2-th1) );
K3 = M19 + M20*cos(th2) - M21*cos(th1) - M22*cos(th2-th1);
K4 = R4;

phi3 = acos(K1/K2) + acos(K3/K2/K4) ;
```

```
N1=R1; N3=D; N4=R2; N5=R2;
N6=0.5* (D^2+R2^2-R4^2+R3^2+R1^2);
N7=D*R2; N8=D*R1; N9=R1*R2; N10=R1*R2;

K5 = N1 - N3*cos(th1) - N4*cos(th2-th1);
K6 = N6 + N7*cos(th2) - N8*cos(th1) - N9*cos(th2-th1);
K7 = R3;

phi4 = acos(K5/K2) + acos(K6/K2/K7);

phi3_d = phi3*180/pi;
phi4_d = phi4*180/pi;

th3_d = phi4_d+th1_d-180;
th4_d = th2_d+180-phi3_d;

th3 = th3_d*pi/180;

x = R1*cos(th1) + R3*cos(th3);
y = R1*sin(th1) + R3*sin(th3);

if ~(isreal(x) || isreal(y) )
    x = NaN;
    y = NaN;
end
if (phi4_d>180 || phi3_d>180)
```

```
    x = NaN;
    y = NaN;
end

end
```

A.3 Four Link Inverse Kinematics

```
function [th1_d, th2_d] = FiveLinkAngles_Inv(x,y)

% D = 7.4641; R1 = 4; R2 = 4; R3 = 2; R4 = 2;
% D = 3.4641; R1 = 4; R2 = 4; R3 = 2; R4 = 2;
R1 = 4; R2 = 4; R3 = 2; R4 = 2;
[D,th1_d_min, th1_d_max, th2_d_min, th2_d_max]
   =FiveLinkGeometry(R1, R2, R3, R4);

R13 = sqrt(x^2+y^2);
r3  = acos( (R1^2+R13^2-R3^2) / (2*R1*R13) );
rA  = atan(y / x);

th1_d = 180/pi * (r3+rA);

R24 = sqrt((D-x)^2+y^2);
r4  = acos((R2^2+R24^2-R4^2) / (2*R2*R24) );
rB  = atan( y / (D-x) );

th2_d =  180 - 180/pi*(rB+r4);
```

```
if ~isreal(th1_d) || ~isreal(th2_d)
    th1_d = NaN;
    th2_d = NaN;
end

if th1_d>93 || th1_d<65
    th1_d = NaN;
    th2_d = NaN;
end

if th2_d<87 || th2_d>115
    th1_d = NaN;
    th2_d = NaN;
end

end
```

A.4 Two Link Error Analysis

```
th1_d = [(60:1:90)];
th3_d = [(-10:1:45)];

D = 7.4641; R1 = 4; R3 = 2;

X = zeros([length(th3_d),length(th1_d) ]);
Y = zeros([length(th3_d),length(th1_d) ]);
E_X = zeros([length(th3_d),length(th1_d) ]);
E_Y = zeros([length(th3_d),length(th1_d) ]);
```

```
E    = zeros([length(th3_d),length(th1_d) ]);
E_R_max  = zeros([length(th3_d),length(th1_d) ]);
E_R_mean = zeros([length(th3_d),length(th1_d) ]);

for i=1:length(th1_d)
    for j=1:length(th3_d)

        [x_nom, y_nom] = TwoBarKinemtics(th1_d(i),th3_d(j),
          R1, R3);
        [x_11, y_11]   =
          TwoBarKinemtics(th1_d(i)+0.01,th3_d(j), R1, R3);
        [x_12, y_12]   =
          TwoBarKinemtics(th1_d(i)-0.01,th3_d(j), R1, R3);
        [x_21, y_21]   =
          TwoBarKinemtics(th1_d(i),th3_d(j)+0.01, R1, R3);
        [x_22, y_22]   =
          TwoBarKinemtics(th1_d(i),th3_d(j)-0.01, R1, R3);
        [x_31, y_31]   =
          TwoBarKinemtics(th1_d(i)+0.01,th3_d(j)+0.01, R1,
          R3);
        [x_32, y_32]   =
          TwoBarKinemtics(th1_d(i)-0.01,th3_d(j)-0.01, R1,
          R3);
        [x_41, y_41]   =
          TwoBarKinemtics(th1_d(i)+0.01,th3_d(j)-0.01, R1,
          R3);
```

```
[x_42, y_42]    =
    TwoBarKinemtics(th1_d(i)-0.01,th3_d(j)+0.01, R1,
    R3);


E_x = sqrt(mean([(x_nom-x_11)^2, (x_nom-x_12)^2,
    (x_nom-x_21)^2, (x_nom-x_22)^2, ...
                    (x_nom-x_31)^2, (x_nom-x_32)^2,
                        (x_nom-x_41)^2, (x_nom-x_42)^2]
                        ) );
E_y = sqrt(mean([(y_nom-y_11)^2, (y_nom-y_12)^2,
    (y_nom-y_21)^2, (y_nom-y_22)^2, ...
                    (y_nom-y_31)^2, (y_nom-y_32)^2,
                        (y_nom-y_41)^2, (y_nom-y_42)^2]
                        ) );




E_x_2 = [(x_nom-x_11)^2, (x_nom-x_12)^2,
    (x_nom-x_21)^2, (x_nom-x_22)^2, ...
                    (x_nom-x_31)^2, (x_nom-x_32)^2,
                        (x_nom-x_41)^2, (x_nom-x_42)^2] ;
E_y_2 = [(y_nom-y_11)^2, (y_nom-y_12)^2,
    (y_nom-y_21)^2, (y_nom-y_22)^2, ...
                    (y_nom-y_31)^2, (y_nom-y_32)^2,
                        (y_nom-y_41)^2, (y_nom-y_42)^2] ;
E_r    = sqrt(E_x_2 + E_y_2) ;
```

```
            E_R_max(j,i) = max(E_r);
            E_R_mean(j,i) = mean(E_r);

            X(j,i)   = x_nom;    Y(j,i)   = y_nom;
            E_X(j,i) = E_x;  E_Y(j,i) = E_y;
            E(j,i)   = sqrt(E_x^2+E_y^2);

        end
    end



line([0, R1*cos(pi/180*60),
    R1*cos(pi/180*60)+R3*cos(pi/180*30)],...
        [0, R1*sin(pi/180*60),
            R1*sin(pi/180*60)+R3*sin(pi/180*30)],...
            'Color','b','LineWidth',0.5);


rectangle('Position',[3.732-1.7, 4.464-0.5, 1.25, 1]);


axis equal
```

A.5 Two Link Inverse Error Analysis

```
% th1_d = [(60:1:90)];
```

```
% th3_d = [(-10:1:45)];
clear E* X* Y* x* y* T*
figure(1)
%clf
hold on

D = 7.4641; R1 = 4; R3 = 2;
rectangle('Position',[D/2-1.7-4, 4.464-0.5+0.736, 1.25, 1]);

x = [D/2-1.7    : 0.01 : D/2-1.7+1.25 ];
y = [4.464-0.5 : 0.01: 4.464+0.5];

% x = [7.4/2-1.7 : 0.005 : 7.4/2-1.7+1.25 ];
% y = [4.4-0.5    : 0.02 : 4.4+0.5];

X = zeros([length(y),length(x) ]);
Y = zeros([length(y),length(x) ]);
E_X = zeros([length(y),length(x) ]);
E_Y = zeros([length(y),length(x) ]);
E   = zeros([length(y),length(x) ]);
E_R_max  = zeros([length(y),length(x) ]);
E_R_mean = zeros([length(y),length(x) ]);
E_X_max  = zeros([length(y),length(x) ]);
E_Y_max  = zeros([length(y),length(x) ]);
E_X_mean = zeros([length(y),length(x) ]);
E_Y_mean = zeros([length(y),length(x) ]);
```

```
TH1_d = zeros([length(y),length(x) ]);
TH3_d = zeros([length(y),length(x) ]);

for i=1:length(x)
    for j=1:length(y)
        [th1_d, th3_d]=TwoBarKinemtics_Inv(x(i), y(j), R1,
            R3);
        X(j,i) = x(i); Y(j,i) = y(j);
        TH1_d(j,i) = th1_d; TH3_d(j,i) = th3_d;
    end
end




for i=1:length(x)
    for j=1:length(y)
        [x_nom, y_nom] =
            TwoBarKinemtics(TH1_d(j,i),TH3_d(j,i), R1, R3);
        [x_11, y_11]   =
            TwoBarKinemtics(TH1_d(j,i)+0.01,TH3_d(j,i), R1,
            R3);
        [x_12, y_12]   =
            TwoBarKinemtics(TH1_d(j,i)-0.01,TH3_d(j,i), R1,
            R3);
```

```
        [x_21, y_21]    =
            TwoBarKinemtics(TH1_d(j,i),TH3_d(j,i)+0.01, R1,
            R3);
        [x_22, y_22]    =
            TwoBarKinemtics(TH1_d(j,i),TH3_d(j,i)-0.01, R1,
            R3);
        [x_31, y_31]    =
            TwoBarKinemtics(TH1_d(j,i)+0.01,TH3_d(j,i)+0.01,
            R1, R3);
        [x_32, y_32]    =
            TwoBarKinemtics(TH1_d(j,i)-0.01,TH3_d(j,i)-0.01,
            R1, R3);
        [x_41, y_41]    =
            TwoBarKinemtics(TH1_d(j,i)+0.01,TH3_d(j,i)-0.01,
            R1, R3);
        [x_42, y_42]    =
            TwoBarKinemtics(TH1_d(j,i)-0.01,TH3_d(j,i)+0.01,
            R1, R3);


%           E_x = sqrt(mean([(x_nom-x_11)^2, (x_nom-x_12)^2,
    (x_nom-x_21)^2, (x_nom-x_22)^2, ...
%                            (x_nom-x_31)^2, (x_nom-x_32)^2,
    (x_nom-x_41)^2, (x_nom-x_42)^2] ) );
%           E_y = sqrt(mean([(y_nom-y_11)^2, (y_nom-y_12)^2,
    (y_nom-y_21)^2, (y_nom-y_22)^2, ...
%                            (y_nom-y_31)^2, (y_nom-y_32)^2,
    (y_nom-y_41)^2, (y_nom-y_42)^2] ) );
```

```
            E_x_2 = [(x_nom-x_11)^2, (x_nom-x_12)^2,
                (x_nom-x_21)^2, (x_nom-x_22)^2, ...
                                (x_nom-x_31)^2, (x_nom-x_32)^2,
                                    (x_nom-x_41)^2, (x_nom-x_42)^2] ;
            E_y_2 = [(y_nom-y_11)^2, (y_nom-y_12)^2,
                (y_nom-y_21)^2, (y_nom-y_22)^2, ...
                                (y_nom-y_31)^2, (y_nom-y_32)^2,
                                    (y_nom-y_41)^2, (y_nom-y_42)^2] ;
            E_r   = sqrt(E_x_2 + E_y_2) ;
            E_x   = sqrt(E_x_2);
            E_y   = sqrt(E_y_2);

            E_R_max(j,i) = max(E_r);
            E_R_mean(j,i) = mean(E_r);

            %X(j,i)   = x_nom;   Y(j,i)   = y_nom;
            E_X_max(j,i) = max(E_x);
            E_Y_max(j,i) = max(E_y);
            E_X_mean(j,i) = mean(E_x);
            E_Y_mean(j,i) = mean(E_y);
%           E(j,i)   = sqrt(E_x^2+E_y^2);

        end
    end
```

```
line([0, R1*cos(pi/180*60),
    R1*cos(pi/180*60)+R3*cos(pi/180*30)]-4,...
        [0, R1*sin(pi/180*60),
            R1*sin(pi/180*60)+R3*sin(pi/180*30)]+0.736,...
           'Color','b','LineWidth',0.5);




surf(X-4,Y+0.736,E_R_max,'EdgeColor','none'); view([0 90]);
    colorbar; axis equal ; box on; view(0,90)
%surf(x,y,E_Y_max); view([0 90]); colorbar; axis equal ; box
    on; view(0,90)


axis equal
hold off


Temp = isnan(E_R_max);
Coverage = 1 - sum(Temp(:)==1) / length(E_R_max(:))
Mean = nanmean(E_R_max(:))
STDev = nanstd(E_R_max(:))
```

A.6 Four Link Error Analysis

```
D = 3.4641; R1 = 4; R2 = 4; R3 = 2; R4 = 2; th1_d_I = 90;
    th2_d_I = 30;
% Five link Error Analysis
th1_d = [65:1:93];
```

```
th2_d = 87:1:115;


X   = zeros([length(th2_d),length(th1_d) ]);
Y   = zeros([length(th2_d),length(th1_d) ]);
E_X = zeros([length(th2_d),length(th1_d) ]);
E_Y = zeros([length(th2_d),length(th1_d) ]);
E   = zeros([length(th2_d),length(th1_d) ]);

for i=1:length(th1_d)
    for j=1:length(th2_d)
        [phi3_d, phi4_d, th3_d, th4_d, x_nom, y_nom] =
          FiveLinkAngles(th1_d(i),th2_d(j));
        [phi3_d, phi4_d, th3_d, th4_d, x_11, y_11] =
          FiveLinkAngles(th1_d(i)+0.01,th2_d(j));
        [phi3_d, phi4_d, th3_d, th4_d, x_12, y_12] =
          FiveLinkAngles(th1_d(i)-0.01,th2_d(j));
        [phi3_d, phi4_d, th3_d, th4_d, x_21, y_21] =
          FiveLinkAngles(th1_d(i),th2_d(j)+0.01);
        [phi3_d, phi4_d, th3_d, th4_d, x_22, y_22] =
          FiveLinkAngles(th1_d(i),th2_d(j)-0.01);
        [phi3_d, phi4_d, th3_d, th4_d, x_31, y_31] =
          FiveLinkAngles(th1_d(i)+0.01,th2_d(j)+0.01);
        [phi3_d, phi4_d, th3_d, th4_d, x_32, y_32] =
          FiveLinkAngles(th1_d(i)-0.01,th2_d(j)-0.01);
        [phi3_d, phi4_d, th3_d, th4_d, x_41, y_41] =
          FiveLinkAngles(th1_d(i)+0.01,th2_d(j)-0.01);
```

```
        [phi3_d, phi4_d, th3_d, th4_d, x_42, y_42] =
            FiveLinkAngles(th1_d(i)-0.01,th2_d(j)+0.01);

        E_x = sqrt(mean([(x_nom-x_11)^2, (x_nom-x_12)^2,
            (x_nom-x_21)^2, (x_nom-x_22)^2, ...
                        (x_nom-x_31)^2, (x_nom-x_32)^2,
                            (x_nom-x_41)^2, (x_nom-x_42)^2]
                            ));
        E_y = sqrt(mean([(y_nom-y_11)^2, (y_nom-y_12)^2,
            (y_nom-y_21)^2, (y_nom-y_22)^2, ...
                        (y_nom-y_31)^2, (y_nom-y_32)^2,
                            (y_nom-y_41)^2, (y_nom-y_42)^2]
                            ));

        X(j,i)   = x_nom;    Y(j,i)   = y_nom;
        E_X(j,i) = E_x; E_Y(j,i) = E_y;
        E(j,i)   = sqrt(E_x^2+E_y^2);

    end
end

figure(3)
hold on


plot(X,Y, 'r','Linewidth',2);
```

```
th1_d = [93];
th2_d = [87:1:115];

X = zeros([length(th2_d),length(th1_d) ]);
Y = zeros([length(th2_d),length(th1_d) ]);

for i=1:length(th1_d)
    for j=1:length(th2_d)
        [phi3_d, phi4_d, th3_d, th4_d, x, y] =
            FiveLinkAngles(th1_d(i),th2_d(j));
        X(j,i) = x; Y(j,i) = y;
    end
end

plot(X,Y, 'r','Linewidth',2);

th1_d = [65:1:93];;
th2_d = [87:1:115];

X = zeros([length(th2_d),length(th1_d) ]);
Y = zeros([length(th2_d),length(th1_d) ]);

for i=1:length(th1_d)
    for j=1:length(th2_d)
        [phi3_d, phi4_d, th3_d, th4_d, x, y] =
            FiveLinkAngles(th1_d(i),th2_d(j));
        X(j,i) = x; Y(j,i) = y;
```

```
    end
end

plot(X,Y, 'r.','Linewidth',2);




line([0, R1*cos(pi/180*th1_d_I),
   R1*cos(pi/180*th1_d_I)+R3*cos(pi/180*th2_d_I) ,...
      R1*cos(pi/180*th1_d_I)+2*R3*cos(pi/180*th2_d_I) ,...
      2*(R1*cos(pi/180*th1_d_I)+R3*cos(pi/180*th2_d_I))], ...
     [0, R1*sin(pi/180*th1_d_I),
        R1*sin(pi/180*th1_d_I)+R3*sin(pi/180*th2_d_I) ,...
      R1*sin(pi/180*th1_d_I), 0],...
       'Color','b','LineWidth',4);

rectangle('Position',[3.4641/2-1.25/2, 5-0.3, 1.25, 1]);




figure(1)
subplot(2,1,1), plot(th2_d, X),
   legend('30','40','50','60','64');
```

```
subplot(2,1,2), plot(th2_d, Y),
   legend('30','40','50','60','64');

figure(3)
% plot(X, Y, '-');
hold on;
line([0, R1*cos(pi/180*th1_d_I),
   R1*cos(pi/180*th1_d_I)+R3*cos(pi/180*th2_d_I) ,...
       R1*cos(pi/180*th1_d_I)+2*R3*cos(pi/180*th2_d_I) ,...
       2*(R1*cos(pi/180*th1_d_I)+R3*cos(pi/180*th2_d_I))], ...
      [0, R1*sin(pi/180*th1_d_I),
         R1*sin(pi/180*th1_d_I)+R3*sin(pi/180*th2_d_I) ,...
       R1*sin(pi/180*th1_d_I), 0],...
        'Color','b','LineWidth',4);



rectangle('Position',[3.732-1.7/2, 4.464-0.5, 1.25, 1]);

sz_th2 = length(th2_d);
sz_th1 = length(th1_d);
hold on;
 plot(X(:,1),Y(:,1), 'r', ...
%       X(:,sz_th1),Y(:,sz_th1),'g',   ...
%       X(sz_th2,:),Y(sz_th2,:),'b',   ...
%      X(1,:),Y(1,:), 'r',
       'Linewidth',2);
```

```
axis equal
```

A.7 Four Link Inverse Error Analysis

```
D = 3.4641; R1 = 4; R2 = 4; R3 = 2; R4 = 2; th1_d_I = 90;
    th2_d_I = 30;

% Five link Error Analysis

clear E* X* Y* x* y* T*
figure(1)
% clf
hold on


x = [D/2-1.25/2: 0.01 : D/2+1.25/2 ];
y = [4.7         : 0.01 : 5.7];

X = zeros([length(y),length(x) ]);
Y = zeros([length(y),length(x) ]);
E_X = zeros([length(y),length(x) ]);
E_Y = zeros([length(y),length(x) ]);
E   = zeros([length(y),length(x) ]);
E_R_max  = zeros([length(y),length(x) ]);
```

```
E_R_mean = zeros([length(y),length(x) ]);
E_X_max  = zeros([length(y),length(x) ]);
E_Y_max  = zeros([length(y),length(x) ]);
E_X_mean = zeros([length(y),length(x) ]);
E_Y_mean = zeros([length(y),length(x) ]);

TH1_d = zeros([length(y),length(x) ]);
TH2_d = zeros([length(y),length(x) ]);

for i=1:length(x)
    for j=1:length(y)
        [th1_d, th2_d] = FiveLinkAngles_Inv(x(i),y(j));
%         if (~isreal(th1_d) || ~isreal(th2_d))
%             [x(i),y(j), th1_d, th2_d ]
%             th1_d = NaN;
%             th2_d = NaN;
%         end
        X(j,i) = x(i); Y(j,i) = y(j);
        TH1_d(j,i) = th1_d; TH2_d(j,i) = th2_d;
    end
end

for i=1:length(x)
    for j=1:length(y)
        [phi3_d, phi4_d, th3_d, th4_d, x_nom, y_nom] =
            FiveLinkAngles(TH1_d(j,i),TH2_d(j,i));
```

```
        [phi3_d, phi4_d, th3_d, th4_d, x_11, y_11] =
           FiveLinkAngles(TH1_d(j,i)+0.01,TH2_d(j,i));
        [phi3_d, phi4_d, th3_d, th4_d, x_12, y_12] =
           FiveLinkAngles(TH1_d(j,i)-0.01,TH2_d(j,i));
        [phi3_d, phi4_d, th3_d, th4_d, x_21, y_21] =
           FiveLinkAngles(TH1_d(j,i),TH2_d(j,i)+0.01);
        [phi3_d, phi4_d, th3_d, th4_d, x_22, y_22] =
           FiveLinkAngles(TH1_d(j,i),TH2_d(j,i)-0.01);
        [phi3_d, phi4_d, th3_d, th4_d, x_31, y_31] =
           FiveLinkAngles(TH1_d(j,i)+0.01,TH2_d(j,i)+0.01);
        [phi3_d, phi4_d, th3_d, th4_d, x_32, y_32] =
           FiveLinkAngles(TH1_d(j,i)-0.01,TH2_d(j,i)-0.01);
        [phi3_d, phi4_d, th3_d, th4_d, x_41, y_41] =
           FiveLinkAngles(TH1_d(j,i)+0.01,TH2_d(j,i)-0.01);
        [phi3_d, phi4_d, th3_d, th4_d, x_42, y_42] =
           FiveLinkAngles(TH1_d(j,i)-0.01,TH2_d(j,i)+0.01);


%         E_x = sqrt(mean([(x_nom-x_11)^2, (x_nom-x_12)^2,
    (x_nom-x_21)^2, (x_nom-x_22)^2, ...
%                          (x_nom-x_31)^2, (x_nom-x_32)^2,
    (x_nom-x_41)^2, (x_nom-x_42)^2] ) );
%         E_y = sqrt(mean([(y_nom-y_11)^2, (y_nom-y_12)^2,
    (y_nom-y_21)^2, (y_nom-y_22)^2, ...
%                          (y_nom-y_31)^2, (y_nom-y_32)^2,
    (y_nom-y_41)^2, (y_nom-y_42)^2] ) );
```

```
            E_x_2 = [(x_nom-x_11)^2, (x_nom-x_12)^2,
                (x_nom-x_21)^2, (x_nom-x_22)^2, ...
                                (x_nom-x_31)^2, (x_nom-x_32)^2,
                                    (x_nom-x_41)^2, (x_nom-x_42)^2] ;
            E_y_2 = [(y_nom-y_11)^2, (y_nom-y_12)^2,
                (y_nom-y_21)^2, (y_nom-y_22)^2, ...
                                (y_nom-y_31)^2, (y_nom-y_32)^2,
                                    (y_nom-y_41)^2, (y_nom-y_42)^2] ;
            E_r   = sqrt(E_x_2 + E_y_2) ;
%             if ~isreal(E_r)
%                 [x_nom, y_nom, TH1_d(j,i),TH2_d(j,i)]
%             end
            E_x   = sqrt(E_x_2);
            E_y   = sqrt(E_y_2);

            E_R_max(j,i) = max(E_r);
            E_R_mean(j,i) = mean(E_r);

            %X(j,i)    = x_nom;    Y(j,i)    = y_nom;
            E_X_max(j,i) = max(E_x);
            E_Y_max(j,i) = max(E_y);
            E_X_mean(j,i) = mean(E_x);
            E_Y_mean(j,i) = mean(E_y);
%             E(j,i)    = sqrt(E_x^2+E_y^2);


    end
```

```
end



A_desired = polyarea(...
      [(D/2-1.25/2), (D/2+1.25/2), (D/2+1.25/2),
         (D/2-1.25/2), (D/2-1.25/2)], ...
      [ (5-0.3)     , (5-0.3)     , (5-0.3+1)    , (5-0.3+1),
         (5-0.3)]);


xmid = D/2;
dxmidu = D - 1.6127*2;
dxmidm = D - 1.1488*2;
dxmidl = D - 1.6837*2;
yu = 5.7;
ym = 5.5262;
yl = 4.7;
A_effective = polyarea( ...
      [1.6127, 1.1488, 1.6837, ...
       1.6837+dxmidl, 1.1488+dxmidm, 1.6127+dxmidu, 1.6127
          ],...
      [5.7, ym, 4.7, 4.7, ym, 5.7, 5.7 ]);


% E_mean = sum(nansum(E))*0.05*0.05 / A_effective;
```

```
line([0, R1*cos(pi/180*th1_d_I),
   R1*cos(pi/180*th1_d_I)+R3*cos(pi/180*th2_d_I) ,...
      R1*cos(pi/180*th1_d_I)+2*R3*cos(pi/180*th2_d_I) ,...
      2*(R1*cos(pi/180*th1_d_I)+R3*cos(pi/180*th2_d_I))], ...
     [0, R1*sin(pi/180*th1_d_I),
        R1*sin(pi/180*th1_d_I)+R3*sin(pi/180*th2_d_I) ,...
      R1*sin(pi/180*th1_d_I), 0],...
       'Color','b','LineWidth',0.5);

rectangle('Position',[3.4641/2-1.25/2, 5-0.3, 1.25, 1]);
surf(x,y,E_R_max,'EdgeColor','none'); view([0 90]);
   colorbar; axis equal ; box on; view(0,90)

axis equal
hold off

Temp = isnan(E_R_max);
Coverage = 1 - sum(Temp(:)==1) / length(E_R_max(:))
Mean = nanmean(E_R_max(:))
STDev = nanstd(E_R_max(:))
```

A.8 Top Level Path Controller S-Function

```
function msfnc_ChangeInput(block)
% Level-2 M file S-Function for times two demo.
%   Copyright 1990-2004 The MathWorks, Inc.
%   $Revision: 1.1.6.2 $
```

```
  setup(block);

%endfunction

function setup(block)

  %% Register number of input and output ports
  block.NumInputPorts  = 2;
  block.NumOutputPorts = 1;

  %% Register parameters
  block.NumDialogPrms     = 2;
  block.DialogPrmsTunable = {'Tunable','Tunable'};

  %% Setup functional port properties to dynamically
  %% inherited.
  block.SetPreCompInpPortInfoToDynamic;
  block.SetPreCompOutPortInfoToDynamic;

  block.InputPort(1).Dimensions        =
     [block.DialogPrm(2).Data 3];       %The list of way
     points
  block.InputPort(1).DirectFeedthrough = true;
  block.InputPort(2).Dimensions        = [1 3];       %The
     current point
  block.InputPort(2).DirectFeedthrough = false;
```

```
    block.OutputPort(1).Dimensions        = [1 3];
    %block.OutputPort(2).Dimensions        = 1;
    %% Set block sample time to inherited
    block.SampleTimes = [-1 0];

%   %% Setup Dwork
%    block.NumContStates = 1;

    %% Set the block simStateComliance to default (i.e., same
       as a built-in block)
    block.SimStateCompliance = 'DefaultSimState';

    %% Run accelerator on TLC
    block.SetAccelRunOnTLC(true);

    %% Register methods
    block.RegBlockMethod('Outputs',                 @Output);
    block.RegBlockMethod('PostPropagationSetup',
       @DoPostPropSetup);
    block.RegBlockMethod('Start',                   @Start);
    block.RegBlockMethod('InitializeConditions',
       @InitializeConditions);
    block.RegBlockMethod('SetInputPortDimensions',
       @SetInpPortDims);
```

```
%endfunction

function DoPostPropSetup(block)
  block.NumDworks = 3;

  block.Dwork(1).Name            = 'CurrPComm';
  block.Dwork(1).Dimensions      = 3;
  block.Dwork(1).DatatypeID      = 0;      % double
  block.Dwork(1).Complexity      = 'Real'; % real
  block.Dwork(1).UsedAsDiscState = true;

  block.Dwork(2).Name            = 'PCommCnt';
  block.Dwork(2).Dimensions      = 1;
  block.Dwork(2).DatatypeID      = 0;      % double
  block.Dwork(2).Complexity      = 'Real'; % real
  block.Dwork(2).UsedAsDiscState = true;

  block.Dwork(3).Name            = 'PrevTStmp';
  block.Dwork(3).Dimensions      = 1;
  block.Dwork(3).DatatypeID      = 0;      % double
  block.Dwork(3).Complexity      = 'Real'; % real
  block.Dwork(3).UsedAsDiscState = true;

  %% Register all tunable parameters as runtime parameters.
  block.AutoRegRuntimePrms;

%endfunction
```

```
function Start(block)
    block.Dwork(1).Data = block.InputPort(1).Data(1,:);
    block.OutputPort(1).Data = transpose(block.Dwork(1).Data);
   %block.OutputPort(2).Data = 0;
    block.Dwork(2).Data = 1;
    block.Dwork(3).Data = 0;
%endfunction

function InitializeConditions(block)
% block.InputPort(1).Data(1,:)
%    block.Dwork(1).Data = block.InputPort(1).Data(1,:);
%    block.Dwork(2).Data = 1;
%endfunction

function Output(block)
  Cnt = block.Dwork(2).Data;
  Pcurr = sqrt(block.InputPort(2).Data(1)^2 +
      block.InputPort(2).Data(2)^2);
  Pcomm = sqrt(block.Dwork(1).Data(1)^2 +
      block.Dwork(1).Data(2)^2);
  %x_current = block.InputPort(2).Data(1);
  %y_current = block.InputPort(2).Data(2);

  %x_command = block.Dwork(1).Data(1);
  %y_command = block.Dwork(1).Data(2);
```

```
%Error_x = abs(x_command-x_current);
%Error_y = abs(y_command-y_current);
%Error   = max(Error_x, Error_y);

Error = abs( Pcurr - Pcomm );
%block.OutputPort(2).Data = Error;
Duration = block.CurrentTime - block.Dwork(3).Data;

%if (( Error <  block.DialogPrm(1).Data)   &&
   Duration>0.01)
if ( (Error <  block.DialogPrm(1).Data) &&
   (block.CurrentTime>0) ) %%  && Duration>0.005)
    if block.Dwork(2).Data <
       size(block.InputPort(1).Data(:,1), 1 )
        block.Dwork(2).Data = Cnt + 1;
        block.Dwork(3).Data = block.CurrentTime;
    end
    %block.Dwork(1).Data =
       block.InputPort(1).Data(block.Dwork(2).Data,:);
end

block.Dwork(1).Data =
   block.InputPort(1).Data(block.Dwork(2).Data,:);
block.OutputPort(1).Data = transpose(block.Dwork(1).Data);

%  if block.CurrentTime > 5
```

```
%         block.OutputPort(1).Data =
    block.InputPort(1).Data(2,:);
%   end


%endfunction
```

A.9 Path Generation Function

```
function PathMatrix = WayPointsToPath( WayP, NoSteps )

NoWayP = size(WayP, 1);
PathMatrix = [];

for i = 1 : NoWayP-1
    InitWayP = WayP(i,   :);
    FinlWayP = WayP(i+1,:);

    %Dx = InitWayP(1)-FinlWayP(1)
    X = linspace(InitWayP(1),FinlWayP(1), NoSteps);
    Y = linspace(InitWayP(2),FinlWayP(2), NoSteps);
    Z = (i+1) * ones(size(X));
    PathMatrix_temp = [X' , Y' , Z'];
    PathMatrix = [PathMatrix; PathMatrix_temp];
end
```

APPENDIX B

AMDAHL'S LAW

Amdahl's Law measures the speedup of a computer program (or function) as a result of distributing this program over several processors.

Basically, Amdahl's Law answers the question: "what is the benefit of using different versions of distributed control as compared to centralized control?"

It is given by

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}} \tag{45}$$

where

S : Speedup

P : Number of operations that can be parallelized

N : Number of processors

APPENDIX C

CYCLOMATIC COMPLEXITY

The *Cyclomatic Complexity* metric is one of the most important evaluation metrics in software engineering. It is a measure of the number of decisions made within a computer function. One of the software structured testing key requirement is that all the software's decision outcomes must be exercised independently during testing. The number of tests required for a software module is equal to the cyclomatic complexity of that module [143]. Since the main function of a control architecture is to make automated decisions regarding the whole system, cyclomatic complexity presents a very appealing metric. Intelligent control architectures often have to choose a course of action (either from among a preprogrammed repository, or using active learning methods). The more decision options a control architecture has, the more complex it gets.

Cyclomatic complexity is given by

$$M = E - N + 2P \tag{46}$$

where

M : Cyclomatic complexity

E : Number of edges

N : Number of nodes

P : Number of connected components (exit nodes)

A cyclomatic complexity of less than 10 is an acceptable cyclomatic complexity. If the cyclomatic complexity is between 10 and 20, it counts as moderate complexity.

They range from 20 to 40 counts as an extremely complex system. Designs with cyclomatic complexity larger than 40 are unacceptable designs.

REFERENCES

[1] "Capability engineering." Brochure.

[2] "The Web Graph Database," http://infogrid.org/wiki/Reference/WhatIsMetaModeling [cited March 2013].

[3] "Robot riddle solved," *New Scientist Magazine*, p. 34, August 1988.

[4] "Congressional budget data," tech. rep., US House of Congress, 2005.

[5] "Title 10–armed forces," *U.S. Code, Sub-title B–Army, Part I–Organization*, 2007.

[6] "Smart grid system report," tech. rep., U.S. Department of Energy, July 2009.

[7] Albus, J., "A reference model architecture for intelligent unmanned ground vehicles," in *Proceedings of the SPIE 16th Annual International Symposium on Aerospace/Defense Sensing, Simulation and Controls*, pp. 303–310, Citeseer, 2002.

[8] Albus, J. S. and Barbera, A. J., "Rcs: A cognitive architecture for intelligent multi-agent systems," *Annual Reviews in Control*, vol. 29, no. 1, pp. 87 – 99, 2005.

[9] Anderson, P., "Perspective: Complexity theory and organization science," *organization Science*, vol. 10, no. 3, pp. 216–232, 1999.

[10] Atkinson, C. and Kuhne, T., "Model-driven development: a metamodeling foundation," *Software, IEEE*, vol. 20, no. 5, pp. 36–41, 2003.

[11] Aved'yan, E., Parks, P. C., and Mason, J., *Learning systems*. Springer-Verlag New York, Inc., 1995.

[12] Bajd, T., Mihelj, M., Lenarčič, J., Stanovnik, A., and Munih, M., *Robotics*, vol. 43. Springer, 2010.

[13] Bakule, L. and others, "Decentralized control and communication," *Annual Reviews in Control*, 2012.

[14] Bakule, L., "Decentralized control: An overview," *Annual Reviews in Control*, vol. 32, no. 1, pp. 87–98, 2008.

[15] Behbahani, A., "Adaptive distributed intelligent control architecture for future propulsion systems," in *61st Meeting Of The Society For Machinery Failure Prevention Technology, Virginia Beach, Virginia, Paper: Session 5B-Cape Colony B*, 2007.

[16] Behbahani, A. R., "Adaptive distributed intelligent control architecture for future propulsion systems (preprint)," tech. rep., DTIC Document, 2007.

[17] Bellman, R. E., *An introduction to artificial intelligence: Can computers think?* Boyd & Fraser Publishing Company, 1978.

[18] BENASKEUR, A., MCGUIRE, P., BRENNAN, R., LIGGINS, G., and WOJCIK, P., "A distributed intelligent tactical sensor management system," *International Journal of Intelligent Control and Systems*, vol. 12, no. 2, pp. 97–106, 2007.

[19] Benioff, M. and Lazowska, E., "Pitac report to the president on computational science: Ensuring americas competitiveness," 2005.

[20] Bergman, T. L., Lavine, A. S., Incropera, F. P., and DeWitt, D. P., *Fundamentals of heat and mass transfer*. Wiley, 2011.

[21] Bondi, A. B., "Characteristics of scalability and their impact on performance," in *Proceedings of the 2nd international workshop on Software and performance*, pp. 195–203, ACM, 2000.

[22] Bondi, A. B., "Characteristics of scalability and their impact on performance," in *Proceedings of the 2nd international workshop on Software and performance*, pp. 195–203, ACM, 2000.

[23] Booch, G., Rumbaugh, J., and Jacobson, I., *Unified Modeling Language User Guide, The (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005.

[24] Boskovic, J. D., Prasanth, R., and Mehra, R. K., "A multi-layer autonomous intelligent control architecture for unmanned aerial vehicles," *Journal of Aerospace Computing, Information, and Communication*, vol. 1, no. 12, pp. 605–628, 2004.

[25] Boyd, S., Baratt, C., and Norman, S., "Linear controller design: Limits of performance via convex optimization," *Proceedings of the IEEE*, vol. 78, no. 3, pp. 529–574, 2002.

[26] Brataas, G. and Hughes, P., "Exploring architectural scalability," in *ACM SIGSOFT Software Engineering Notes*, vol. 29, pp. 125–129, ACM, 2004.

[27] Brogan, W. L., *Modern Control Theory*. Prentice Hall, third edition ed., 1990.

[28] Burch, J., Passerone, R., and Sangiovanni-Vincentelli, A., "Modeling techniques in design-by-refinement methodologies," *System Specification & Design Languages*, pp. 283–292, 2004.

[29] Capella, J., Bonastre, A., and Ors, R., "An advanced and distributed control architecture based on intelligent agents and neural networks," in *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2003. Proceedings of the Second IEEE International Workshop on*, pp. 278–283, IEEE, 2003.

[30] Carbonari, R., Spillner, K., Pilat, M., Wilkins, D. C., and Tatem, P. A., "Supervisory control system for ship damage control: Volume 3-human computer interface and visualization," tech. rep., DTIC Document, 2001.

[31] Cares, J. R., *Distributed networked operations: The foundations of network centric warfare*. Iuniverse Inc, 2006.

[32] Carle S. Droste, J. E. W., "The general dynamics case study on the f-16 fly-by-wire flight control system," tech. rep., AIAA Professional Study Series, April 2003.

[33] Carlock, P. G. and Fenton, R. E., "System of systems (sos) enterprise systems engineering for information-intensive organizations," *Systems Engineering*, vol. 4, no. 4, pp. 242–261, 2001.

[34] Cassandras, C. and Li, W., "Sensor networks and cooperative control," pp. 4237–4238, Dec. 2005.

[35] Chavdarov, I., "Kinematics and force analysis of a five-link mechanism by the four spaces jacoby matrix," *Problems of Engineering Cybernatics and Robotics*, vol. 55, 2005. Bulgarian Academy of Sciences.

[36] Clements, P., "A survey of architecture description languages," in *Proceedings of the 8th international workshop on software specification and design*, p. 16, IEEE Computer Society, 1996.

[37] Collier, S. E., "Ten steps to a smarter grid," *Industry Applications Magazine, IEEE*, vol. 16, no. 2, pp. 62–68, 2010.

[38] Covanich, W., McFarlane, D., and Farid, A. M., "Guidelines for evaluating the ease of reconfiguration of manufacturing systems," in *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, pp. 1214–1219, IEEE, 2008.

[39] D. S. Tavkhelidze, N. S. D., "Kinematic analysis of five-link spherical mechanisms," *Mechanism and Machine Theory*, vol. 9, pp. 181–190, 1974.

[40] Dandashi, F., "Dod architecture framework overview," tech. rep., 2003, 2003.

[41] Davitashvili, N. and Gelashvili, O., "Synthesis of a Spatial Five-Link Mechanism with Two Degrees of Freedom According to the Given Laws of Motion," *Proceedings of EUCOMES 08*, pp. 159–166, 2008.

[42] Davitashvili, N., "Designing five-link hinged mechanisms taking into account the angle drive," *Mechanism and machine theory*, vol. 18, no. 6, pp. 481–489, 1983.

[43] De Bernardinisa, L., Pinelloa, C., and Sgroia, A., "Platform-Based Design for Embedded Systems,"

[44] de Niz, D., "Diagrams and languages for model-based software engineering of embedded systems: UML and AADL."

[45] Dieter, G., *Engineering Design: A Materials and Processing Approach*. McGraw-Hill Science/Engineering/Math, third edition ed., 1999.

[46] Doerry, N. and Amy, J., "Functional decomposition of a medium voltage dc integrated power system," in *ASNE Shipbuilding in Support of the Global War on Terrorism Symposium, Biloxi, MS*, 2008.

[47] Doerry, N., "Designing electrical power systems for survivability and quality of service," vol. 119, pp. 25–34, Wiley Online Library, 2007.

[48] Doerry, N., "Next generation integrated power systems (ngips) for the future fleet," in *IEEE Electric Ship Technologies Symposium*, 2009.

[49] Doerry, N. and McCoy, K., "Next generation integrated power system: Ngips technology development roadmap," tech. rep., DTIC Document, 2007.

[50] Doerry, N. H. and Clayton, D. H., "Shipboard electrical power quality of service," in *Electric Ship Technologies Symposium, 2005 IEEE*, pp. 274–279, IEEE, 2005.

[51] Doerry, N. H. and Fireman, H., "Designing all electric ships," in *19th International Marine Design Conference*, pp. 475–497, 2006.

[52] Duboc, L., Rosenblum, D. S., and Wicks, T., "A framework for modelling and analysis of software systems scalability," in *Proceedings of the 28th international conference on Software engineering*, pp. 949–952, ACM, 2006.

[53] Dunnington, L., Stevens, H., and Grater, G., "Integrated engineering plant for future naval combatants-technology assessment and demonstration roadmap," *Systems Engineering Group, Engineering Technology Center, Marine Technology Division, Anteon Corporation*, 2003.

[54] Farid, A., "Facilitating ease of system reconfiguration through measures of manufacturing modularity," *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 222, no. 10, pp. 1275–1288, 2008.

[55] Fielding, C., "The design of fly-by-wire flight control systems," *Aeronautical Journal*, vol. 105, no. 1051, pp. 543–549, 2001.

[56] Garamone, J., "Joint Vision 2020 Emphasizes Full-spectrum Dominance." Online, URL: http://www.defense.gov/news/newsarticle.aspx?id=45289 2000.

[57] Gellings, C. W., *The smart grid: enabling energy efficiency and demand response*. The Fairmont Press, Inc., 2009.

[58] Girard, A. R., de Sousa, J. B., Misener, J. A., and Hedrick, J. K., "A control architecture for integrated cooperative cruise control and collision warning systems," in *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*, vol. 2, pp. 1491–1496, IEEE, 2001.

[59] Grois, E., Wilkins, D. C., Earman, I., Bobak, M., and Brady, A., "Supervisory control system for ship damage control: Volume 4-intelligent reasoning," tech. rep., DTIC Document, 2001.

[60] Group, O. M., "http://www.omg.org."

[61] Gupta, R. P. and Varma, R. K., "Agent based software integration at distribution control center," in *Power Engineering Society General Meeting, 2004. IEEE*, pp. 522–527, IEEE, 2004.

[62] Hamman, M., Wilkins, D. C., Tatem, P. A., and Williams, F. W., "Supervisory control system for ship damage control: Volume 7: Software architecture," tech. rep., DTIC Document, 2001.

[63] Haslego, C. and Polley, G., "Designing plate-and-frame heat exchangers," *Chemical engineering progress*, vol. 98, no. 9, pp. 32–37, 2002.

[64] Heo, J. and Lee, K., "A multi-agent system-based intelligent control system for a power plant," pp. 1050–1055 Vol. 2, June 2005.

[65] Hipel, K. W., Jamshidi, M. M., Tien, J. M., and White, C. C., "The future of systems, man, and cybernetics: application domains and research methods," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 37, no. 5, pp. 726–743, 2007.

[66] Hossack, J. A., Menal, J., McArthur, S. D., and McDonald, J. R., "A multiagent architecture for protection engineering diagnostic assistance," *Power Systems, IEEE Transactions on*, vol. 18, no. 2, pp. 639–647, 2003.

[67] Huang, H.-M., Hira, R., and Quintero, R., "A submarine maneuvering system demonstration based on the nist real-time control system reference model," in *Intelligent Control, 1993., Proceedings of the 1993 IEEE International Symposium on*, pp. 376–381, Aug 1993.

[68] Hughes, J., "Intelligrid architecture application guide," EPRI Technical Report Product ID 1013610, EPRI, December 2006.

[69] Jackson, E. and Sztipanovits, J., "Formalizing the structural semantics of domain-specific modeling languages," *Journal of Software and Systems Modeling (SoSym)*, vol. In Press, 2009.

[70] Jamshidi, M., "Theme of the IEEE SMC 2005, Waikoloa, Hawaii, USA," http://ieeesmc2005.unm.edu/.

[71] Jamshidi, M., "Large-scale systems, modeling and control, volume 9 of system science and engineering," 1983.

[72] Jamshidi, M., Sahin, F., and Nanayakkara, T., *Intelligent control systems with an introduction to system of systems engineering.* CRC, 2009.

[73] Jamshidi, M., "Large-scale systems: modeling, control, and fuzzy logic," 1996.

[74] Johnson, E. N. and Schrage, D. P., "The georgia tech unmanned aerial research vehicle: Gtmax," in *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, pp. 11–14, Citeseer, 2003.

[75] Jung, J. and Liu, C.-C., "Multi-agent system technologies and an application for power system vulnerability," in *Power Engineering Society General Meeting, 2004. IEEE*, pp. 61–64, IEEE, 2004.

[76] Junpu, W., Hao, C., Yang, X., and Shuhui, L., "An architecture of agent-based intelligent control systems," in *Intelligent Control and Automation, 2000. Proceedings of the 3rd World Congress on*, vol. 1, pp. 404–407, IEEE, 2000.

[77] Kays, W. M. and London, A. L., *Compact heat exchangers.* McGraw-Hill, New York, NY, 1984.

[78] Kenneth E. Kendall, J. E. K., *Systems Analysis and Design.* Prentice Hall, eighth edition ed., 2010.

[79] Kortenkamp, D., Burridge, R., Bonasso, P., Schrenkenghoist, D., and Hudson, M., "An intelligent software architecture for semiautonomous robot control," in *Autonomy Control Software Workshop, Autonomous Agents*, vol. 99, pp. 36–43, 1999.

[80] Kosakaya, J., Kobayashi, A., and Yamaoka, K., "Cooperative multi-agent-based control technology for supervisory control and data-acquisition systems," in *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA'03. IEEE Conference*, vol. 2, pp. 19–26, IEEE, 2003.

[81] Kotov, V., *Systems of systems as communicating structures.* Hewlett Packard Laboratories, 1997.

[82] Kurzweil, R., Schneider, M. L., and Schneider, M. L., *The age of intelligent machines*, vol. 579. MIT press Cambridge, 1990.

[83] Langer, A. M., *Analysis and Design of Information Systems*. Springer, third edition ed., 2010.

[84] Lee, E., Neuendorffer, S., and Wirthlin, M., "Actor-oriented design of embedded hardware and software systems," *JOURNAL OF CIRCUITS SYSTEMS AND COMPUTERS*, vol. 12, no. 3, pp. 231–260, 2003.

[85] Li, Y., Balchanos, M., Nairouz, B., Weston, N., and Mavris, D., "Modeling and simulation of integrated intelligent systems," in *Simulation Conference, 2008. WSC 2008. Winter*, pp. 1225–1233, IEEE, 2008.

[86] Louden, J. J. and Deputy, N. T., "Total ownership cost," *NAVSEA Naval Sea Systems Command*, 2000.

[87] Luckham, D., Vera, J., and Meldal, S., "Three concepts of system architecture," 1995.

[88] Maddux, M. K. C. and Jain, D. S. C., "Cae for the manufacturing engineer: the role of process simulation in concurrent engineering," *MATERIAL AND MANUFACTURING PROCESS*, vol. 1, no. 3-4, pp. 365–392, 1986.

[89] Marris, E., "Energy: Upgrading the grid," *Nature*, vol. 454, pp. 570–573, July 2008.

[90] McArthur, S. and Davidson, E., "Multi-agent systems for diagnostic and condition monitoring applications," in *Power Engineering Society General Meeting, 2004. IEEE*, pp. 50–54, IEEE, 2004.

[91] Medvidovic, N. and Taylor, R., "A classification and comparison framework for software architecture description languages," *Software Engineering, IEEE Transactions on*, vol. 26, no. 1, pp. 70–93, 2000.

[92] Meystel, A. M. and Albus, J. S., *Intelligent Systems: Architecture, Design, and Control*. John Wiley and Sons INC, 2002.

[93] Montroll, M. and McDermott, E., "Capability-based acquisition in the missile defense agency: Independent research project," tech. rep., DTIC Document, 2003.

[94] Moor, T. and Raisch, J., "Supervisory control of hybrid systems within a behavioural framework," *Systems & control letters*, vol. 38, no. 3, pp. 157–166, 1999.

[95] Nakamura, Y. and Ghodoussi, M., "Dynamics computation of closed-link robot mechanisms with nonredundant and redundant actuators," *Robotics and Automation, IEEE Transactions on*, vol. 5, no. 3, pp. 294–302, 1989.

[96] NETL Modern Grid Initiative and others, "A vision for the modern grid," tech. rep., United States Department of Energy, National Energy Technology Laboratory, 2007.

[97] Nguyen, V., Jeong, M., Ahn, S., Moon, S., and Baik, S., "A robust localization method for mobile robots based on ceiling landmarks," *Modeling Decisions for Artificial Intelligence*, pp. 422–430, 2007.

[98] NIAN, X.-H. and CAO, L., "Bmi approach to the interconnected stability and cooperative control of linear systems," *Acta Automatica Sinica*, vol. 34, no. 4, pp. 438–444, 2008.

[99] Nicolescu, G., *Model-Based Design for Embedded Systems*, vol. 1. CRC, 2009.

[100] Oden, J., Belytschko, T., Hughes, T., Johnson, C., Keyes, D., Laub, A., Petzold, L., Srolovitz, D., and Yip, S., "Revolutionizing engineering science through simulation: A report of the national science foundation blue ribbon panel on simulation-based engineering science," *Arlington, VA: National Science Foundation*, 2006.

[101] of Defense Architecture Framework Working Group, D. and others, "Dod architecture framework, version 1.5," *Department of Defense, USA*, 2007.

[102] Olfati-Saber, R. and Murray, R. M., "Distributed cooperative control of multiple vehicle formations using structural potential functions," in *IFAC World Congress*, pp. 346–352, 2002.

[103] Pahl, G., *Engineering Design: A Systematic Approach*. Springer Verlag, 2007.

[104] Pei, R., "System of systems integration (sosi)-a" smart" way of acquiring army c412ws systems," in *Summer Computer Simulation Conference*, pp. 574–579, Society for Computer Simulation International; 1998, 2000.

[105] Picón-Núñez, M., Martínez-Rodríguez, G., and López-Robles, J., "Alternative design approach for multipass and multi-stream plate heat exchangers for use in heat recovery systems," *Heat transfer engineering*, vol. 27, no. 6, pp. 12–21, 2006.

[106] Polycarpou, M. M., Yang, Y., and Passino, K. M., "Cooperative control of distributed multi-agent systems," *IEEE Control Systems Magazine*, 2001.

[107] Rawlings, J., "Tutorial: model predictive control technology," *American Control Conference, 1999. Proceedings of the 1999*, vol. 1, pp. 662–676 vol.1, 1999.

[108] Rawlings, J., "Tutorial overview of model predictive control," *IEEE Control Systems Magazine*, vol. 20, no. 3, pp. 38–52, 2000.

[109] Ren, W. and Beard, R., *Distributed consensus in multi-vehicle cooperative control: theory and applications*. Springer, 2007.

[110] Ren, W. and Cao, Y., "Simulation and experimental study of consensus algorithms for multiple mobile robots with information feedback," *Intelligent Automation and Soft Computing*, vol. 14, no. 1, p. 73, 2008.

[111] Rich, E. and Knight, K., *Artificial intelligence*. Computer Science Series. McGraw-Hill, second edition ed., 1991.

[112] Rolander, N., Pekala, M., and Scheidt, D., "A software simulation testbed to evaluate next-generation control algorithms," in *Americal Society of Naval Engineers (ASNE) Automation and Control Symposium*, 2007.

[113] Rosendall, P., Pekala, M., and Scheidt, D., "Reconfiguring connected resource distribution systems," in *Computational Intelligence in Control and Automation (CICA), 2011 IEEE Symposium on*, pp. 94–101, IEEE, 2011.

[114] Russell, S. J., Norvig, P., Davis, E., Russell, S. J., and Russell, S. J., *Artificial intelligence: a modern approach*. Prentice hall Upper Saddle River, NJ, 2010.

[115] Sage, A. P. and Cuppan, C. D., "On the systems engineering and management of systems of systems and federations of systems," *Information Knowledge Systems Management*, vol. 2, no. 4, pp. 325–345, 2001.

[116] Sage, A. and Rouse, W., eds., *Handbook of Systems Engineering and Management*. Wiley-Interscience, 2009.

[117] Saridis, G., "Intelligent robotic control," *Automatic Control, IEEE Transactions on*, vol. 28, no. 5, pp. 547–557, 1983.

[118] Scattolini, R., "Architectures for distributed and hierarchical model predictive control–a review," *Journal of Process Control*, vol. 19, no. 5, pp. 723–731, 2009.

[119] Schmidt, D., "Model-driven engineering," *IEEE computer*, vol. 39, no. 2, pp. 25–31, 2006.

[120] Schrage, D. and Vachtsevanos, G., "Software-enabled control for intelligent uavs," in *Computer Aided Control System Design, 1999. Proceedings of the 1999 IEEE International Symposium on*, pp. 528–532, 1999.

[121] Schultz, K., "Towards agile control of ship auxiliary systems," in *Resilient Control Systems (ISRCS), 2011 4th International Symposium on*, pp. 154–157, IEEE, 2011.

[122] Sciavicco, L. and Siciliano, B., *Modelling and control of robot manipulators*. Springer, 2001.

[123] Shah, H., Bahl, V., Moore, K., Flann, N., and Martin, J., "Resource allocation and supervisory control architecture for intelligent behavior generation," in *Proceedings of SPIE*, vol. 5083, p. 493, 2003.

[124] Shah, R., *Compact heat exchangers*. Taylor & Francis, 1990.

[125] Shamma, J., ed., *Cooperative Control of Distributed Multi-Agent Systems*. John Wiley & Sons Ltd, 2007.

[126] Shelton, H. H., *Joint vision 2020*. US Government Printing Office, 2000.

[127] Shou, G., Wilkins, D. C., Hoemmen, M., Mueller, C., and Tatem, P. A., "Supervisory control system for ship damage control: Volume 2-scenario generation and physical ship simulation of fire, smoke, flooding, and rupture," tech. rep., DTIC Document, 2001.

[128] Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G., *Robotics: modelling, planning and control*. Springer, 2011.

[129] Siljak, D. D., *Decentralized control of complex systems*. Academic Press (Boston), 1991.

[130] Sinha, N. K. and Gupta, M. M., *Soft computing and intelligent systems: theory and applications*. Academic Press, 1999.

[131] Smith, C. U. and Williams, L. G., *Performance solutions: a practical guide to creating responsive, scalable software*, vol. 1. Addison-Wesley Boston, MA;, 2002.

[132] Spacy, W., "Ii (2004). capability-based acquisition in the missile defense agency," *Journal of Contract Management*, pp. 10–19.

[133] Srivastava, S., Cartes, D., Maturana, F., Ferrese, F., Pekala, M., Zink, M., Meeker, R., Carnahan, D., Staron, R., Scheidt, D., and others, "A control system test bed for demonstration of distributed computational intelligence applied to reconfi guring heterogeneous systems," *Instrumentation & Measurement Magazine, IEEE*, vol. 11, no. 1, pp. 30–37, 2008.

[134] Stein, G., "Respect the unstable," *Control Systems Magazine, IEEE*, vol. 23, no. 4, pp. 12–25, 2003.

[135] Tari, Z. and Bukhres, O., *Fundamentals of distributed object systems: the CORBA perspective*, vol. 8. Wiley-Interscience, 2001.

[136] Tassey, G., "The economic impacts of inadequate infrastructure for software testing," *National Institute of Standards and Technology, RTI Project*, 2002.

[137] Tzafestas, S. G., *Methods and applications of intelligent control*. Kluwer Academic Publishers, 1997.

[138] US Air Force, "Acquisition and sustainment life cycle management," Tech. Rep. 63-101, AIR FORCE INSTRUCTION 63-101, April 2009.

[139] Veeke, H., Ottjes, J., and Lodewijks, G., *The Delft Systems Approach: Analysis and Design of Industrial Systems*. Springer Verlag, 2008.

[140] Wadlington, P. L., Sniezek, J. A., Wilkins, D. C., Tatem, P. A., and Williams, F. W., "Supervisory control system for ship damage control: Volume 6-experimental measurement of situation assessment," tech. rep., DTIC Document, 2001.

[141] Waldock, A. and Nicholson, D., "A framework for cooperative control applied to a distributed sensor network," *The Computer Journal*, vol. 54, no. 3, pp. 471–481, 2011.

[142] Walks, J. and Mearman, J., "Integrated engineering plant," *2005 Distributed Intelligence for Automated Survivability (DIAS) Program Review*, 2005.

[143] Wallace, D. R., Watson, A. H., and McCabe, T. J., "Structured testing: A testing methodology using the cyclomatic complexity metric," *NASA*, no. 19980111068, 1996.

[144] Wang, H. and Linkens, D. A., *Intelligent supervisory control: A qualitative bond graph reasoning approach*, vol. 14. World Scientific Publishing Company Incorporated, 1996.

[145] Watson, A. H., McCabe, T. J., and Wallace, D. R., "Structured testing: A testing methodology using the cyclomatic complexity metric," *NIST special Publication*, vol. 500, no. 235, pp. 1–114, 1996.

[146] Webb, M., "Capabilities-based engineering analysis (cbea)," in *Proc. of 6th International Conference on Complex Systems*, 2006.

[147] Whitten, J., Bentley, L., and Dittman, K., *Systems Analysis and Design Methods*. McGraw-Hill Higher Education, fifth edition ed., 2000.

[148] Wilkins, D. C., Schultz, K., Daniels, M., Carbonari, R., Shou, G., Spillner, B., Gimbel, K., Bulitko, V., and Williams, F. W., "The dc-scs supervisory control system for ship damage control," *Knowledge-Based Systems*, 2000.

[149] Wilkins, D. C., Sniezek, J. A., Tetem, P. A., and Williams, F. W., "Supervisory control system for ship damage control: Volume 1-design overview," tech. rep., DTIC Document, 2001.

[150] Willems, J., "The behavioral approach to open and interconnected systems," *Control Systems Magazine, IEEE*, vol. 27, pp. 46–99, Dec. 2007.