CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Autonomous system control in unknown operating conditions
Authors	Sohège, Yves
Publication date	2021-08-24
Original Citation	Sohège, Y. 2021. Autonomous system control in unknown operating conditions. PhD Thesis, University College Cork.
Type of publication	Doctoral thesis
Rights	© 2021, Yves Sohège. - https://creativecommons.org/publicdomain/zero/1.0/
Download date	2026-11-01 03:03:54
Item downloaded from	https://hdl.handle.net/10468/11901



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Autonomous System Control in Unknown Operating Conditions

Yves Sohège
BSc
112391881

Thesis submitted for the degree of
Doctor of Philosophy



NATIONAL UNIVERSITY OF IRELAND, CORK

COLLEGE OF SCIENCE, ENGINEERING AND FOOD SCIENCE

SCHOOL OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY
INSIGHT CENTRE FOR DATA ANALYTICS
LERO CENTRE FOR SOFTWARE RESEARCH

August 2021

Head of Department: Prof Cormac Sreenan

Supervisors: Prof Gregory Provan
Dr Sabin Tabirca

Contents

List of Figures . . . vii
List of Tables . . . x
Abstract . . . xii
Acknowledgements . . . xiv

I Introduction 1
1 Motivation . . . 2
2 Background . . . 5
2.1 Control for Stationary Dynamics . . . 6
2.1.1 Problem Description . . . 6
2.1.2 Model-based control . . . 8
2.1.3 Learning-based control . . . 8
2.1.4 Summary . . . 9
2.2 Control for Known Non-Stationary Dynamics . . . 9
2.2.1 Problem Description . . . 9
2.2.2 Multiple-Model Adaptive Control . . . 10
2.2.3 Switched vs Blended MMAC . . . 12
2.2.4 Model-based Supervisory control . . . 12
2.2.5 Learning-based Supervisory control . . . 13
2.2.6 Summary . . . 13
2.3 Control for Unknown Non-Stationary Dynamics . . . 13
2.3.1 Open Problem Description . . . 13
3 Approach . . . 14
3.1 Supervisory Control Problem . . . 15
3.1.1 Switching vs Blending Supervisory Control - Papers A/H . . . 15
3.1.2 Randomized Blended Control - Paper B . . . 16
3.1.3 Learning-based Supervisory Control - Papers C/D/E/F/G . . . 17
3.1.4 Summary: Supervisory Control in Unknown Operating Modes . . . 18
3.2 Low-Level Controller Tuning Problem . . . 19
3.2.1 Convex Hull-based Controller tuning - Paper I . . . 19
3.2.2 Summary: Low-Level Controller Tuning for Unknown Operating Modes . . . 20
3.3 Thesis Contributions: . . . 20
3.4 List of Impacts: . . . 21
3.5 Scope . . . 22
4 Contributions . . . 26
5 Publications . . . 31
5.1 Conference Papers . . . 31
5.2 Journal Articles . . . 32
6 Conclusion & Future Work . . . 32

II Publications **34**

A Comparing Switching vs Mixing Model-Predictive Controllers for Robust Fault-Tolerant Control **35**
1 Introduction . . . 36
2 Related Work . . . 37
3 Running Example . . . 38
4 Fault Tolerant Control Schemes . . . 40
4.1 State-Space Model . . . 40
4.1.1 Observer-Based Control . . . 41
4.1.2 Observer-Based Diagnosis . . . 41
4.2 Switching Model-Predictive Control . . . 42
4.2.1 Discrete Switching Controller . . . 42
4.2.2 Mixing Controller . . . 43
4.3 Stability Properties . . . 45
5 Experimental Design . . . 46
5.1 Software Configuration . . . 46
5.2 System Fault Injection and Detection . . . 47
5.3 Controller Design . . . 47
5.4 Discrete Switching and Merging implementation . . . 48
6 Experimental Analysis . . . 49
6.1 Switching vs. Merging . . . 49
6.2 Impact of Delays and Fault Isolation Errors . . . 50
7 Discussion . . . 51

B Fault-Tolerant Control for Unseen Faults using Randomized Methods **52**
1 Introduction . . . 53
2 Related Work . . . 54
3 Notation and Preliminaries . . . 54
3.1 System Model . . . 54
3.2 Running Example . . . 56
3.3 Performance Measures . . . 57
3.4 Blended Controller Models . . . 59
4 Theoretical Results . . . 60
4.1 Stability Properties . . . 60
4.2 Stabilizability Properties . . . 61
4.3 Performance Computation . . . 61
4.4 Performance: Known Mode Set Υ . . . 61
4.5 Performance: Unknown Mode Set . . . 62
5 Empirical Analysis . . . 62
6 Conclusions . . . 63
7 Appendix . . . 66

C Online Reinforcement Learning for Trajectory Following with Unknown Faults **68**
1 Introduction . . . 69
2 Related Work . . . 70

3 Reinforcement Learning . . . 71
4 RL Results . . . 72
5 Reinforcement Learning Blended Control . . . 73
5.1 RL States . . . 74
5.2 RL Actions . . . 75
5.3 RL Rewards . . . 75
5.4 Credit Assignment Problem . . . 76
5.5 Error Generation . . . 77
5.6 Simulation Details . . . 78
6 Conclusion . . . 78

D Unknown Fault Tolerant Control using Deep Reinforcement Learning: A blended control approach 79
1 Introduction . . . 80
2 Deep Reinforcement Learning Blended Control . . . 81
2.1 Low-Level Controllers . . . 82
2.2 High-Level Controller . . . 82
2.3 Blending Function . . . 83
3 Running Example & Related Work . . . 84
3.1 Comparison to DRLBC . . . 85
4 Deep Reinforcement Learning (DRL) . . . 86
5 Quadcopter Implementation and Training . . . 87
5.1 Low-Level Controllers . . . 88
5.2 High-Level Controller . . . 89
5.3 Training Details . . . 89
5.3.1 Rotor Fault Generation . . . 90
5.3.2 Reward function . . . 90
5.4 Training Results . . . 91
6 Experiments . . . 91
6.1 Experiment 1: Nominal Control . . . 92
6.2 Experiment 2: Rotor Faults of 50% at 5s intervals . . . 92
7 Conclusion . . . 95

E Deep Reinforcement Learning and Randomized Blending for Control under Novel Disturbances 97
1 Introduction . . . 98
2 Background & Related Work . . . 100
3 Deep Reinforcement Learning Randomized Blended Control . . . 102
3.1 Low-Level Controllers . . . 102
3.2 Performance Estimate . . . 103
3.3 Blending Function . . . 103
3.4 High-Level Controller . . . 103
3.5 Architecture comparison . . . 103
4 Quadcopters . . . 104
5 Quadcopter Implementation and Training . . . 106
5.1 Low-Level Controllers . . . 106
5.2 Performance Estimation . . . 106

5.3 Blending Function . . . 106
5.4 High-Level Controller . . . 107
5.4.1 Training Conditions . . . 107
5.4.2 Reward function . . . 107
6 Experiments . . . 108
6.1 Experimental Results . . . 109
6.1.1 Known Disturbances . . . 109
6.1.2 Unknown Disturbances . . . 109
7 Conclusion . . . 111

F Neural-Symbolic Fault Tolerant Control for Quadcopter Trajectory-Following Tasks 112
1 Introduction . . . 113
2 Approach . . . 114
2.1 Model-based FTC . . . 114
2.2 Data-Driven FTC . . . 115
2.3 Hybrid FTC . . . 116
3 Quadcopters . . . 116
3.1 Quadcopter Dynamics . . . 116
3.2 Architecture . . . 117
3.3 Comparison of Learning Tasks . . . 118
3.3.1 Model-Based . . . 119
3.3.2 Data-Driven . . . 119
3.3.3 Hybrid . . . 119
4 Hybrid Quadcopter Control . . . 120
4.1 Model-based Low-Level Control . . . 120
4.2 Learning-based High-Level Control . . . 121
4.3 Randomized Blended Control . . . 121
4.4 Training Details . . . 121
4.4.1 Fault Generation . . . 121
4.5 Data-Driven Quadcopter Controller . . . 122
5 Experiments . . . 123
5.1 Experimental Design . . . 123
5.2 Parameter Spaces . . . 124
5.3 Trajectory Following Results . . . 124
5.3.1 Hybrid Approach . . . 124
5.3.2 Data-Driven Approach . . . 125
6 Conclusion . . . 125

G A Novel Hybrid Approach for Fault-Tolerant Control of UAVs based on Robust Reinforcement Learning 126
1 Introduction . . . 127
2 Related Work . . . 128
3 Preliminaries . . . 129
3.1 Robust Markov Decision Processes . . . 129
3.2 Optimizing for the worst-case performance . . . 130
3.3 Optimizing for the average-case performance . . . 131

4 Hybrid Approach for FTC . . . 131
4.1 Model-based controllers . . . 131
4.2 Learning-based controller . . . 133
4.3 Stability Properties . . . 134
5 Quadcopter Experiments . . . 135
5.1 Operating Conditions . . . 135
5.2 Trajectory tracking task and reward definition . . . 136
5.3 Other Training Setup Details . . . 137
5.4 Results & Discussion . . . 137
6 Conclusion . . . 138

H Comparison of Control and Cooperation Frameworks for Blended Autonomy 141
1 Introduction . . . 142
2 Related Work . . . 143
3 Technical Description . . . 143
3.1 Multi-Agent Framework . . . 144
3.2 Autonomous Modes . . . 144
3.2.1 Mode Specification . . . 145
3.3 Dynamical System Model . . . 145
4 Multi-Agent Coordination . . . 146
4.1 Assumptions . . . 146
4.1.1 Reward Model . . . 146
4.1.2 Objectives . . . 146
4.2 Leader-Follower Control Approach . . . 147
4.3 Blended Control Approach . . . 148
5 Bicycle Model . . . 149
6 Empirical Experiments . . . 151
6.1 Leader-Follower . . . 151
6.2 Blending . . . 152
6.3 Oracle-based Switching . . . 152
6.4 Experiment 1: No Human input . . . 153
6.5 Experiment 2: Human on Collision Course . . . 153
6.6 Experiment 3: Delays on various signals . . . 154
6.7 Experiment 4: Noise Rejection . . . 155
7 Conclusions . . . 157

I Learning Sufficient Low-Level Controller Parameters for Blended Control in Non-Stationary Conditions 159
1 Introduction . . . 161
2 Background & Notation . . . 163
3 Task: Theoretical Framework . . . 166
3.1 Architecture . . . 166
3.2 System Description . . . 167
3.2.1 Stationary System Dynamics . . . 167
3.2.2 Non-Stationary System Dynamics . . . 168
3.3 Task Optimization for Stationary System Dynamics . . . 168

3.4 Task Optimization for Non-Stationary System Dynamics . . . 169
3.4.1 Low-level controller set optimization: a parameter coverage approach . . . 170
3.4.2 Low-level controller set optimization: a worst-case scenario approach . . . 171
4 Approach . . . 172
4.1 Particle Swarm Optimization (PSO) . . . 172
4.2 Particle Filtering (PF) . . . 173
4.3 Clustered Particle Filtering (CPF) . . . 174
5 Application Domain: Quadcopter . . . 175
5.1 Dynamics . . . 175
5.2 Control Task . . . 177
5.3 Control Architecture . . . 178
5.3.1 Learning Attitude Controller Sets . . . 179
5.4 Simulator . . . 180
6 Experiments . . . 181
6.1 Controller Parameter Sets . . . 181
6.2 Performance Comparison . . . 182
6.3 Multiple Operating Conditions . . . 185
7 Conclusion . . . 186
8 Additional Experimental Details . . . 188
8.1 Trajectory details . . . 188
8.1.1 PSO Attitude Controllers . . . 188
8.1.2 PSO Parameter Details . . . 189
8.1.3 CPF Attitude Controllers . . . 189
8.1.4 CPF Parameter Details . . . 189
9 Detailed Controller tuning results . . . 190
9.1 Experiment 2: Additional Results . . . 190

List of Figures

1.1 Simple control loop for without considering the operating environment. 5
1.2 Multiple Model Adaptive Control 10
1.3 Multiple Model Adaptive Control open problems for unknown conditions 15
1.4 Thesis Contributions visualized on MMAC architecture. 24
1.5 Thesis Overview Flowchart 25

A.1 Two tank gravity drained system. 38
A.2 Use of multiple controllers for FTC 42
A.3 Comparison of controllers for FTC with Pump fault at $t = 50$s 47
A.4 Comparison of controllers for FTC with Pump and Valve fault 49
A.5 Impact of delay in FDI on switching and merging controllers for FTC with Pump fault. 50
A.6 Impact of inaccuracy in FDI on switching and merging controllers for FTC with Pump fault. 50

B.1 System and controller framework 56
B.2 Schematic of quadcopter 56
B.3 Trajectory for quadcopter 58
B.4 $\upsilon = 0.91$ at $t = 15s$, a) Randomized fault-based Blending, b) Randomized clock-based, c)-f) Optimal blend C_R delayed by varying τ 64
B.5 $\upsilon = 0.91$ at $t = 15s$, a) Randomized fault-based Blending, b) Randomized clock-based, c)-f) Optimal blend C_R delayed by varying τ 65

C.1 Matrix learned after 250 learning cycles. Rows represent different system states and each column represents a blended controller. 73
C.2 Evaluation phase τ timings. 73
C.3 Left: Simulation Path. Right: Roll , Pitch and Yaw axis of the quadcopter. 74
C.4 Graph showing ρ (green), current error (orange) and blended controller used (blue). The subspaces $S^1, S^2, ..., S^5$ are indicated along the Y-axis using the *Level* terminology. 77
C.5 Rotor Fault magnitude ι and benchmark errors Γ throughout simulation. Note: larger errors are given longer time to stabilize. . . . 77

D.1 Deep Learning Blended Control architecture showing the 3 main parts: Low-level controllers, High-Level Controller and Blending Function. . 81
D.2 Quadcopter attitude controller outputs for trajectory tracking under rotor fault. Y-Axis represents Reference and Actual position as well as controller response in red, Aggressive (Top) and Smooth (Bottom). 83
D.3 Quadcopter Deep Reinforcement Learning Blended Control architecture. 88
D.4 Average reward obtained over 3000 episodes shown against nominal tracking performance. 91

D.5 Agent actions under rotor faults for blending Roll (Blue) and Pitch (Red). 94
D.6 Grey: Low-level controller responses and Red: Blended signal through DRL. The blended signal is able to stabilize around the set-point quicker. 94
D.7 Trajectory under rotor failure for different control architectures. Red: Switched Architecture, Blue: Blended DRL Architecture. Black: X and Y Reference. 95

E.1 The DRLRBC architecture (with two controllers). 102

F.1 Two Control architectures for quadcopters showing Data-driven control (red) and Model-based control (blue) parts. 117
F.2 Mean Cumulative Reward obtained over 5 independent training cycles of 12000 episodes. Scores below 0 indicate the quadcopter left the safe zone during the episode. 122
F.3 Experimental results showing the amount of time a quadcopter was outside of the safety bound for rotor faults of varying magnitudes. Lower is better. 123

G.1 Hybrid control architecture for fault tolerant quadcopter trajectory tracking. Low-level control is established using cascading PID control with additional roll and pitch controllers. A supervisory neural network controller estimates the probability distribution used to sample the blending weights. 132
G.2 Trajectories executed under different operating conditions when using Low-gain controllers (Blue) or High-gain controllers (Red) 133
G.3 Shows reference trajectory in black with safe zone in dashed green. When the quadcopter leaves the safe zone the agent is rewarded with a -1 while positive rewards are only possible by reaching the goal. . . . 137

H.1 Architecture for blended autonomy system: (a) shows the leader-follower approach, and (b) shows the blended approach 147
H.2 Bicycle Model of a car 149
H.3 Double Lane change reference trajectory 151
H.4 Leader-Follower framework in nominal operating conditions 152
H.5 Total tracking error for double lane change for Experiments 1,2 and 4. Blue: Leader Follower, Orange: Oracle-based Switching , Yellow: Human Dominant Blending, Purple: AS Dominant Blending 154
H.6 Total tracking error for double lane change with given delays. 154
H.7 Total tracking error for double lane change with given delays. 155
H.8 Leader Follower Total tracking error for double lane change with noise injected to u_H at t=3s. 155
H.9 Oracle-based Switching Total tracking error for double lane change with noise injected to u_H at t=3s. 156
H.10 Human Dominant Blending Total tracking error for double lane change with noise injected to u_H at t=3s. 156

H.11 AS Dominant Blending Total tracking error for double lane change with noise injected to u_{AS} at t=3s. 157

I.1 Generic hierarchical control architecture. 166
I.2 Example of Operational Volume. The red line shows a trajectory with the green surrounding area representing the operational volume associated with it. 178
I.3 Quadcopter cascading control architecture. The learning focus of this article are the roll and pitch PD controller sets. 179
I.4 Quadcopter attitude PD parameter space. Top: Coloured polygons represent the subspaces $\Theta^*_{\lambda_i}$ found through CPF for each operating condition. Diamond icons represent Θ^{CPF}_{Λ} and cross icons Θ^{PSO}_{Λ}. Bottom: Shows the convex hull coverage of the PD parameter space for each controller set. 182
I.5 Comparison average number of steps outside operational volume when using: 1) Switching using Θ^{PSO}_{Λ}, 2) Randomized Blended Control using Θ^{PSO}_{Λ} and 3) Randomized Blended Control using Θ^{CPF}_{Λ}, in various operating conditions. Each cell represents the average over 100 runs and X and Y axis define the operating condition parameters (also found in Table I.1). 184
I.6 Quadcopter attitude PD parameter space showing the convex hull coverage of the PD parameter space for each controller set and the controllers found using PSO on dual fault conditions. 185
I.7 Comparison average number of steps outside operational volume when using: 1) Switching using the extended controller set $\Theta^{PSO-ext}_{\Lambda}$ (9 controllers) , 2) Randomized Blended Control using Θ^{PSO}_{Λ} (5 controllers) and 3) Randomized Blended Control using Θ^{CPF}_{Λ} (3 controllers), in various combinations of conditions. Each cell represents the average over 100 runs and X and Y axis define the operating condition parameters. 187
I.8 Particle Swarm Optimization for quadcopter attitude PD controller gain parameters under various fault conditions. 1) Attitude Noise : 0.6rad | 2) Position Noise : 3.6m | 3) Rotor Loss of effectiveness : 20% | 4) Wind : 12m/s. 191
I.9 Clustered Particle Filtering for quadcopter attitude PD controller gain parameters under various fault conditions. 1) Nominal Conditions | 2) Attitude Noise | 3) Position Noise | 4) Rotor Loss of effectiveness | 5) Wind 192
I.10 Boxplots showing performance of each high-level architecture under random magnitude operating parameters over 500 simulations. 193

List of Tables

1.1 Part 1: Empirical comparison and results overview for each article. . 29
1.2 Part 2: Empirical comparison and results overview for each article. . . 30

B.1 Notation 54

C.1 Empirical Analysis of RLC improvment in path deviation error against the original controller. 73
C.2 $\bar{\tau}^i$ values for each partition of S after the RL simulation. 76
C.3 Low level nominal and fault controller tuning for each control axis. . 78

D.1 PID parameters for aggressive and smooth reactions to faults from a quadcopter simulation. 85
D.2 Training parameters used 90
D.3 Experiment 1: Average Tracking Accuracy under nominal conditions. . . . 92
D.4 Experiment 2: Average Tracking error under 50% rotor loss of efficiency. 93

E.1 PID parameters for low-level controller tuning used. 106
E.2 DDPG Training parameters used 108
E.3 Experimental Disturbance Details. 109
E.4 Experiment list showing Total Attitude Loss (in radians) and Total Trajectory Loss (in meters) for the compared control systems. Best performing in bold text. 110

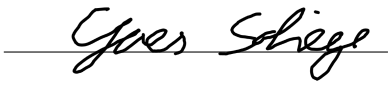
F.1 PID parameters for low-level model-based control architecture. 120

G.1 Quadcopter domain parameters and corresponding disturbance magnitudes for each level. 136
G.2 PID parameters for low-level model-based control architecture. 136

H.1 Notation used in the article 150

I.1 Quadcopter operating conditions indicated using sub-script and fault magnitudes used in experiment section using superscript. 180
I.2 Parameters used for experiment 1. 190
I.3 Quadcopter Roll and Pitch PD controller sets found using PSO and CPF approaches. Full set of environments can be found in Table I.1. . 193

I, Yves Sohège, certify that this thesis is my own work and has not been submitted for another degree at University College Cork or elsewhere.



Yves Sohège

Abstract

Autonomous systems have become an interconnected part of everyday life with the recent increases in computational power available for both onboard computers and offline data processing. The race by car manufacturers for level 5 (full) autonomy in self-driving cars is well underway and new flying taxi service startups are emerging every week, attracting billions in investments. Two main research communities, Optimal Control and Reinforcement Learning stand out in the field of autonomous systems, each with a vastly different perspective on the control problem. Controllers from the optimal control community are based on models and can be rigorously analyzed to ensure the stability of the system is maintained under certain operating conditions. Learning-based control strategies are often referred to as model-free and typically involve training a neural network to generate the required control actions through direct interactions with the system. This greatly reduces the design effort required to control complex systems. One common problem both learning- and model-based control solutions face is the dependency on *a priori* knowledge about the system and operating conditions such as possible internal component failures and external environmental disturbances. It is not possible to consider every possible operating scenario an autonomous system can encounter in the real world at design time. Models and simulators are approximations of reality and can only be created for known operating conditions. Autonomous system control in unknown operating conditions, where no *a priori* knowledge exists, is still an open problem for both communities and no control methods currently exist for such situations.

Multiple model adaptive control is a modular control framework that divides the control problem into supervisory and low-level control, which allows for the combination of existing learning- and model-based control methods to overcome the disadvantages of using only one of these. The contributions of this thesis consist of five novel supervisory control architectures, which have been empirically shown to improve a system's robustness to unknown operating conditions, and a novel low-level controller tuning algorithm that can reduce the number of required controllers compared to traditional tuning approaches. The presented methods apply to any autonomous system that can be controlled using model-based controllers and can be integrated alongside existing fault-tolerant control systems to improve robustness to unknown operating conditions. This impacts autonomous system designers by providing novel control mechanisms to improve a system's robustness to unknown operating conditions.

For a better world...

Acknowledgements

Throughout my studies at University College Cork, there have been so many people that have helped me in countless ways and I cannot thank all of them one by one, but I will try to express my everlasting gratitude to them as much as I can.

Firstly, I would like to thank my supervisor and mentor, Prof. Gregory Provan, whose continued guidance over the past four years has made this entire thesis possible. At every opportunity, I was encouraged to take charge of my research development, which offered me the freedom to travel, network, and develop myself in a way I could not have imagined possible at the start of this journey. For this, and so many other things, I will always be grateful. Dr. Marcos Quiñones-Grueiro also deserves a special mention as my informal co-supervisor, as I would not have been able to navigate through the complex research world without such an excellent supervisory team to guide me along the way. I am immensely thankful for having had the opportunity to work with both of you these past few years and hope our collaborations extend far into the future.

I would like to express my unbounded gratitude to my family and close friends, who have supported me throughout my entire life. I never dreamed I would be in the position I find myself in today and this has only been possible because of every little moment that I was allowed to share with each of you along the way. These moments allow me to look back at this period of my life with joy, happiness, and love, forever.

My work colleagues, fellow students, mentors, and the support staff in University College Cork, specifically the Insight and Lero Centres, created an incredibly constructive and inspiring environment for me over the past eight years of my studies. I am extremely grateful for all of the support and assistance you provided daily.

I am especially grateful to my mother, father, brother, and a few specific people who deserve to be mentioned. These include Brian, Kamil, Aaron, Momo, Mimo, Mesut, Marie, Milan, Tadhg, Andrea, Diego, Federico, Cathal, Chrys, Happy, Ben, Marcos, Camila, Patrick, May, Adam, Emilie, Igor, Asia, Fenics and Snow.

As a final acknowledgement to all who came before me, no better line comes to mind than the eternal quote:
"If I have seen further it is by standing on the shoulders of Giants"
— Isaac Newton, 1675

Part I

Introduction

1 Motivation

Autonomous Systems cover a vast range of application domains including self-driving cars [1], unmanned aerial vehicles (UAVs) [2], submersibles [3], humanoid-robots [4], spacecrafts [5], manufacturing and chemical plants [6], and many more. The problem of autonomous system control originated in the 1950s and comes from the field of sequential decision problems [7].

Sequential decision problems [8] are generally defined as optimization problems and cover a huge range of problems. The problems can be divided into various classes, with the *control problem* being one class of problems. Due to the many variations of sequential decision problems over 15 different research communities have developed to address them, each with their own vocabulary, solution methods, and is supported by at least one book and over 1000 research papers [9]. The two communities that have made the most progress on the control problem are the *Optimal Control* [10] and *Reinforcement Learning* [11] communities, each with very different (some argue polar opposite [9]) perspectives and approaches.

The optimal control community enjoys a rich history and has developed mathematical frameworks around model-based design and analysis. Model-based design is centered around understanding the dynamics of a system and developing a model of the underlying physics. Several control mechanisms, such as Proportional Integral Derivative (PID) [12] and Model Predictive Controllers (MPC) [13] can be used once a model of the system is available. The optimization problem is typically framed around choosing the actions that will minimize a *loss function* over time. Stability has been a central focus in the control community since its inception and analysis tools, such as Lyapunov functions [14], allow system designers to guarantee the system is controllable under certain conditions. Nearly all autonomous systems deployed in the real-world are run with model-based controllers [12].

Reinforcement Learning is based on the Markov Decision Process [15] and originated more recently in the 1990s [11]. The field has made incredible progress in the last decade due to the increase in computational power available to train reinforcement learning agents. The optimization function is typically defined as a *reward function*, where the goal is to take actions that maximize the expected reward obtained over time [1]. Neural networks and model-based simulators have been used to learn control of complex systems, such as UAVs [16]. However, the training process typically takes millions of iterations, which is not feasible for expensive systems in the real world that can be crashed. Training is mostly restricted to simulated environments.

Both communities face several problems with the control of real-world systems. Model-based design involves approximating the real-world dynamics, and regardless of how accurate the model is it will always remain an approximation [17]. Learning in simulation suffers the same limitations as it is also based on a model of the system. This problem is known as the simulation-to-reality (*sim2real* [18]) problem. Real-world environments are also inherently non-stationary and it is not possible to consider every possible failure or disturbance a system can experience throughout its operational lifetime at design time. To further complicate the problem, many systems, such as flying vehicles, are based on real-time control, meaning immediate actions need to be taken to stabilize the system in case of failures. This limits many of the current control approaches, which require a significant amount of data to confidently take a corrective action [19].

The problem of controlling a system in known non-stationary environments has resulted in the emergence of many sub-communities in both the learning and control community. The learning community addresses this problem in the Meta-Reinforcement Learning [20] and Robust Reinforcement Learning [21] communities, while Fault-Tolerant Control [19], Adaptive Control [22], and Robust Control [23] emerged in the control community. Interestingly, there are many overlaps in the general approaches of both communities to overcome the non-stationary dynamics, which are either based on the identification of the non-stationary changes (active) or creating robustness to changes without identification at design time (passive) [24, 19].

Robustness is generally defined as the ability to operate under bounded parameter uncertainty [25]. For example, the weight of a UAV might change for each flight depending on varying payloads, or the number of passengers and GPS position data might be noisy. This is different from actual failure situations such as total rotor failures, which change the dynamics of the UAV and hence require the control system to adapt the way the vehicle is controlled. The robust and fault-tolerant control communities are closely related since both of these aspects are necessary for real-world systems and this overlap is typically called Robust Fault-Tolerant Control. Similarly, the Robust and Meta Reinforcement Learning communities also mirror this relationship in creating robustness to parameter uncertainty and adapting to new situations.

In summary, each of the mentioned communities is approaching parts of the same problem with vastly different perspectives and mechanisms. If *a priori* information about the system and environment is available both learning-based and model-based controllers are feasible. There is also extensive overlap in the form of hybrid

approaches that use mechanisms from several communities [26].

The real-time control problem in unknown non-stationary operating conditions is still not solved by either community, as both are highly dependent on *a priori* information about the system and environment, which by definition is not available at design time. This is a major problem for the large-scale integration of autonomous systems in the real world. This thesis focuses on the intersection between learning and control and utilizes existing methods developed across the above-mentioned communities. The approach is based on a well-known, modular control framework for non-stationary systems developed in the adaptive control community, called Multiple Model Adaptive Control [27]. This framework divides the control task into a supervisory and low-level control problem, which allows for the integration of both model-based and learning-based controllers. Model-based controllers have significant advantages for the low-level control problem in the real world due to the stability guarantees provided by the control community. Learning-based controllers provide a flexible supervisory controller that can overcome the limitations of a pre-designed model-based supervisory controller, such as fault identification delays and manual design efforts.

This research makes two main contributions. The first is a set of five novel supervisory control methods that have empirically been shown to improve a system's robustness to unknown operating conditions. These consist of one pure model-based and four hybrid model/learning-based control methods. The second is a novel low-level controller tuning algorithm that automatically finds a sufficient controller set to cover all known operating modes. The contributions are presented across nine articles. The hybrid design of the presented methods drastically reduces the design effort required to create a robust control system as learning is mostly automated. All learning-based supervisory controllers presented in this thesis maintain the stability properties of the model-based low-level controllers during training.

The contributions have the following impacts. 1) They provide control system designers with novel control mechanisms to improve a system's robustness to unknown operating conditions. Currently, no control methods exist for such situations. 2) The presented control methods can be integrated alongside existing fault-tolerant control systems and are applicable for any autonomous system, given that a model of the system exists. 3) The approach explored offers a new perspective on the combination of learning- and model-based controllers and allows for future developments from both fields, such as better model-based controllers or faster training algorithms, to be incorporated without modifications.

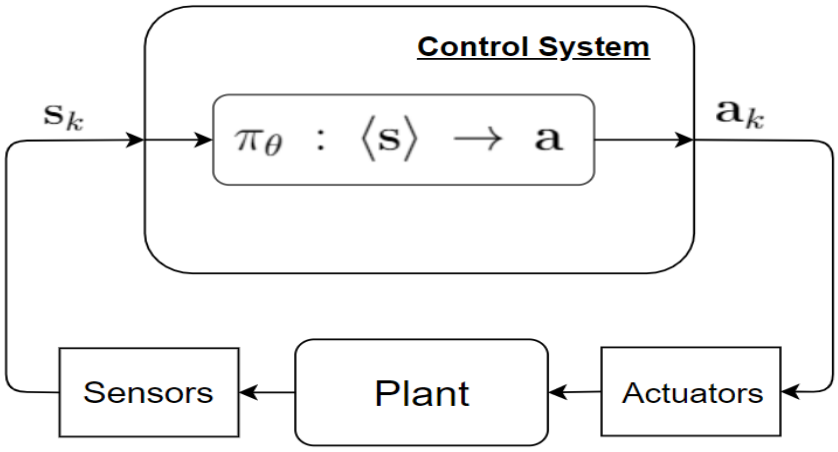


Figure 1.1: Simple control loop for without considering the operating environment.

2 Background

The control problem is presented in three parts of increasing difficulty, firstly stationary system dynamics, followed by known non-stationary system dynamics, and finally unknown non-stationary system dynamics. A notation and acronym overview can be found in the table below. A simple control loop can be seen in Figure 1.1.

The behavior of a system is characterized using the tuple $\langle \mathcal{S}, \mathbf{\Lambda}, \mathcal{A} \rangle$ and k as a temporal index. The state of the system is described by a state vector $\mathbf{s}_k \in \mathcal{S}$ and the control actions issued to the plant by action vector $\mathbf{a}_k \in \mathcal{A}$. A system can operate in a set of modes $\lambda_k \in \mathbf{\Lambda}$, which characterize the operating conditions, e.g., plant fault conditions or adverse environmental conditions. A controller/policy is a function that maps the system states to the corresponding control actions and is defined using a parameter vector θ. The set of modes is defined using a parameter set $\mathbf{\Lambda} \in \Re^n$, with n being the number of parameters. External disturbances to the system and component failures are defined in subsets $\mathbf{E}$ and $\mathbf{F}$, such that $\mathbf{\Lambda} = \mathbf{E} \cup \mathbf{F}$.

PID Proportional Integral Derivative Controller
MPC Model-Predictive Controller
LQR Linear Quadratic Regulator
UAV Unmanned Aerial Vehicle
MMAC Multiple Model Adaptive Control
FDI Fault-Detection and Isolation
DDPG Deep Deterministic Policy Gradient
TRPO Trust Region Policy Optimization
PPO Proximal Policy Optimization

$\mathcal{S}$ State space of the system
$\mathbf{s}_k$ State vector at time k
$\mathcal{A}$ Control action space
$\mathbf{a}_k^i$ Control action vector issued by controller i at time k
$\mathbf{E}$ Set of operating modes defined by known environment parameters
$\mathbf{F}$ Set of operating modes defined by known fault parameters
$\mathbf{U}$ Set of unknown modes
$\mathbf{\Lambda}$ Set of all known operating modes
λ_k Operating mode vector at time k
$\mathcal{T}$ Control Task
$\mathcal{L}$ Loss function of a task
Ψ Stochastic transition function
Σ^λ Plant Dynamics parameterized by operating mode λ
$\mathcal{K}$ Duration of a task
π_θ Low-level system controller parameterized by θ
θ Parameters defining low-level controller π
Θ Set of controller parameters
Π_Θ Set of low-level controllers parameterized by Θ
$\mathcal{C}(\Theta)$ Convex hull coverage of parameter space by set Θ
$\varphi_\mathbf{k}$ Controller weight vector
ω Supervisory controller parameter
Φ_ω Supervisory control law

2.1 Control for Stationary Dynamics

The problem is formulated using a mix of notation from both learning and control community. The notation in the individual articles accompanying this thesis may be different.

2.1.1 Problem Description

Stationary dynamics refer to the system operating only in nominal plant states and the environment does not change. In this situation, the dynamics can be defined using the function $\Sigma : \mathcal{S} \times \Lambda \times \mathcal{A} \rightarrow \mathcal{S}$. Here, $\mathbf{\Lambda}$ is fixed for any task, i.e., the environment parameters are known *a priori* and that do not change during a task, and faults do not

occur. The objective of the system is to complete a task $\mathcal{T}$ defined by the tuple

$$\mathcal{T} = (\mathcal{L}, \Psi, \mathcal{K}) \tag{1.1}$$

where

- $\mathcal{L}$ is a loss function
$$\mathcal{L} : \langle \mathbf{s}, \lambda, \mathbf{a} \rangle \rightarrow [0, 1] \tag{1.2}$$
that assigns a [0,1] loss at time k;
- $\Psi : \mathcal{S} \times \Lambda \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a stochastic transition function that estimates the probability of reaching state $\mathbf{s}'$ at $k + 1$ when action $\mathbf{a}_k \in \mathcal{A}$ is taken by the control system ($\Psi = P(\mathbf{s}_{k+1} = \mathbf{s}' | \mathbf{s}_k, \lambda, \mathbf{a}_k)$);
- $\mathcal{K}$ corresponds to the duration of the task.

The objective of solving a control task is thus to minimize the loss $\mathcal{L}$ in solving task $\mathcal{T}$ over $\mathcal{K}$ time-points. Task optimization can be defined as:

Definition 1 (Task Optimization)**.** *Given a task $\mathcal{T}$, select a sequence of control actions $\mathcal{A}_{1:\mathcal{K}}$ that drive the system through a state/environment sequence $\mathcal{S}_{1:\mathcal{K}}, \mathbf{\Lambda}_{1:\mathcal{K}}$ to minimize*

$$\sum_{k=1}^{\mathcal{K}} \mathcal{L}(\mathbf{s}_k, \lambda, \mathbf{a}_k), \tag{1.3}$$

where $\mathbf{s}_k \in S_{1:\mathcal{K}},\ \lambda \in \Lambda_{1:\mathcal{K}}, \mathbf{a}_k \in \mathcal{A}_{1:\mathcal{K}}$.

Note that the mode λ is fixed in the interval $1 : \mathcal{K}$. We assume that there exists an optimal control action $\mathbf{a}^*$ at every time step, i.e.,

$$\exists\, \mathbf{a}_k^* = \arg\min_{\mathbf{a}_k \in \mathcal{A}} \mathcal{L}(\mathbf{s}_k, \lambda, \mathbf{a}_k). \tag{1.4}$$

Given a controller $\pi_\theta : \langle \mathbf{s} \rangle \rightarrow \mathbf{a}$ characterized by the parameter vector θ, the optimization problem to solve a task for stationary system dynamics is the following

$$\theta^* = \arg\min_{\theta \in \Theta} \sum_{k=1}^{\mathcal{K}} \mathcal{L}(\mathbf{s}_k, \lambda, \pi_\theta(\mathbf{s}_k)) \tag{1.5}$$

Problem 1.5 can be solved by both learning-based and model-based controllers if the following conditions are satisfied for task $\mathcal{T}$

A. $\mathcal{L}(\mathbf{s}, \lambda, \mathbf{a}) \leq C < \infty\ ,\ \forall \mathbf{s} \in \mathcal{S}, \lambda \in \Lambda, \mathbf{a} \in \mathcal{A}$, and some bound C.

B. Ψ and $\mathcal{L}$ do not change with time.

2.1.2 Model-based control

The control community has developed many generic controllers to solve Problem 1.5, such as Proportional Integral Derivative (PID), Model-Predictive Control (MPC), and Linear Quadratic Regulator (LQR) being among the most popular [19]. Nearly all real-world control systems consist of model-based controllers, with the most common being the PID controller due to its simplicity and effectiveness [12].

Model-based controllers require, by definition, a model of the system so the controller parameters can be tuned to give the required action. Obtaining an accurate model can be extremely challenging as all models are approximations of the real-world and making these models more accurate is a core focus of the control community. Once a model is obtained, a vast number of controllers and automated tuning methods become available to solve equation 1.5.

Stability is a central focus in the control community and mathematical analysis tools, such as Lyapunov functions, have been an integral part of the design of control systems for several decades. System stability can typically be ensured up to a certain parameter threshold, after which control is no longer possible [19]. For example, a UAV controller may be designed for wind speeds up to 10km/h, anything above will result in the loss of vehicle control [28]. The importance of stability guarantees can not be overstated when operating in a real-world scenario, where failure can result in catastrophic consequences such as the loss of human lives. This is one of the reasons why real-world control systems are dominated by model-based controllers.

2.1.3 Learning-based control

The most popular controller implementation for learning-based control is by far the neural network, specifically a deep network (deep reinforcement learning) [29]. In this case, θ^* represents the optimal neural network weight assignment that will map the system states to the correct actions and is obtained by training the network on a task offline.

Some important focuses of the learning community are **performance** and **faster learning**. A staggering amount of new training algorithms are published every year, such as Deep Deterministic Policy Gradient (DDPG) [30], Trust-Region Policy Optimization (TRPO)[31], Proximal Policy Optimization (PPO)[32], etc. These algorithms are typically designed around specific types of problems, such as discrete vs

continuous state/action spaces, on/off-policy learning, various internal configurations to structure the learning process (e.g. actor/critic), and how the neural networks updates are performed. The solution to a discrete reinforcement learning problem is Dynamic Programming to achieve the Bellman optimality criteria [11].

While all of these algorithms try to maximize the amount of information obtained from each action taken, it is not possible to ensure these actions are always safe. Training an agent in a real-world setting, where a bad action can have serious consequences, is not feasible with current approaches. Training is mostly done in simulation over millions of iterations on a task. This leads to the same constraints as model-based controllers, a model of the system (simulator) is needed to learn a control policy.

In contrast to model-based controllers, there is no way to guarantee how a neural network-based controller will perform in a real-world setting, as the training simulation is only an approximation of reality [33]. This problem is widely referred to as the sim2real transfer problem and has been one of the main hurdles in deploying learning-based controllers in the real world [18].

2.1.4 Summary

Both model-based and learning-based control approaches require a model of the system to tune/train a controller/policy. Model-based controllers have a clear advantage as these controllers can be analyzed to fully understand how they will perform under varying conditions. Learning-based controllers can only be trained in simulation, also requiring a model of the system, but are currently lacking any stability guarantees for the safe application in the real world.

2.2 Control for Known Non-Stationary Dynamics

2.2.1 Problem Description

Non-stationary dynamics refer to changes in the plant or environment and can be characterized using mode transitions and are modelled as hybrid mode changes. The dynamics can be defined using the function $\Sigma^{\lambda} : \mathcal{S} \times \mathbf{\Lambda} \times \mathcal{A} \rightarrow \mathcal{S}$. A task for non-stationary system dynamics is defined by the tuple,

$$\mathcal{T}^{\lambda} = (\mathcal{L}, \Psi^{\lambda}, \mathcal{K}) \tag{1.6}$$

, where the behavior of the system depends on the parameter configuration vector $\lambda \subset \mathbf{\Lambda}$ such that $\Psi^{\lambda} = P(\mathbf{s}_{k+1} = \mathbf{s}' | \mathbf{s}_k, \lambda_k, \mathbf{a}_k)$. For example, the dynamics

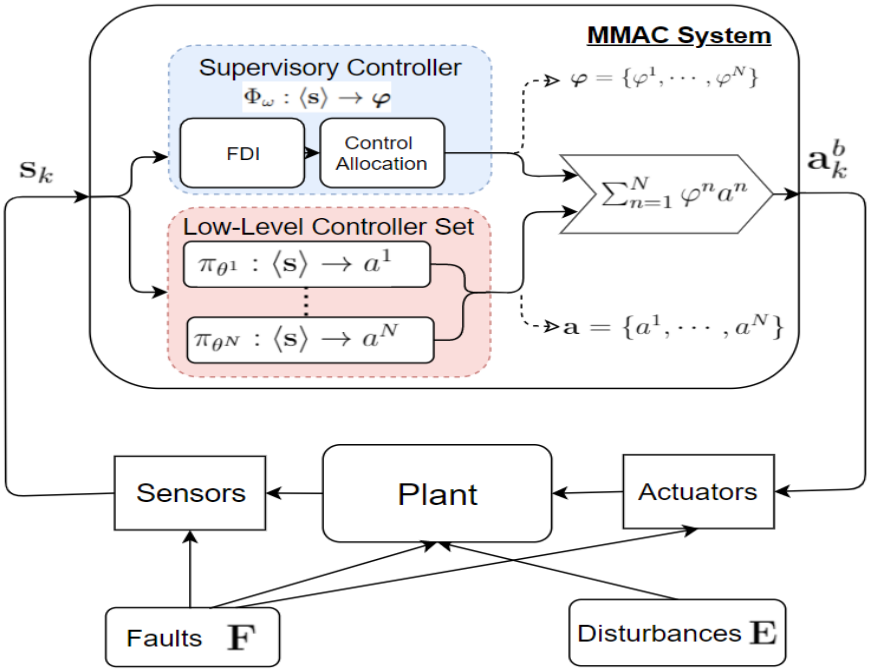


Figure 1.2: Multiple Model Adaptive Control

of a flying vehicle experiencing a rotor fault or wind conditions are different from nominal conditions. Therefore, the parameter vector λ encodes different system and environmental conditions.

Similar to stationary dynamics, a task for non-stationary system dynamics involves minimizing the loss function. However, a single controller cannot solve such a task because the second condition of Problem 1.5 is not satisfied (Ψ may change with time depending on the operating mode).

2.2.2 Multiple-Model Adaptive Control

Multiple-Model Adaptive Control (MMAC) is a modular control framework developed in the 1970s [34], based on the 'divide and conquer' strategy, to deal with non-stationary operating conditions. The control task is split into **supervisory control** (high-level) and **low-level control**, which can be seen in Figure 1.2. A set of low-level controllers $\Pi = \{\pi_{\theta^1}, \cdots, \pi_{\theta^N}\}$ is used to overcome the non-stationary nature of the transition function Ψ^λ. The low-level controllers run in parallel generating an action vector $\boldsymbol{a}_k = \{a_k^1, \cdots, a_k^N\}$ at every time step. The supervisory control task is to identify λ_k and generate a weight vector $\boldsymbol{\varphi}_k = \{\varphi_k^1, \cdots, \varphi_k^N\}$ to convexly combine the low-level control actions $\boldsymbol{a}_k = \{a_k^1, \cdots, a_k^N\}$. The supervisory controller can be defined as $\Phi_\omega : \langle \mathbf{s} \rangle \rightarrow \boldsymbol{\varphi}$, under the constraints that $\varphi^i \in [0, 1]$ and $\sum_i \varphi^i = 1$.

Definition 2 (Multiple Model Adaptive Control)**.** *Given a low-level controller set* Π

that generates an action vector $\boldsymbol{a} = \{a^1, \cdots, a^N\}$*, and a corresponding weight vector* $\boldsymbol{\varphi} = \{\varphi^1, \cdots, \varphi^N\}$ *generated by a supervisory controller, multiple model adaptive control applies the weighted action* $\boldsymbol{a}^b = \sum_{i=1}^{N} \varphi^i a^i$*, such that: (1)* $\forall_i, a^i \in \boldsymbol{a}, \varphi^i \in [0, 1]$*, and (2)* $\sum_i \varphi^i = 1$.

The loss function for a multiple model adaptive control system can be rewritten in terms of a weighted action vector,

$$\min \sum_{k=1}^{\mathcal{K}} \mathcal{L}(\mathbf{s}_k, \lambda_k, \mathbf{a}_k \boldsymbol{\varphi}_k), \tag{1.7}$$

where $\mathbf{s}_k \in S_{1:\mathcal{K}},\ \lambda_k \in \Lambda_{1:\mathcal{K}},\ \mathbf{a}_k \in \mathcal{A}_{1:\mathcal{K}},$ and $\boldsymbol{\varphi}_k \in [0, 1]_{1:\mathcal{K}}$.

The actions applied to the system are dependent on two parts, weight vector $\boldsymbol{\varphi}_k$ and action vector $\mathbf{a}_k$. The optimization problem to complete the task can also be split into two parts, supervisory controller optimization and low-level controller set optimization.

Problem 1 (Low-level Controller Set Optimization)**.** *Given a set of known operating modes* $\Lambda = \{\lambda^1, \cdots, \lambda^N\}$*, find a low-level controller parameter set* $\Theta^* = \{\theta^1, \cdots, \theta^N\}$ *such that*

$$\Theta^* = \arg\min_{\theta^i \in \Theta} \sum_{k=1}^{\mathcal{K}} \mathcal{L}(\mathbf{s}_k, \lambda, \pi_{\theta^i}(\mathbf{s}_k)),\ \forall\, \lambda^i \in \Lambda \tag{1.8}$$

In other words, the goal is to optimize the low-level controller set such that all known operating modes are covered by some controller in the set. As the number of conditions considered increases this becomes increasingly complex and is not feasible for every possible condition. This problem can only be solved if $\boldsymbol{\varphi}$ is fixed during the low-level controller optimization process, hence it is omitted here. The operating modes in Λ are treated separately and are assumed not to occur simultaneously. Solving this optimization problem results in a low-level controller set Π_{Θ^*}, which contains a controller for each mode in Λ.

The supervisory control problem cannot be solved unless a low-level controller set exists, as the supervisory controller combines the actions generated by the controllers.

Problem 2 (Supervisory Controller Optimization)**.** *Given a set of known operating modes* $\Lambda = \{\lambda^1, \cdots, \lambda^N\}$ *and a low-level controller set* Π_{Θ^*} *(which satisfies equation 1.8) that generates an action vector* $\mathbf{a}_k$*, find a supervisory controller parameter vector*

$\omega^* \in \Omega$ *such that,*

$$\omega^* = \arg\min_{\omega \in \Omega} \sum_{k=1}^{\mathcal{K}} \mathcal{L}(\mathbf{s}_k, \lambda_k, \mathbf{a}_k \Phi_{\omega^*}(\mathbf{s}_k)), \quad \forall\, \lambda^i \in \Lambda \tag{1.9}$$

In other words, the supervisory controller should be configured such that the weight vector $\boldsymbol{\varphi}_k$ applies the loss minimal action from the action vector $\boldsymbol{a}_k$ to the system.

2.2.3 Switched vs Blended MMAC

There are two main types of supervisory controllers, Switched [35] and Blended (Mixed) [27]. Both have the same convex combination constraint, i.e. $\sum_i \varphi^i = 1$, but they differ in the parameter range allowed for φ. Most systems utilize a switched supervisory controller where each weight takes a discrete value of 0 or 1, and given the constraint, this only allows a single controller full control at any time. This makes perfect sense when the best performing low-level controller is known.

Blended Control allows controller weights from a continuous parameter range, such that $0 \leq \varphi \leq 1$. In most cases, this is used for partial fault-tolerant control [2] or in situations where the entire dynamics of the vehicle change, such as a tilt-rotor aircraft [36]. Blended control is a more complicated supervisory controller than simply switching due to the increased parameter range of φ [37].

2.2.4 Model-based Supervisory control

Model-based design of supervisory controllers typically revolves around fault-detection and isolation (FDI) using a set of system monitors, such as Kalman Filters (and the many variations). Each monitor is tuned for a specific operating mode in Λ, which creates a one-to-one mapping of system monitors to low-level controllers. This is one of the main reasons switching-based supervisory controllers are so common [38].

This approach has two fundamental drawbacks, which are the need for *a priori* knowledge of the operating mode to tune the filter and an inherent delay in the mode estimation computation. The delay in model-based FDI mechanisms is a big problem for real-time control systems, such as UAVs. A supervisory controller based on model-based identification can only work for known operating conditions considered at design time, as a system monitor needs to be tuned for each mode.

2.2.5 Learning-based Supervisory control

Neural networks are powerful function approximators and can be used to implement the same functionality as the set of system monitors. Training a neural network to generate the weight vector φ has mostly been explored for switched control as it overcomes the inherent FDI delays experienced by model-based controllers [39]. Learning-based supervisory controllers are also easier to tune as training is mostly automated, which overcomes the manual tuning of system monitors required by model-based methods.

Currently, no real-time training algorithms exist and most learning-based supervisory controllers are trained offline. There is significant progress in the online- and meta-learning community [40]. Meta reinforcement-learning is working towards controllers that can adapt to novel task/operating conditions with a small amount of data. These methods currently fail when the novel task is too different from the set of training tasks but have developed significantly in recent years [20]. However, meta-learning is typically applied to learn full system control and has not been used to train supervisory controllers so far.

2.2.6 Summary

Model-based supervisory controllers typically involve pre-defining system monitors for each $\lambda \in \Lambda$ and switching control to the corresponding low-level controller. Learning-based controllers can implement the same functionality as a set of system monitors but with significantly less tuning effort and no fault-detection and isolation delays. Overall, both approaches are feasible for autonomous system control in known operating conditions.

2.3 Control for Unknown Non-Stationary Dynamics

2.3.1 Open Problem Description

It is impossible to know all faults and adverse environments a system can experience in its operational lifetime *a priori*. This means there will always be an unknown set of modes $\mathbf{U}$ not available at design time. The only difference to the known non-stationary dynamics problem is that the operating mode can be known or unknown, $\lambda \in \mathbf{\Lambda} \cup \mathbf{U}$. The dynamics of such a system can be defined using the function $\Sigma^{\lambda} : \mathcal{S} \times \mathbf{\Lambda} \times \mathbf{U} \times \mathcal{A} \rightarrow \mathcal{S}$. The loss function and task definition do not change besides $\lambda \in \mathbf{\Lambda} \cup \mathbf{U}$. Both Problem 1 and 2 can be extended to $\mathbf{U}$ but cannot be solved by current learning- or model-based controllers.

Open Problem 1 (Low-level Controller Set Optimization)**.** *Assuming there exists an unknown set of operating modes* $\mathbf{U}$ *and given a set of known operating modes* $\Lambda = \{\lambda^1, \cdots, \lambda^N\}$*, find a low-level controller parameter set* $\Theta^* = \{\theta^1, \cdots, \theta^N\}$ *such that*

$$\Theta^* = \arg\min_{\theta^i \in \Theta} \sum_{k=1}^{\mathcal{K}} \mathcal{L}(\mathbf{s}_k, \lambda_k, \pi_{\theta^i}(\mathbf{s}_k)),\ \forall\ \lambda^i \in \Lambda \cup \mathbf{U} \tag{1.10}$$

Since no *a priori* information about the set $\mathbf{U}$ is available at design time, it is not possible to solve open problem 1. The supervisory optimization problem can be similarly extended, if we assume open problem 1 is solved but suffers the same problem.

Open Problem 2 (Supervisory Controller Optimization)**.** *Assume there exists an unknown set of operating modes* $\mathbf{U}$*. Given a set of known operating modes* $\Lambda = \{\lambda^1, \cdots, \lambda^N\}$ *and a low-level controller set* Π_{Θ^*} *(assumed to satisfy equation 1.10) that generates an action vector* $\mathbf{a}_k$*, find a supervisory controller parameter vector* ω^* *such that,*

$$\omega^* = \arg\min_{\omega \in \Omega} \sum_{k=1}^{\mathcal{K}} \mathcal{L}(\mathbf{s}_k, \lambda_k, \mathbf{a}_k \Phi_{\omega^*}(\mathbf{s}_k)),\ \ \forall\ \lambda^i \in \Lambda \cup \mathbf{U} \tag{1.11}$$

The fundamental problem that both learning-based and model-based controllers face is the dependency on *a priori* knowledge of the operating modes Λ. Currently, no control methods exist for situations when $\lambda \in \mathbf{U}$.

The main goal is to investigate if a learning-based supervisory controller can be trained on known operating conditions Λ to improve the robustness of a system to unknown operating modes $\mathbf{U}$. The presented supervisory control methods use a blended supervisory controller because of the increased parameter range of φ. This gives a supervisory controller more options to combine the action vector $\mathbf{a}_k$ for unknown operating conditions.

3 Approach

Open Problems 1 and 2 are interdependent and usually, the low level controller optimization problem must be solved before the supervisory control problem becomes feasible. However, when a system encounters an unknown operating mode in a real-world situation, some controller set Π must be present. If we assume that a suitable control action $\mathbf{a}^*$ for $\lambda \in \mathbf{U}$ can be interpolated using some optimal combination φ^* of the actions generated by the existing controller set Π, then Open Problem 2 can be

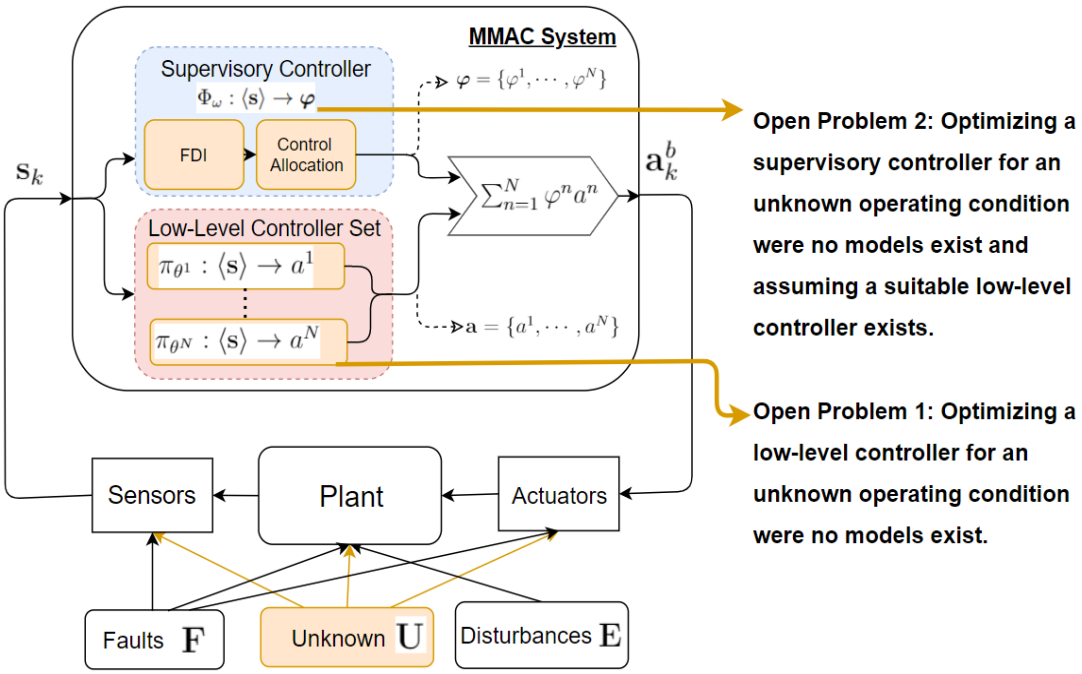


Figure 1.3: Multiple Model Adaptive Control open problems for unknown conditions

addressed. This thesis presents a set of novel supervisory control architectures that can be used to improve the robustness of a system to unknown operating modes and a novel low-level controller tuning algorithm that can reduce the size of the controller set required compared to traditional approaches. The relevant paper under discussion in each subsection is indicated and further discussion and results can be found in Part 2.

3.1 Supervisory Control Problem

Open Problem 2 is approached under the assumption that a suitable control action $\mathbf{a}^*$ exists for $\lambda \in \mathbf{U}$.

3.1.1 Switching vs Blending Supervisory Control - Papers A/H

A comparative study between switching and blending model-based supervisory controllers is presented in two articles on different application domains, a simple water tank system, and a car steering trajectory tracking task.

In summary, a blended controller is less sensitive to FDI delays and false-positive identification. The speed and accuracy of FDI are found to be critical to the performance of switched supervisory controllers. The results show that sharing control between several controllers using blended control can reduce the impact of badly tuned

low-level controllers.

Further, for the case of unknown operating environments, a switched control framework is restricted to the low-level controllers defined at design time as no interpolation is possible. For the remainder of the thesis, the supervisory controllers investigated are based on a blended controller due to the ability to interpolate between control actions.

3.1.2 Randomized Blended Control - Paper B

The first contribution of this thesis is an alternative supervisory control methodology based on randomization instead of identification when $\lambda \in \mathbf{U}$. The idea behind randomizing the supervisory control actions is that in an unknown operating environment, where no *a priori* information exists, every low-level controller has an equal chance of being the one that can stabilize the system. A continuous uniform probability distribution Δ, defined over interval $[\alpha, \beta]$, is used to sample the blending weight vector, $\boldsymbol{\varphi}_k \sim \Delta(\alpha, \beta)$, where $[\alpha, \beta] = [0, 1]$ and $\sum_i \varphi^i = 1$. Note $[\alpha, \beta]$ is fixed in this article. The key notion that makes this approach work is that randomization is computationally cheap and allows the weights to be re-sampled at every iteration of the control loop. By using a uniform probability distribution to sample the blending weights, control is essentially distributed uniformly over the entire low-level controller set.

In the event of an unknown operating condition, this approach has the advantage of not relying on any single controller at any time, as is the case with identification-based supervisory controllers. Given that the operating condition can't be identified in real-time, the single controller an identification-based approach uses is essentially chosen randomly. The phrase *'Don't put your eggs in one basket'* seems appropriate to describe the idea behind fast randomized blended control when $\lambda \in \mathbf{U}$. One clear drawback of this approach is that it is sub-optimal because of the fixed sampling interval $[\alpha, \beta]$ and the uniform distribution used.

Randomized Blended Control was empirically compared against a Switched Controller with optimal fault-detection and isolation on a quadcopter trajectory tracking task under rotor loss of effectiveness. The switched controller contains *a priori* knowledge of the optimal $\varphi*$ for the unknown operating mode, making it an unrealistic benchmark. We introduce FDI delays of varying lengths to compare how randomization performs against optimal but time-delayed identification. The quadcopter simulation uses a cascading PID controller architecture for position and attitude control. Fault tolerance is focused on the attitude control system using a set of

PID controllers for roll and pitch control and both supervisory controllers use the same low-level controllers.

The results show that randomized blended control outperforms the benchmark switched controller, if the FDI delay is larger than 0.5 seconds, the smallest time delay investigated. This is significant because unknown operating modes cannot be identified, which means randomized blended control can offer an alternative and effective supervisory control approach for unknown operating modes. A major advantage of randomized blended control is that no *a priori* knowledge is needed.

3.1.3 Learning-based Supervisory Control - Papers C/D/E/F/G

The next contribution is a set of four novel learning-based supervisory controllers that are empirically shown to improve robustness to unknown operating conditions. The contribution is spread over five articles, all focused around quadcopter trajectory tracking under various faults and disturbances. Three articles (Paper D, E, and G) are centered around offline training on Λ. One article (Paper C) represents a framework to check the feasibility of learning blended control online when $\lambda \in \mathbf{U}$. The final article (Paper F) presents the advantages of applying learning in a hybrid control framework as utilizing model-based controllers drastically simplifies the control task compared to learning direct control of the system.

Paper D: The first supervisory control approach is based on learning to directly generate the blending weights for the controller set offline and is referred to as Deep Reinforcement Learning Blended Control (DRLBC). This approach represents an identification-based method as no randomization is involved in the control loop. A neural network is used to directly map states to controller weights, $\mathbf{s}_k \rightarrow \boldsymbol{\varphi}_k \in [0, 1]$. A blending-based neural network as a supervisory controller overcomes the drawbacks of both switched and model-based control, namely FDI delays and manual system monitor definition.

Paper E: The second supervisory control approach is a combination of Randomized Blended Control and DRLBC, called Deep Reinforcement Learning Randomized Blended Control (DRLRBC). The neural network is used to map the states to an improved parameter interval for randomization, $\mathbf{s}_k \rightarrow [\alpha_k, \beta_k] \in [0, 1]$. This allows the interval to change during the execution of a task. The blending weights are sampled from the uniform distribution as before $\boldsymbol{\varphi}_k \sim \Delta(\alpha_k, \beta_k)$. As the system experiences changes in λ_k, the neural network can control the interval used for randomization. Instead of distributing control equally over all controllers, adding a neural network to guide the randomization is shown to further improve robustness to unknown operating

conditions.

Paper G: The third supervisory controller is based on an extension of DRLRBC. The key difference in this approach is the use of a neural network to learn the parameters of a continuous multivariate probability distribution, such as a Dirichlet distribution $\mathcal{D}(\delta)$. The blending weights are sampled such that, $\boldsymbol{\varphi}_k \sim \mathcal{D}(\delta_k)$, where δ_k parameterizes the distribution. DRLRBC learns two parameters to form an interval for each of the N controllers resulting in a 2*N-sized action space for the agent. However, $\mathcal{D}(\delta)$ only requires one parameter for each controller, which reduces the agent's action space by half. Another major advantage of using a Dirichlet distribution is that samples always sum to 1, which naturally enforces the blended control constraints on $\boldsymbol{\varphi}$, i.e. $\sum_i \varphi^i = 1$. This approach has shown the most promise empirically and overcomes the drawbacks of using a uniform probability distribution, while also simplifying the learning problem.

Paper C: The fourth supervisory controller is based on Q-learning with a simplified blending space as a preliminary investigation into the feasibility of online learning. The agent learns to generate $\boldsymbol{\varphi_k}$ from a discretized blending space, such that $\varphi^i \in [0, 0.1, \cdots, 1]$. This simplifies the action space of the agent compared to the previously presented controllers. While the results show the agent can improve robustness over time online, better-suited algorithms and implementations have been developed since this article was published, such as RL using decision trees [41].

Paper F: The final article compares the complexity of learning control using a hybrid blended control framework comprising of a learning-based supervisory controller and a set of model-based low-level controllers against directly learning full system control. The results show that it is significantly simpler to learn a supervisory controller because model-based controllers remove the need to learn nominal system control, which can take learning-based approaches a huge number of episodes.

3.1.4 Summary: Supervisory Control in Unknown Operating Modes

The results from articles A and B highlight the advantages of using blended control over switched control and present a novel supervisory control architecture based on randomization for unknown operating modes. The results presented in articles C - G show that a learning-based supervisory controller can be trained offline on known operating conditions Λ to improve the robustness to unknown operating conditions $\mathbf{U}$. Three neural network-based supervisory controller architectures, based on both randomization and identification, are presented. All three can improve robustness when $\lambda \in \mathbf{U}$ by training on the known conditions Λ offline. Preliminary results

show online learning can also improve robustness when $\lambda \in \mathbf{U}$, however better implementations are available that need to be investigated. The hybrid MMAC architecture ensures that the actions of the learning-based supervisory controller do not break the stability guarantees of the model-based low-level controllers. This also eliminates the need to learn nominal system control, which is a major advantage over other hybrid or learning-based fault-tolerant control approaches.

3.2 Low-Level Controller Tuning Problem

Open Problem 2 was investigated based on the assumption that a suitable control action $\mathbf{a}^*$ exists for $\lambda \in \mathbf{U}$. The action vector is ultimately dependent on the controller parameter set Θ. For Open Problem 1, the assumption transforms to the existence of an optimal controller parameter configuration θ^* for $\lambda \in \mathbf{U}$. This is the same assumption from two perspectives. If we further assume that one of the presented supervisory controllers is later used to approximate $\mathbf{a}^*$, then the low-level controller parameter tuning problem transforms into ensuring that $\theta^* \in \mathcal{C}(\Theta)$, where $\mathcal{C}(\Theta)$ defines the convex hull coverage of parameter set Θ.

Traditional tuning approaches treat every operating mode separately such that $\theta^i \in \Theta$ is tuned for $\lambda^i \in \Lambda$. While this approach has been shown to work in practice it leads to a rapidly growing low-level controller set as more conditions are considered. If the low-level controller set is instead tuned based on the coverage provided by the entire set $\mathcal{C}(\Theta)$, then Θ only grows if the required controller is not already covered by the set. For this kind of tuning approach, all operating modes in Λ must be considered together instead of separately.

3.2.1 Convex Hull-based Controller tuning - Paper I

The third contribution of this thesis is a novel learning-based controller tuning algorithm (Paper I) that automatically constructs a convex hull in the low-level controller parameter space given a set of operating modes Λ. Instead of optimizing for a single optimal parameter, the presented algorithm finds a region of suitable controller parameters for each mode. The regions for all modes are then combined using a convex hull. The extreme parameters that define the combined convex hull can be used as Θ, which ensures a sufficient coverage of the controller parameter space for Λ.

This type of tuning ensures that the size of the parameter set Θ is not dependent on the number of operating modes considered, i.e. $|\Theta| \neq |\Lambda|$. This is empirically demonstrated on a quadcopter trajectory tracking task with five known operating modes, $|\Lambda| = 5$. Using traditional tuning approach results in five low-level controllers,

i.e. $|\Theta| = 5$. The results show that the presented algorithm can achieve comparable robustness to five operating conditions using only three controllers, $|\Theta| = 3$. This is because the parameter coverage of the set as a whole is more important than the number of parameters within the set. Traditional tuning approaches do not consider the coverage of the controller set, which leads to an unnecessary growth in the size of Θ. This is the reason the presented algorithm can reduce the number of required parameters compared to other approaches.

The algorithm is based on a novel combination of existing algorithms including Particle Filtering [42], clustering, and convex hulls. All of these algorithms are extensively documented and parameter tuning is well understood. The presented algorithm requires one additional parameter that represents the acceptable loss threshold on a task. The extreme parameters used for Θ are currently selected manually and this can be automated in the future by integrating extremum-seeking methods.

3.2.2 Summary: Low-Level Controller Tuning for Unknown Operating Modes

A novel convex hull-based low-level controller tuning algorithm is presented in article I. The controller set found using the presented algorithm is empirically compared against a traditional tuning method on a quadcopter trajectory tracking simulation under several faults and disturbances. The results show the algorithm can reduce the size of Θ compared to traditional approaches, without a loss in performance. No other tuning algorithms based on convex hulls currently exist. The presented tuning algorithm can ensure that the controller parameter space is sufficiently covered for $\lambda \in \Lambda$ and provides best-effort coverage for $\lambda \in \mathbf{U}$, based on the *a priori* knowledge available at design time.

3.3 Thesis Contributions:

Fault-tolerant control approaches are generally split into two categories, active and passive. State-of-the-art active fault-tolerant control methods mainly rely on residual analysis to identify faults and disturbances and are generally considered more effective but are highly dependant on the speed of identification and accuracy of the model [43]. Passive approaches are entirely tuned at design time and offer little flexibility outside of the known operating conditions. Learning-based control approaches have made significant advances in recent years but currently have limited application to unknown operating conditions as no models or data exist for these situations and are lacking stability guarantees [33]. This thesis makes the following contributions:

A. A novel randomization-based supervisory control approach for unknown

operating conditions. This approach overcomes the dependency on *a priori* knowledge and mode-estimation delays inherent to existing identification-based supervisory control approaches.

B. A set of four novel learning-based supervisory controllers that have been empirically shown to improve a system's robustness to unknown operating conditions. A key advantage over existing hybrid and learning-based control approaches is that the presented supervisory controllers maintain the stability guarantees of the model-based low-level controllers during training.

C. A novel learning-based controller tuning algorithm that can reduce the number of low-level controllers compared to other tuning approaches, without a significant loss in performance. This is achieved by constructing a convex hull in the low-level controller parameter space based on all known operating conditions Λ.

Together, these contributions advance the current state of the art by offering a new perspective on how to address unknown operating conditions. Instead of identifying the exact operating condition using computationally expensive analysis and using a single controller, randomization and blended control are used to distribute system control over all controllers simultaneously. This novel concept is further extended using learning-based methods to improve the distributions used for randomization based on existing fault knowledge at design time. The novel low-level controller tuning algorithm was developed to complement the presented supervisory control methods and overcomes the large growth in the set of controllers required to provide fault tolerance using current methods. The results presented in the accompanying articles show that the presented approaches outperform current state-of-the-art approaches under unknown operating conditions and warrant further investigation and real-world tests.

3.4 List of Impacts:

A. This thesis provides control system designers with novel control mechanisms to improve a system's robustness to unknown operating conditions. Currently, no control methods exist for such situations.

B. The presented control methods can be integrated alongside existing fault-tolerant control systems and are applicable for any autonomous system, given that a model of the system exists.

C. The approach explored offers a new perspective on the combination of learning- and model-based controllers and allows for future developments from both

fields, such as better model-based controllers or faster training algorithms, to be incorporated without modifications.

D. The simulation files for all experiments presented in the articles accompanying this thesis have been made available for the research community. These are centered around quadcopter trajectory tracking under various external disturbances and internal system faults. The modular implementations are based on Python and MATLAB and provide a flexible basis for future research on robust fault-tolerant quadcopter control.

3.5 Scope

The overall goal of this thesis is to develop novel control methods that improve a system's robustness to unknown operating conditions. This is accomplished by combining methods from several communities to overcome the disadvantages of each on their own. The open problem that is studied is extending multiple model adaptive control to unknown operating conditions, where no *a priori* knowledge exists. This problem can be divided into two open sub-problem, supervisory controller optimization and low-level controller set optimization. All of the work was completed within the MMAC framework with the major focus on Blended Control. MMAC is a state-of-the-art control framework that can be applied to any system with multiple controllers. Blended Control ensures smooth transitions between the controllers and allows for the synthesis of new controllers from within the set. MMAC allows for a hybrid control framework that can take advantage of the stability properties of model-based design and the adaptability and ease of tuning offered by learning-based control methods. The modular framework design allows existing fault-tolerant control systems to be extended with the presented control methods. Future developments in both learning and control, such as better model-based controllers or real-time learning algorithms to be integrated without major modifications.

The supervisory control problem (Open Problem 2) is approached assuming a suitable action $\mathbf{a}^*$ for an unknown mode $\lambda \in \mathbf{U}$ can be interpolated from the action vector. The low-level controller tuning problem (Open Problem 1) is approached under the assumption that an optimal controller tuning θ^* exists for the unknown modes $\lambda \in \mathbf{U}$. Open Problem 1 can then be translated into a parameter coverage problem but it is currently not possible to guarantee $\theta^* \in \mathcal{C}(\Theta)$ without *a priori* knowledge of $\mathbf{U}$. Such situations are beyond the scope of this thesis, as the presented methods will not improve robustness when $\theta^* \notin \mathcal{C}(\Theta)$.

A preliminary comparison between the presented methods show that using a neural network to generate the underlying probability distribution for randomized blended control (Paper G) is most promising. A thorough comparison in terms of different neural network architectures and training algorithms for the presented supervisory controllers is left for future work and is expected to further improve the methods. The focus of this thesis is on investigating a variety of methods to give system designers more options depending on the capabilities of the system of interest. For example, micro-aerial vehicles with limited onboard computing power will benefit more from the computationally cheap randomized supervisory controller, instead of a learning-based supervisory controller, which could be more useful for larger systems.

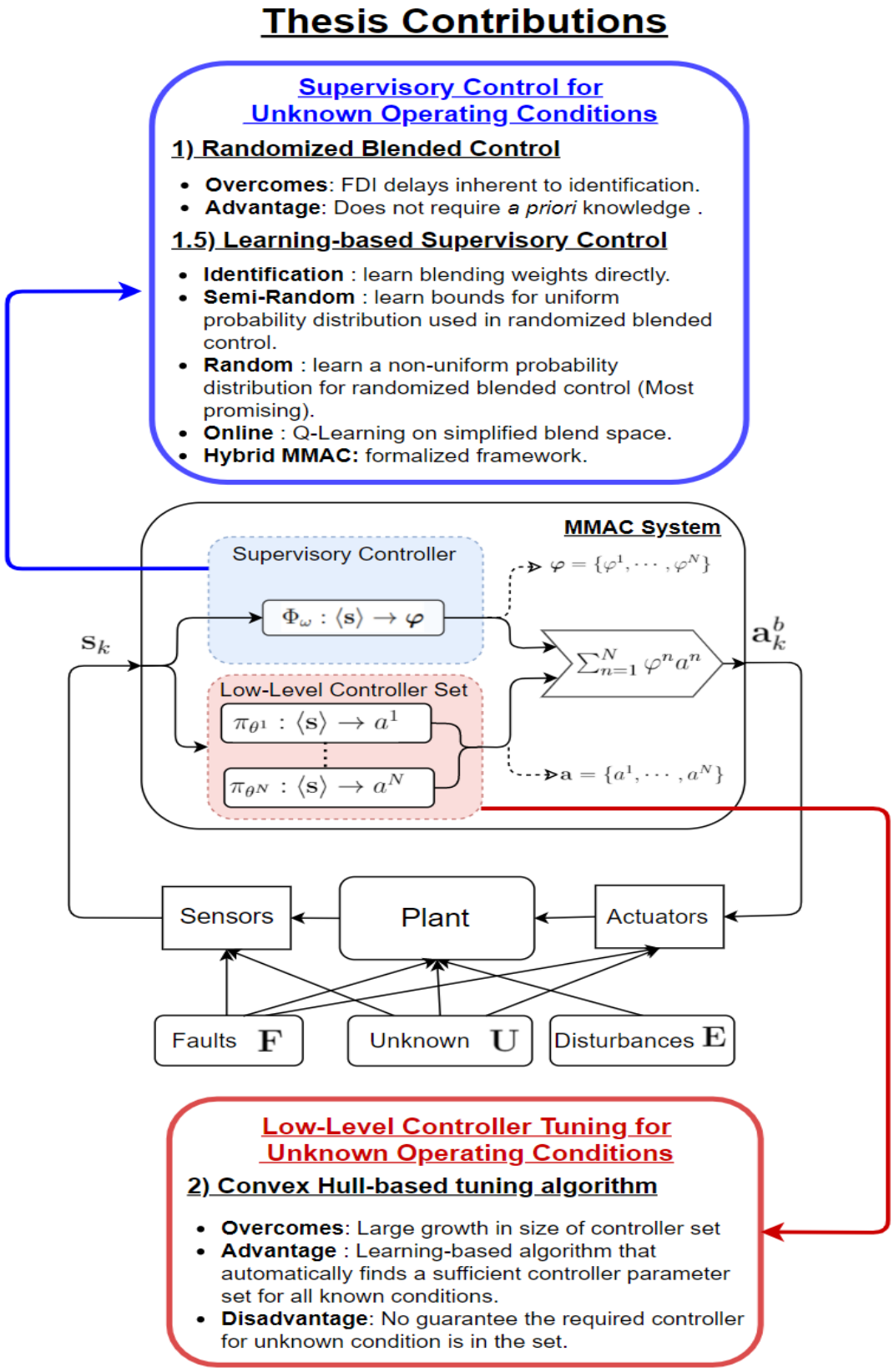


Figure 1.4: Thesis Contributions visualized on MMAC architecture.

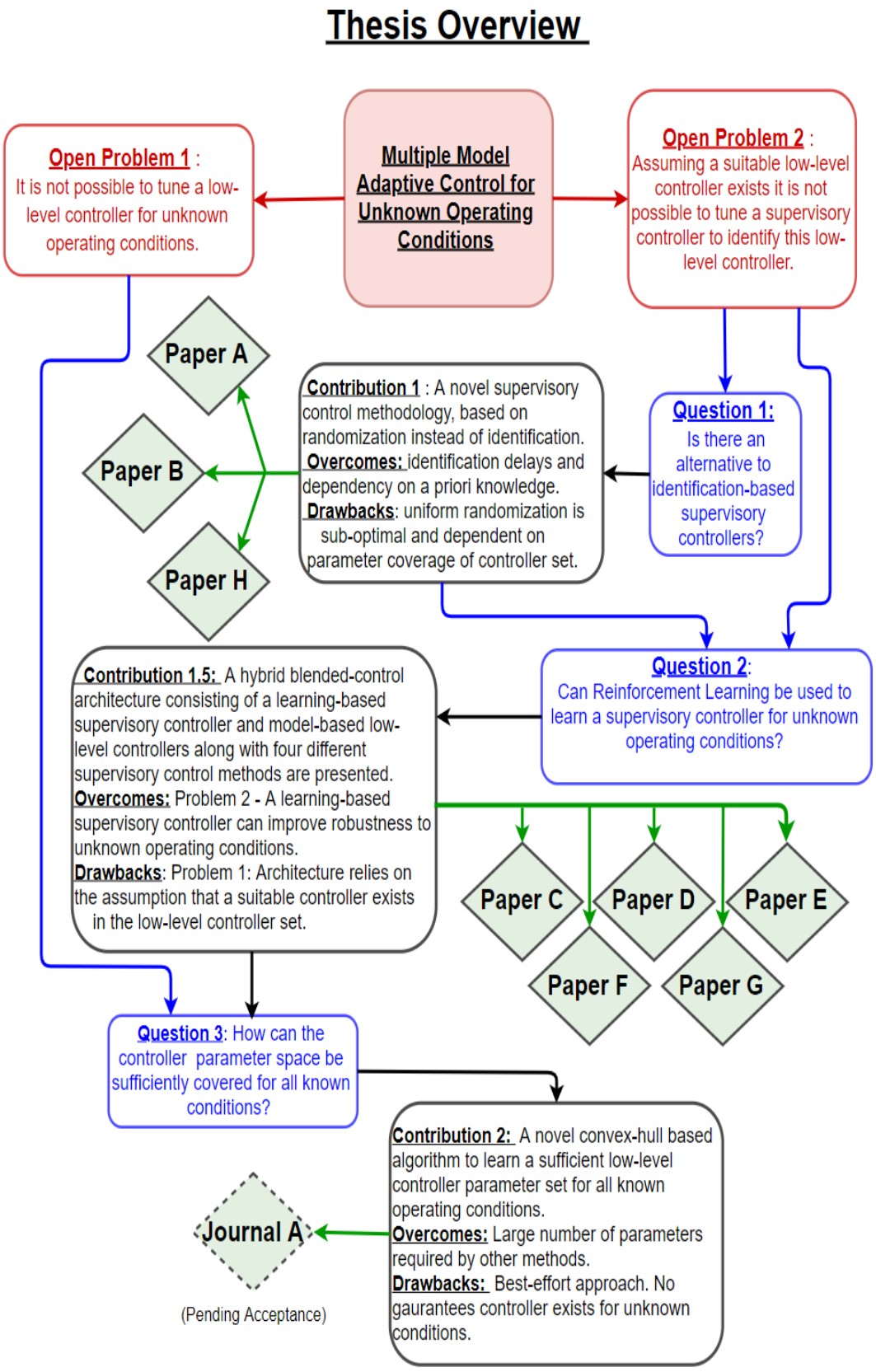


Figure 1.5: Thesis Overview Flowchart

4 Contributions

My contributions to the below articles are summarized here. I conducted all of the experiments and empirical validations that each article is based on. I also developed most of the novel algorithms that are presented, while it has to be mentioned that both Gregory Provan and Marcos Quinones-Grueiro were instrumental in providing essential definitions, mathematical underpinnings of the methodology, theoretical proofs, and writing to get these articles accepted. A percentage-based indicator is given along with a summary of each article to highlight my proportional contribution. Note: Paper H falls outside of the main contributions of this thesis.

Paper A represents an initial robustness investigation and comparison of Switched and Blended control. The empirical validation is conducted using MPC controllers on a two-tank system experiencing valve and pump faults under various FDI delays. The main results are that blended control is less impacted by FDI delays and more robust to faults overall. This is because a switched framework usually requires a parameter to reach a certain threshold before a fault is identified and control is switched. I contributed the experimental implementation, result analysis, image generation and the experimental section in the article.
Contribution: 50% , Co-Authors: 50%

Paper B proposes a novel supervisory control methodology based on randomization instead of identification, called Randomized Blended Control (RBC). The main focus of this article is the theoretical stability and stabilizability properties of using a randomized blending weight. The main contribution of this article is that RBC is a theoretically sound and empirically effective supervisory control method. It was validated on a real-time quadcopter trajectory tracking task under rotor faults. The key property that makes randomization comparable to identification-based approaches is the computational simplicity that allows for the weights to be re-sampled at every iteration of the control loop, overcoming the computational delay experienced by identification-based methods. For unknown operating conditions, this approach can overcome the requirement of *a priori* knowledge but is sub-optimal due to the uniform probability distribution used to sample the weights. This paper forms the basis for safe learning-based supervisory control. An untrained and randomly initialized learning-based controller will essentially behave like randomized blended control at the start of the training phase. My contributions to this article were the concept design and experimental implementation and evaluation. The mathematical formulation, stability analysis and other theoretical aspects were done by my co-authors.

Contribution: 30% , Co-Authors: 70%

Paper C shows initial results for using a learning-based supervisory controller. Q-learning is used to train a supervisory controller to generate the blending weights under unknown operating conditions. The main result is a demonstration that the blending weights can be learned online within a few hundred training epochs but this is not fast enough for real-time learning. The experiments are conducted on a quadcopter trajectory tracking task under rotor loss of effectiveness. I wrote the majority of the article (from section 4 on-wards) and conducted all of the experimental work, algorithm design and result analysis.
Contribution: 70% , Co-Authors: 30%

Paper D utilizes an offline training method to train a supervisory neural network controller on known operating conditions to directly generate the blending weights. The empirical validation is carried out on a quadcopter trajectory tracking task under heavy rotor faults. The main result is that a neural network can improve the robustness to known operating conditions compared to a switched controller. This removes the need to manually define the system monitors of a model-based supervisory controller as well as improving the reaction speed of the supervisory controller. A drawback is that for unknown operating conditions, the supervisory controller does not contain randomization in the control loop. My contributions to this article were the algorithm design, description and implementation of experimental analysis. Sections 1 and 4 were contributed by my co-authors.
Contribution: 70% , Co-Authors: 30%

Paper E investigates a combination of the methods proposed in Paper B and D. A neural network is used to generate an upper and lower bound, which are used as a subspace to uniformly sample the blending weights. This combines the advantages of RBC and learning-based supervisory controllers and improves the robustness to unknown operating conditions by incorporating randomness into the control loop. The main contributions of this paper are a novel supervisory controller and empirical validation on a quadcopter trajectory tracking task under known and unknown operating conditions including rotor faults, wind disturbances, and attitude and position noise. I contributed the algorithm design, description and implementation of experiments and analysis and wrote the majority of the article.
Contribution: 70% , Co-Authors: 30%

Paper F formalizes the general hybrid control framework investigated throughout this thesis. The main contributions are a comparison of pure learning-based

controllers with a hybrid control framework in terms of the complexity of the learning problem. The experiments are conducted on a quadcopter trajectory tracking task under rotor faults. The hybrid approach can utilize the benefits of both model-based and learning-based control methods. The analysis shows that utilizing model-based controllers for low-level control and applying learning strictly on a supervisory-level drastically simplifies the problem size. This is mostly due to the smaller action space of the supervisory control problem. An extension to the framework presented in Paper E is also included. The neural network generates the mean and standard deviation of the probability distribution used for RBC directly, instead of an upper and lower bound. This removes the limitations of using a uniform distribution. I contributed the implementation of experiments and analysis and wrote the majority of the article. Section 1 and 2 were contributed by my co-authors.

Contribution: 70% , Co-Authors: 30%

Paper G incorporates a training method from the field of robust reinforcement learning called Domain Randomization to train the supervisory controllers. The control architecture is the same as in Paper F but the training methods investigated can apply to any of the neural network-based supervisory controllers presented in this thesis. Empirical validation is done using trajectory tracking experiments for a quadcopter subject to rotor faults, wind disturbances, and severe position and attitude noise. The performance is compared to RBC to analyze the improvement obtained through training. The results show that training can drastically improve the overall performance of the system in a variety of operating conditions. I contributed the experimental design and implementation as well as writing most of the sections while my co-authors contributed sections 1,3 and 4.3.

Contribution: 60% , Co-Authors: 40%

Paper I presents a novel low-level controller tuning algorithm based on the construction of a convex hull in the low-level controller parameter space. Traditional point-based estimation methods result in a large growth in the number of required controllers. The algorithm is empirically validated on a quadcopter trajectory tracking task under five different operating conditions by tuning PID controllers. The results show that the controller set obtained through the presented approach yields a smaller controller set than traditional tuning methods without any loss in performance. For this article I contributed algorithm design, the experimental implementation and analysis and wrote most of the sections. My co-authors contributed sections 1 and 3.

Contribution: 70% , Co-Authors: 30%

Table 1.1: Part 1: Empirical comparison and results overview for each article.

Article	**Baseline**	**Architecture**	**Test Domain / Controller**	**Results**
Paper A	Switched Control	Blended Control	Two Tank System / Model Predictive Controller	Blended control responds faster than switching and is less affected by FDI delays and false positives.
Paper B	Switched Control with optimal FDI	***Randomized Blended Control - RBC (novel)***	Quadcopter / Cascading PID	Randomization is a theoretically sound and empirically effective supervisory control methodology for unknown operating conditions and doesn't require FDI.
Paper C	N/A	***Q-Learning Blended Control (novel)***	Quadcopter / Cascading PID	A supervisory controller can improve blended control online using Q-learning on a discretized blend space while maintaining the stability guarantees of the low-level controllers.
Paper D	Switched Control	***Deep Reinforcement Learning Blended Control (novel)***	Quadcopter / Cascading PID	A neural network trained offline on known operating conditions to directly generate the blending weights. This overcomes FDI delays and manual tuning efforts.
Paper E	Switched Control, Randomized Blended Control	***Deep Reinforcement Learning Randomized Blended Control (novel)***	Quadcopter / Cascading PID	A neural network can be trained offline on known operating conditions to generate an upper and lower bound to improve RBC by sampling from within the bounds.

Table 1.2: Part 2: Empirical comparison and results overview for each article.

Article	**Baseline**	**Architecture**	**Test Domain / Controller**	**Results**
Paper F	Direct Learning, Randomized Blended Control	***Deep Reinforcement Learning Randomized Blended Control - version 2 (novel)***	Quadcopter / Cascading PID	The learning complexity is drastically reduced when model-based controllers are utilized in a hybrid control framework compared to directly learning system control. The neural network used in this article generates the mean and standard deviation of the Randomized Blended Control probability distribution, instead of an upper and lower bound.
Paper G	Randomized Blended Control	Deep Reinforcement Learning Randomized Blended Control - version 2	Quadcopter / Cascading PID	Domain randomization from robust reinforcement learning can be used to further improve the robustness of neural network-based supervisory controllers to unknown operating conditions.
Paper H	Switched Control, Leader - Follower	Blended Control	Car-Steering / Model Predictive Controller	Blended Control is less affected by badly performing low-level controllers than switched control. Sharing control reduces the impact of any individual controller.
Paper I	Switched Control	Randomized Blended Control	Quadcopter / Cascading PID	A convex hull-based controller tuning algorithm that can reduce the number of required low-level controllers compared to traditional tuning methods for known operating conditions, without a loss in performance.

5 Publications

5.1 Conference Papers

A Yves Sohège and Gregory Provan. *Comparing Switching vs. Mixing Model-Predictive Control for Robust Fault-Tolerant Control* published in *Proceedings of the 28th International Workshop on Principles of Diagnosis* (DX'18), Brescia, Italy, 2018.

B Gregory Provan and Yves Sohège. *Fault-Tolerant Control for Unseen Faults using Randomized Methods* published in the *Proceedings of the 4th International Conference on Control and Fault-Tolerant Systems*, Casablanca, Morroco, 2019.

C Yves Sohège and Gregory Provan. *Online Reinforcement Learning for Trajectory Following with Unknown Faults* published in the *Proceedings of the 26th AICS Irish Conference on Artificial Intelligence and Cognitive Science*, Dublin, Ireland, 2018.

D Yves Sohège, Marcos Quiñones-Grueiro and Gregory Provan. *Unknown Fault Tolerant Control using Deep Reinforcement Learning: A blended control approach* published in the *Proceedings of the 30th International Workshop on Principles of Diagnosis*, Klagenfurt, Austria, 2019.

E Yves Sohège, Marcos Quiñones-Grueiro, Gregory Provan and Gautam Biswas. *Deep Reinforcement Learning and Randomized Blending for Control under Novel Disturbances* published in the *Proceedings of the 1st Virtual IFAC World Congress*, Berlin, Germany, 2020.

F Yves Sohège, Marcos Quiñones-Grueiro and Gregory Provan. *Neural-Symbolic Fault Tolerant Control for Quadcopter Trajectory-Following Tasks* published in the *Proceedings of the 31st International Workshop on Principles of Diagnosis*, Tennessee, USA, 2020.

G Yves Sohège, Marcos Quiñones-Grueiro and Gregory Provan. *A Novel Hybrid Approach for Fault-Tolerant Control of UAVs based on Robust Reinforcement Learning* published in the *Proceedings of the International Conference on Robotics and Automation 2021*, Xi'an, China, 2021.

H Gregory Provan and Yves Sohège. *Comparison of control and cooperation frameworks for blended autonomy* published in *Proceedings of the 16th European Control Conference*, Limassol, Cyprus, 2018.

5.2 Journal Articles

I Yves Sohège, Marcos Quiñones-Grueiro and Gregory Provan. *Learning Sufficient Low-Level Controller Parameters for Blended Control in Non-Stationary Conditions* published in the *Journal of Robotics and Autonomous Systems 2021* (pending acceptance)

6 Conclusion & Future Work

The control of autonomous systems in unknown operating conditions is an open problem for both the optimal control and reinforcement learning communities, due to the lack of *a priori* knowledge in terms of models and simulators. For the large-scale integration of autonomous systems in the real world, where unknown operating conditions are unavoidable, this problem needs to be addressed. Currently, system designers have no methods at their disposal to handle unknown operating conditions. This impacts the overall robustness of autonomous systems operating in the real world.

This thesis addresses many of these problems by providing system designers with a set of control mechanisms to improve robustness in unknown operating conditions. This thesis is focused on a hybrid control framework that can overcome the disadvantages of both model-based and learning-based control approaches. This is accomplished by dividing the control problem into supervisory control and low-level control. Model-based controllers can provide stability guarantees, which give them a significant advantage over learning-based controllers for the low-level control problem. Learning-based supervisory controllers implemented using neural networks can overcome the fault-detection and isolation delays inherent to most model-based supervisory controllers. Further, it significantly reduces the effort required by system designers to define a supervisory controller, as training is mostly automated.

The hybrid framework investigated in this thesis provides a new perspective on how to combine learning and model-based control. This new framework ensures the learning-based component can influence the control system, while maintaining many existing model-based stability guarantees. This is a significant advantage over other hybrid control frameworks. There are two main contributions. 1) A set of five novel supervisory control architectures that can improve robustness to unknown operating conditions. 2) A novel convex hull-based low-level controller tuning algorithm that automatically finds a sufficient parameter coverage for all known operating conditions. This provides a solid basis for the supervisory controllers presented while reducing the number of required low-level controllers compared to traditional tuning approaches.

The methods presented can be applied to any autonomous system, where models are available and can be integrated alongside existing fault-tolerant control systems designed to handle known operating conditions. The choice of control method for unknown operating conditions largely depends on the available processing power of the system of interest. Randomized Blended Control can be useful for small systems, such as micro-aerial vehicles with limited processing power, due to the computational simplicity. Larger systems, where this limitation is not as prominent, may benefit more from one of the computationally heavy, learning-based supervisory controllers that can improve control in unknown operating conditions even further.

As the learning and control community mature, faster training algorithms and better model-based controllers will be developed. By decomposing the control task into separate problems, the developments from either community can be integrated into the hybrid framework without modifications. This is due to the modular design of the multiple model adaptive control architecture used as a basis for the presented approach.

There are many ways this work can be extended in the future. The most obvious next step is to validate the results presented in this thesis in a real-world setting. Another promising avenue of research is the online learning extension as this was shown to be feasible but better algorithm and simulator implementations are now available. A formal comparison of the three offline learning supervisory controllers presented would also be interesting as this thesis only demonstrates the feasibility of each approach.

One significant limitation of the presented hybrid approach is the lack of online reconfiguration mechanisms for the low-level controller set for situations where the required control signal is outside of the convex hull of the low-level controller parameter set defined at design time. In future work, this could be overcome by incorporating the online-learning extensions not only on the supervisory level but also as a low-level controller reconfiguration mechanism to adapt to extreme situations.

Part II

Publications

Paper A

Comparing Switching vs Mixing Model-Predictive Controllers for Robust Fault-Tolerant Control

Abstract

We conduct a comparative study between two approaches for combining signals from several Model-Predictive Controllers (MPCs) designed for different fault scenarios. The first is MPC switching where a switch dictates which of the MPC controllers is currently active. The second is MPC mixing where all MPCs are running concurrently and their outputs are blended in proportion to the current estimate of fault state. We demonstrate results using a gravity drained multi-tank system. Our empirical results show that the mixing approach responds more quickly to faults than the switching approach. Further, we show that the speed and accuracy of fault isolation has a critical impact on fault tolerance.

1 Introduction

A fault-tolerant control system (FTC) is a system that is able to identify and recover from system faults and continue operation as normal or to maintain stability to a desired level of overall performance. The need for such systems in areas such as aerospace and industrial processes, where safety and reliability is paramount, has motivated significant research into the design and optimisation of these systems. Examples of such systems can be seen in aircraft [44], spacecraft [45], autonomous quad-copter systems [2], power plants, chemical reactors and ground vehicles, among many more.

FTCs can be divided into two types, passive and active. Passive fault tolerant controllers are designed off-line against predefined models for certain operating conditions and have no ability to react to unanticipated faults. Passive FTC enables fast adaptation to faults, within the predefined operating conditions. Active FTC uses on-line data to reconfigure the controller to stabilise the plant. They have a built in fault detection mechanism which allows them to react to pre-defined faults, but makes the controller reliant on the accuracy on this fault-detection unit. For a comprehensive study between the two approaches see [46].

Both active and passive FTC rely, to varying degrees, on specifying the space of faults that the system will encounter. For passive FTC, approaches such as mixing of controllers tuned to nominal and failure modes are used to maintain system stability, e.g., [2]. Analogously, active FTC can rely on being able to detect pre-specified faults, such as using a bank of observers, with each observer tuned to a particular fault, e.g. [47]. For complex systems, it is impossible to pre-specify all faults, since there are too many fault combinations to consider, and it may be impossible to know all possible faults as *a priori*. As a consequence, it is imperative that a system designer understand the space of possible faults and their impact on a system. Creating a controller that can interpolate its control law from predefined edges of the fault space allows it to adapt to unknown/undefined faults by computing the optimal control policy for this fault state. Very little work has been conducted on exploring the space of faults and their impact on active versus passive FTC.

FTC extends traditional controllers with a method of detecting and tolerating faults. The main purpose of a control system is reference tracking and many different types of controllers including Model Predictive Controllers (MPCs), Proportional Integral Derivative (PID), Feedback Linearisation (FL), etc., have been designed for this purpose. We adopt an MPC approach, which is based on numerical optimization and is categorised as an optimal control strategy [48]. At every time step the MPC solves

an optimization problem to obtain the optimal control input for the current system state. An MPC can be tuned to handle specific error models to make it a FTC. Several MPCs, each designed to handle a separate fault scenarios, can be combined to create a controller that is more robust than any individual MPC can be at maintaining the stability of the system.

We focus on hybrid systems control, where we assume a system that can operate in a set of N modes, with a different set of dynamics for each mode. We assume that we have a subset of N_f failure modes, with $N - N_f$ nominal modes. For each of the nominal modes we pre-compute a control setting. We also need to tolerate when certain failure modes occur, which is the focus of this article.

In this article we conduct a comparative study between two approaches for combining several MPCs designed for different fault scenarios. The first is MPC switching where a switch dictates which of the MPC controllers is currently active. The switching signal is generated by the Fault Detection Unit (FDU) so once a fault is identified the corresponding controller can be activated to handle this fault. The second is MPC mixing where all MPCs are running concurrently and their outputs are blended in proportion to the current fault state estimate generated by the FDU. We demonstrate results using a gravity drained multi-tank system.

2 Related Work

This work builds on extensive prior work in fault-tolerant control (FTC) and Fault Detection and Isolation (FDI).

In the area of FTC [25], a significant body of work has been developed and applied to real-world systems. [49] presents a recent overview of FTC, and [50] presents FTC with relation of system safety. Traditional methods for FTC employ a bank of observers coupled with dedicated controllers, and perform discrete switching. This approach enables designers to tune the system to dedicated faults, but the speed of the system hinges on the speed of FDI. More recent approaches use mixing controllers, e.g., [51, 52], which blend the outputs of multiple controllers and are less reliant on FDI.

Our work builds on research into the use of weighted multiple models for adaptive estimation and control. Early work, e.g., [53, 54] used multiple Kalman filter-based models to improve the accuracy of the state estimate in estimation and control problems. This early work was then extended to applications, e.g., real-world problems

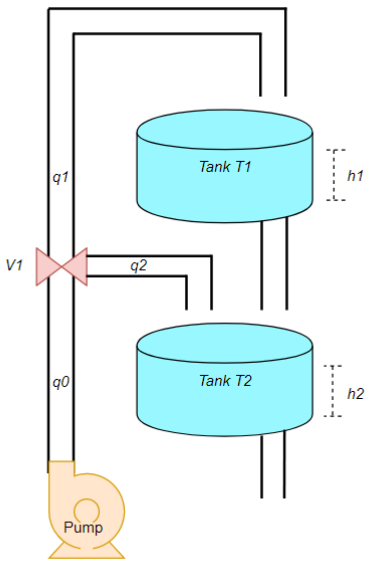


Figure A.1: Two tank gravity drained system.

[55] and fault detection [56].

Blended or weighted multiple model adaptive control (WMMAC) has been used to make control more robust, e.g., [57, 58]. The use of multiple models within the context of MPC, denoted Multiple Models Predictive Control (MMPC), has been addressed by [59]. Beyond this focus on robustness, there have been few applications of multiple blended models to FTC outside of [2]. Our work is novel in its investigation of WMMAC for FTC. We are not aware of prior work that empirically compared active and passive FTC approaches, or that compared the impact of faults on switching vs. mixing supervisory FTC.

3 Running Example

We illustrate our approach using the two-tank system shown in Fig. A.1. Tank T_i has area A_i and inflow q_{i-1}, for $i = 1, 2$. This system connects together two tanks, with a valve V_1 regulating the proportion of the flow q_0 and q_1 into tanks 1 and 2 respectively. The pump flow q_0 and value V_1 can be set between 0 and 1. A valve setting of 0 will cause the entire flow to go into tank 1 and a pump setting of 0 will turn the pump off completely. The control objective is to modify the pump and valve settings to maintain a reference height in both tanks. Water from tank 1 will flow into tank 2 at a constant rate and water will flow out of tank 2 at a constant rate. Sensors give us the current water level h_1 and h_2 in tank 1 and tank 2, respectively.

Both tanks T_1 and T_2 get filled from a pump, with measured flow q_0. The proportions of the flow into each tank are controlled by V_1, hence, our control inputs are $u = \{q_0, V_1\}$.

We assume that we do not directly measure any flows other than the inlet flows q_0 and valve setting V_1. As a consequence, we use the tank heights as a proxy for deriving flows through the two-tank system. We can control the valve settings where we assume a continuous-valued setting ranging over $[0, 1]$.

We can use basic physics to create a model of the 2-tank system by observing mass-balance requirements on each of the tanks, where tank T_i has area A_i, for $i = 1, 2$. The state equations are given by:

$$A_i \frac{dh_i}{dt} = q_{i-1} - q_i,$$

where q_{i-1} denotes the flow into tank i, and q_i denotes the flow out of tank i.

According to Torricelli's Law, flow q_i out of tank i, with liquid level h_i, into tank j, is given by:

$$q_i = \gamma \sqrt{2gh_i}, \tag{A.1}$$

where the coefficient γ is used to model the area of the drainage hole and its friction factor through the hole, and g is gravitational acceleration.

We can use equation A.1 to derive the following equations, since the inflow into tank 2 equals the outflow of tank 1:

$$\begin{aligned} \dot{h}_1 &= q_0 - c_1 \cdot \sqrt{h_1} \\ \dot{h}_2 &= c_1 \cdot \sqrt{h_1} - c_2 \cdot \sqrt{h_2}, \end{aligned} \tag{A.2}$$

where the constants c_1, c_2 summarize the system parameters representing cross-sectional areas, friction factors, gravity, etc. Consequently, the parameter set is given by $\Theta = \{c_1, c_2, c_3, g\}$. We assume that we can measure the height of liquid in each tank. The set of state variables is $\{h_1,\ h_2,\ V_1, q_0\}$, and the set of controllable variables is $\{q_0, V_1\}$.

To formally transform a non-linear system into a linear one, we need to use techniques like small signal linearization or perturbation theory [60, 61]. For example, in small signal linearization, an equilibrium point x_0 of a fault-free non-linear function is first identified, about which the perturbed non-linear function is expanded: $x = x_0 + \delta x$. Then we can use a Taylor series expansion, neglecting the higher order terms, to obtain the linear function.

4 Fault Tolerant Control Schemes

A key aspect of FTC is the ability to adopt a control law that compensates for a system fault. Because having a single nominal-mode plant model does not allow us to simulate faulty behaviours, modern FTC uses multiple models that attempt to cover the space of failure behaviours [62]; many diagnostics approaches also use multiple models, e.g., [63].

However, in realistic problems, the dimension of the plant and of the unknown parameter vector are large. As a consequence, the number M of models needed to satisfy stability and controllability conditions given faults becomes prohibitively large, since M increases exponentially with the dimension of the unknown parameter vector. In addition, switching results in discontinuous control signals, and identification and control are coupled [62].

In the following, we study the impact of various design choices on the FTC schemes for two of the most advanced approaches, switching MPCs and mixing MPCs.

4.1 State-Space Model

Consider a linear time-invariant (LTI) discrete-time system of the form:

$$\begin{aligned} \dot{\boldsymbol{x}}(k+1) &= A_j \boldsymbol{x}(k) + B_j \boldsymbol{u}(k) + \boldsymbol{w} \\ \boldsymbol{y}(k) &= C\boldsymbol{x}(k), \end{aligned} \tag{A.3}$$

where $\boldsymbol{x},\ \boldsymbol{u},\ \boldsymbol{y}$ are the state, control and observation vectors, respectively, and $A_j,\ B_j$ are state matrices for mode j. Finally, C is the observation matrix. We assume that the system can operate in N possible modes $\mathcal{M} = \{\mathcal{M}_1, ..., \mathcal{M}_N\}$, with individual dynamics for mode j ($j = 1, \cdots, N$), captured by the matrices $A_j,\ B_j$.

Since the hybrid system dynamics use a model for each distinct mode, we must also have a unique control law for each distinct mode. More precisely, if we have N modes, then we must have a set $\Lambda = \{\lambda_1, \cdots, \lambda_N\}$ of controllers, with a switching algebra to define when to switch from λ_i to λ_j for $i \neq j,\ i, j \in \{1, ..., N\}$.

We thus characterize the system's N modes by (i) a set of N controllers Λ, and (ii) a set of N parameter settings, i.e., we can partition the system's parameter space Ω into a set of sub-spaces $\{\Omega_1, \Omega_2, \cdots \Omega_N\}$, such that sub-space Ω_i corresponds to controller C_i, for $i = 1, ..., N$. In other words, we assume that we can define the parameter setting sub-spaces such that there exists a controller Λ_i that can guarantee a desired performance level for Ω_i, $i = 1, ..., N$.

4.1.1 Observer-Based Control

We assume that we control the system (in mode i) using a state (Luenberger) observer based on a state-space model with observer matrix L. Using the observed system with observed state and measurement, $\hat{\boldsymbol{x}}(k) \in \mathbb{R}^n$ and $\hat{\boldsymbol{y}}(k) \in \mathbb{R}^p$, respectively:

$$\begin{aligned} \hat{\boldsymbol{x}}(k+1) &= A_i\hat{\boldsymbol{x}}(k) + B_i\boldsymbol{u}(k) + \boldsymbol{w}(k); \\ \hat{\boldsymbol{y}}(k) &= C_i\hat{\boldsymbol{x}}(k); \end{aligned} \tag{A.4}$$

we obtain the observer equations:

$$\begin{aligned} \hat{\boldsymbol{x}}(k+1) &= A_i\hat{\boldsymbol{x}}(k) + B_i\boldsymbol{u}(k) + L_i(\boldsymbol{y}(k) - C_i\hat{\boldsymbol{x}}(k)); \\ \boldsymbol{r}(k) &= \boldsymbol{y}(k) - C_i\hat{\boldsymbol{x}}(k); \\ \boldsymbol{u}(k) &= -K_i\hat{\boldsymbol{x}}(k), \end{aligned}$$

where $\boldsymbol{r}(k) \in \mathbb{R}^p$ is the residual $\parallel \boldsymbol{y}(k) - \hat{\boldsymbol{y}}(k) \parallel$ for some norm $\parallel \cdot \parallel$. We tune the control matrix $K_i \in \mathbb{R}^{l\times p}$ and observer matrix $L_i \in \mathbb{R}^{n\times p}$ so that the closed-loop system and error dynamics are stable. We can rewrite these equations such that we obtain

$$\hat{\boldsymbol{x}}(k+1) = (A_i - B_iK_i)\hat{\boldsymbol{x}}(k) + L_i(\boldsymbol{y}(k) - C_i\hat{\boldsymbol{x}}(k)) \tag{A.5}$$

by substituting $-K_i\hat{\boldsymbol{x}}(k)$ for $\boldsymbol{u}(k)$ into the state equation.

4.1.2 Observer-Based Diagnosis

We partition our modes into nominal and fault modes, $\mathcal{M}_\emptyset$ and $\mathcal{M}_f$, respectively, such that $|\mathcal{M}_\emptyset| + |\mathcal{M}_f| = N$. We model (actuator) faults using a multiplicative fault model with parameter $0 \leq \gamma \leq 1$, where $\gamma = 0$ corresponds to nominal function and $\gamma = 1$ to total failure. For every failure mode $\mathcal{M}_i \in \mathcal{M}_f$ we have a corresponding fault parameter γ_i. Hence we obtain the state variable equation for failure mode j:

$$\dot{\boldsymbol{x}}(k+1) = A_j\boldsymbol{x}(k) + B_j(1-\gamma_j)\boldsymbol{u}(k) + \boldsymbol{w}(k). \tag{A.6}$$

We use the residuals for mode identification. In this article we create a residual for every system mode, i.e., our observer indicates that mode i is active if the corresponding residual $r_i > \epsilon$ for some tunable threshold $\epsilon > 0$. If we have perfect tuning and no noise, then we have a monotonic relationship between γ_i and $r_i > \epsilon$: presence of fault i always leads to a detectable residual r_i corresponding to that fault.

4.2 Switching Model-Predictive Control

Given that our system operates in multiple possible modes, we use a controller that can switch between these modes. Fig. A.2 depicts a generic framework that enables control switching of various types. In this article we examine two switching behaviours: (1) discrete switching (where one control regime is only used at any time), and (2) mixed switching, where we use a proportional mixture of multiple controllers at once. This section provides an overview of these two FTC controllers.

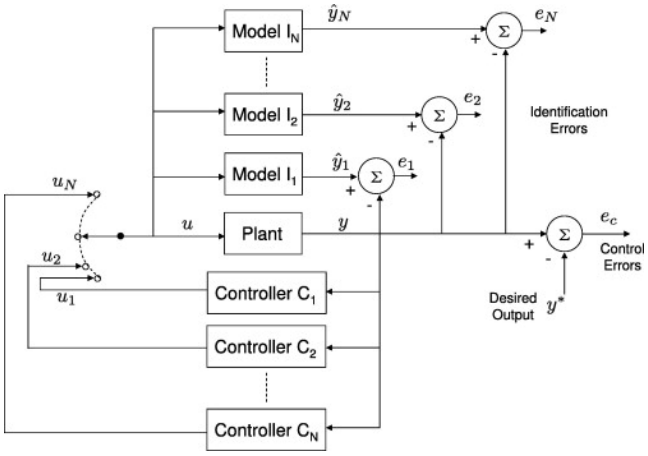


Figure A.2: Use of multiple controllers for FTC

By using candidate controllers designed off-line, the switching architecture can use multiple controllers, either individually or in a mixed mode. The multi-controller is not only capable of generating any of the candidate control laws but also, by controller interpolation, a stable mix of candidate control laws. We can design the mixing controller to converge exponentially quickly to meet the control objective, provided certain conditions on the plant input are met [64].

The architecture, shown in Fig. A.2, has two levels of control: (1) a low-level controller that generates finely-tuned candidate controls for each mode and (2) a high-level controller, called the supervisor, that influences the control by adjusting the low-level controller, typically by selecting or weighting candidate controllers, based on processed plant input/output data.

We now briefly outline the design of the supervisory controller that governs the switching behaviour.

4.2.1 Discrete Switching Controller

This section summarizes the MPC approach that we adopt. A traditional MPC controller includes a nominal operating point at which the plant model applies,

such as the condition at which you linearize a non-linear model to obtain the LTI approximation. We generalize that operating point in terms of a parameter sub-space. When we consider a single fault, then we can define that sub-space in terms of the multiplicative fault parameter γ. Since the actuator effectiveness is in the range $0 \leq \gamma < 1$, we have *a priori* knowledge on the bounds of $\bar{\gamma}$:

$$\bar{\gamma} \in \Omega \subset (0, 1]. \tag{A.7}$$

We can then partition Ω into a set of sub-spaces $\{\Omega_1, \Omega_2, \cdots \Omega_N\}$, where $\Omega_i = [\bar{\gamma}_i^{min}, \bar{\gamma}_i^{max}]$ and $\bar{\gamma}_i^{min}$ and $\bar{\gamma}_i^{max}$ are the lower and upper bounds, respectively, of interval Ω_i. For multiple faults, we must consider the fault vector $\boldsymbol{\gamma} = \{\gamma_1, ..., \gamma_m\}$. We now define a multi-dimensional partition $\boldsymbol{\Omega}$ into a set of sub-spaces $\{\boldsymbol{\Omega}_1, \boldsymbol{\Omega}_2, \cdots \boldsymbol{\Omega}_N\}$, where

$$\boldsymbol{\Omega}_i \subseteq \times_{j=1}^{m} \gamma_{ij},$$

and γ_{ij} denotes the i^{th} sub-space of the set of multiplicative fault parameter set indexed $j = 1, ..., m$. Given this partition, we design controller C_i to guarantee optimality when operating in sub-space $\boldsymbol{\Omega}_i$.

We then design a switching supervisor that, given an estimate of sub-space $\boldsymbol{\Omega}_i$, switches to the i^{th} controller.

We select the i^{th} controller if residual r_i is greatest, measured over all residuals that exceed a given threshold:

$$\Lambda_i(k+1) = \begin{cases} \Lambda_i(k) & \text{if } r_i < \epsilon_i \ \ \forall i \\ \Lambda_{i^*} | i^* =_{r_i \in \hat{R}} \{r_i\} & \text{if } r_i \geq \epsilon_i \end{cases} \tag{A.8}$$

4.2.2 Mixing Controller

We aim to design a mixing scheme for regulating the state vector $\boldsymbol{x}$ to a chosen set-point, assuming the nominal system matrices A and B are known, but multiplicative actuation errors γ are unknown. Given the system in equation A.3, sudden changes in γ will affect the system dynamics. In response, we tune the mixing scheme to blend the controllers appropriately. Given a set of controllers Λ_i, for $i = 1, ..., N$, each of which guarantees optimality when operating in sub-space $\boldsymbol{\Omega}_i$, we use a mixing supervisor for sub-optimal conditions. In other words, when sub-optimality occurs, i.e., parameters do not lie within some Ω_i, we must mix “appropriate" controllers.

In this article we use a linear combination of weighted controller inputs to restore stability given a pre-specified set of faults. In other words, given a set of N controllers

$\Lambda = \{\Lambda_1, \cdots, \Lambda_N\}$ and corresponding probability distribution $\{\varphi_1, \cdots, \varphi_N\}$, our applied control is given by

$$\Lambda^* = \sum_{i=1}^{N} \varphi_i \Lambda_i. \tag{A.9}$$

We compute the distribution φ using the residuals for the N models, $r_i,\ i = 1, ..., N$:

$$\varphi_i : r_i \rightarrow [0, 1]. \tag{A.10}$$

We perform this mapping using the following three steps.

1. Discretization of Residuals We discretize the residual space for fault i into a set of intervals of the form

$$[0, \epsilon), [\epsilon, \epsilon + \delta), ..., [\epsilon + m\delta, r^{max}), [\geq r^{max}]$$

where ϵ and δ are appropriate thresholds and r^{max} is a maximal residual value that indicates the fault magnitude is significant.

2. Weight Assignment to Intervals Given these intervals, we compute a weight for the i^{th} residual interval as follows:

$$w_i(k+1) = \begin{cases} 0 & \text{if } r_i \in [0, \epsilon) \\ \alpha & \text{if } r_i \in [\epsilon, \epsilon + \delta) \\ \cdots & \cdots \\ k\alpha & \text{if } r_i \in [\epsilon, \epsilon + k\delta) \\ \cdots & \cdots \\ 1 & \text{if } r_i \geq r^{max} \end{cases} \tag{A.11}$$

where α is a [0,1] weight chosen by appropriate controller tuning, and $m \in \mathbb{Z}^+$.

3. Weight Normalization Given the weights, we need to convert them into a probability distribution so that our cumulative control signal (equation C.1) remains stable. We assume that we start in a nominal mode, and our objective is to define a new blended controller given a residual $\boldsymbol{r}(k)$. We denote the current (nominal) controller at time step k as $\Lambda_j(k)$; we assume that this controller has probability $\varphi = 1$ assigned to a single mode. We denote alternative controllers as $\Lambda^* = \{\Lambda_l\},\ l = 1, ..., N,\ l \neq j$. We denote our probability distribution for controllers $\Lambda_1(k), ..., \Lambda_N(k)$ using $\varphi(k) = \{\varphi_1(k), ..., \varphi_N(k)\}$. We compute a probability for time step $k + 1$ from the weights through normalization. If all w_i are 0, then we maintain the current controller, $\Lambda_j(k)$; otherwise, if some $w_i > 0$,

we update our weights as given below to generate a new blended controller:

$$\varphi_i(k+1) = \begin{cases} 1-\mathcal{S} & \text{if } i = j \\ \frac{w_i}{\pi} & \text{if } j \neq i \end{cases} \tag{A.12}$$

where $\pi = \sum_i w_i$ and $\mathcal{S} = \sum_i \frac{w_i}{\pi}$.

Example:

Consider the running example of a two tank system described in Section 3. We have a single nominal mode Λ_1 and two failure modes Λ_2 and Λ_3, denoting pump and valve faults, respectively. Our weight intervals are $\{[0, .01),\ [.01, .02), [.02, .03), , \cdots, [\geq .1], \}$, i.e., $\epsilon = \delta = 0.01$ in equation A.11. Assume that we are currently in the nominal mode and detect anomalous residuals for the pump in the intervals [.01,.02) and [.02,.03), corresponding to 1% error and 2% error, respectively. Taking a tuned value of $\alpha = 10$, that leads to the weights $w_2 = 0.1,\ w_3 = 0.2.$ w_1 is calculated at $1 - \mathcal{S}$ where $\mathcal{S}$ is $\sum_i w_i$ and $i = [2, 3]$, which makes $w_1 = 0.7$. Then from Equation 9 we generate the new control signal to be:

$$\Lambda^*(k+1) = 0.7\Lambda_1(k) + 0.1\Lambda_2(k) + 0.2\Lambda_3(k) \tag{A.13}$$

4.3 Stability Properties

The weighting algorithm plays an important role in the control and stability properties of weighted multiple model control (WMMC) frameworks. The closed-loop stability of a WMMC system depends on three conditions [64]: (1) the model set includes the true model(s) of nominal operation of the plant (or the closest such models); (2) the weighting algorithm converges correctly; and (3) each local controller stabilizes its corresponding model. In addition, the convergence rate of a weighting algorithm will have effect on the transient process of the WMMC system.

Providing stability and optimality guarantees for controllers that use multiple models is known to be a difficult task, e.g., [58, 65]. The study of stability properties of switched and hybrid MPC systems has focussed on models defined using piecewise affine systems [66]. Proof of stability assumes that a supervisory controller would switch among low-level controllers with neighbouring, locally affine regions, such that each low-level controller is accurate within its own affine region [67].

Proving stability for FTC is beyond the scope of this article. Nevertheless, we can make a number of observations about this issue.

Incipient faults Incipient faults, which can be characterized in terms of gradual drift of a fault parameter γ, can be theoretically analysed using the piecewise affine system framework. In other words, using results from [66] we can define a nominal region Ω_{nom} with an adjacent fault region Ω_f such that blending of the controllers for the two regions in guaranteed to maintain stability and optimality of the control.

In this case, we can also show that mixing control will always outperform discrete switched control in terms of response times for fault tolerance. The mixing approach enables smooth compensation of faults as the degree of the fault progresses (i.e., for any $\gamma > \epsilon$), whereas the discrete switched controller makes an abrupt switch from nominal to fault controllers once a fault magnitude threshold $\gamma^* \gg \epsilon$ is exceeded, ensuring a period of sub-optimal control with a fault whose magnitude is in the interval $[\epsilon, \gamma^*]$.

Abrupt faults Abrupt faults, which can be characterized in terms of a significant difference in at least one fault parameter γ, cannot be theoretically analysed using the piecewise affine system framework unless we can guarantee contiguity of nominal region Ω_{nom} and fault region Ω_f. Stability depends on the eigenvalues of the matrix $(A - BK)$ from a rewritten version of the observer equations, namely equation A.5. We leave stability analysis as future work.

We have adopted a probabilistic approach based on residuals. In future work, we plan to compare this approach with those using Kalman filters for hypothesis testing, e.g., [58], which are computationally more complex, but may have different stability properties.

5 Experimental Design

5.1 Software Configuration

We implemented the running example described in section 2 in MATLAB/Simulink and ran all experiments in a simulation environment, where the real system and the model that the controller uses to estimate the system state are identical. We injected faults into the simulated real system by modifying the inputs and outputs of the model. We then computed residuals using the output $\hat{\boldsymbol{y}}$ of the model used by the controllers and the real system model output $\boldsymbol{y}$. This type of simulation environment allows for complete control of the fault injection, detection and controller output analysis.

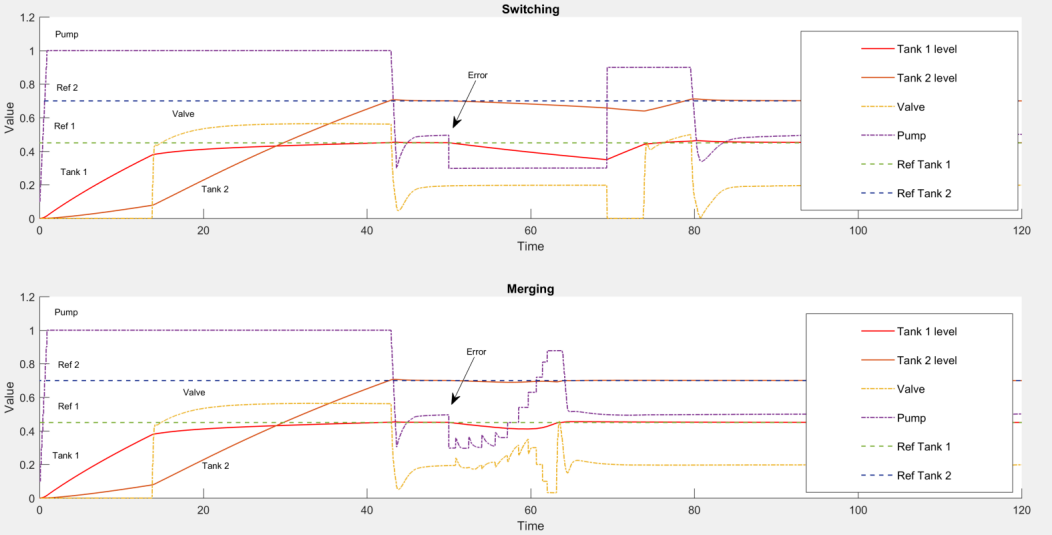


Figure A.3: Comparison of controllers for FTC with Pump fault at $t = 50$s

5.2 System Fault Injection and Detection

We injected faults into the pump and valve using the multiplicative equation (A.6), with pump, valve and tank parameters $\gamma_1,\ \gamma_2,\ \gamma_3 \in [0, 1]$, respectively. γ_1 simulates the pump being blocked by only allowing a fraction of the water flow to go through the pipe. γ_2 simulates valve blockage (i.e., the valve not having full range of motion); we use a Simulink saturation block for this fault. We implemented leaks in both tanks by subtracting a constant value γ_3 of the real system model output for the two tank levels. We generated 4 residuals, for: valve, pump, tank 1 level and tank 2 level.

5.3 Controller Design

We designed three separate MPC controllers to handle different scenarios.

Nominal The first MPC is the nominal plant controller that is designed to handle the no-error state of the system. The plant model was linearised at initial model conditions with no errors present on in the system. The valve and pump output of the controller are both between 0 and 1 representing nominal working conditions.

Pump Fault The second controller is an MPC that has a plant model linearised when the pump was experiencing a fault of 40% reduction in throughput. The valve output range for this controller is also 1 but the pump output has an increased range of 0 to 1.5, since there is a blockage in the pipe the pump should be able to increase water flow to compensate for this. The reference signal coming into the second MPC is the original reference set point for the two tanks plus the

estimated error in the tank levels. The justification for this is that if the error in the tank level and original reference is known then the required reference signal to stabilise the plant at the required set point can be computed.

Valve Fault The third controller is designed for valve fault scenarios, by linearising the model at a point where the valve setting is limited. The reference signal and controller output signal ranges are the same as for the nominal controller, but this controller favours the use of the pump flow over the valve setting to stabilise the plant output.

5.4 Discrete Switching and Merging implementation

We implemented discrete switching and merging approaches on identical systems using the three controllers described in section 5.3, but with different supervisory controllers.

Discrete Switching In the discrete switching implementation, higher level control mechanism analyses the current fault detection output and chooses which controller is the appropriate one for the current fault state. The decision is made by the current maximum error signal. The pump fault controller uses the two water levels in the tanks as the trigger signal. The valve controller uses the valve residual error as the trigger signal. The higher of the two tank error signals is compared to the valve error to decide which controller should get control. Error signals must be over 0.1 to make the system robust against false positive error signals and allow the nominal controller to control the plant under normal conditions.

Merging-Based Switching For the merging mechanism we implemented a look up table of weights, with error signals of the water tank error and valve error used as indexes into it. The weights allow for linear interpolation of the output signal and are defined as follows: {0.0 , 0.1 , 0.2 , ... 0.9, 1}. Every percent in the error signal will index into a higher weight, up to 10%. E.g. 0.01 valve error signal and 0.00 error signal in the tank levels will correspond to 10% of the output signal being generated from the valve error controller, 0% from the pump fault controller and 90% from the nominal controller. If the two weighted signals added together are greater than 1 they are both incrementally reduced until their sum is equal to 1. This allows for the proportion of the signal to be preserved in extreme error cases. This interpolation mechanism is simple and has a corresponding controller merging configuration for every value in the error

space.

6 Experimental Analysis

We ran experiments to compare the impact of faults on FTC designs based on discrete switching and merging approaches. We also examined the impact of time for fault isolation on FTC, since it is typically assumed that faults can be isolated instantly, even though the switching times do not have to be known *a priori*.

6.1 Switching vs. Merging

We ran experiments for single faults (pump/valve/leak) and double faults (pump and valve). Figure A.3 shows the results for inducing a pump fault at $t = 50$s: the merging approach starts modifying the control signal immediately and restores set-point levels in tanks 1 and 2 by $t = 65$s; the switching approach does not perform a discrete switch until $t = 70$s, when the impact of the fault on tank levels exceeds the specified tolerance, and only restores set-point levels in tanks 1 and 2 by $t = 90$s.

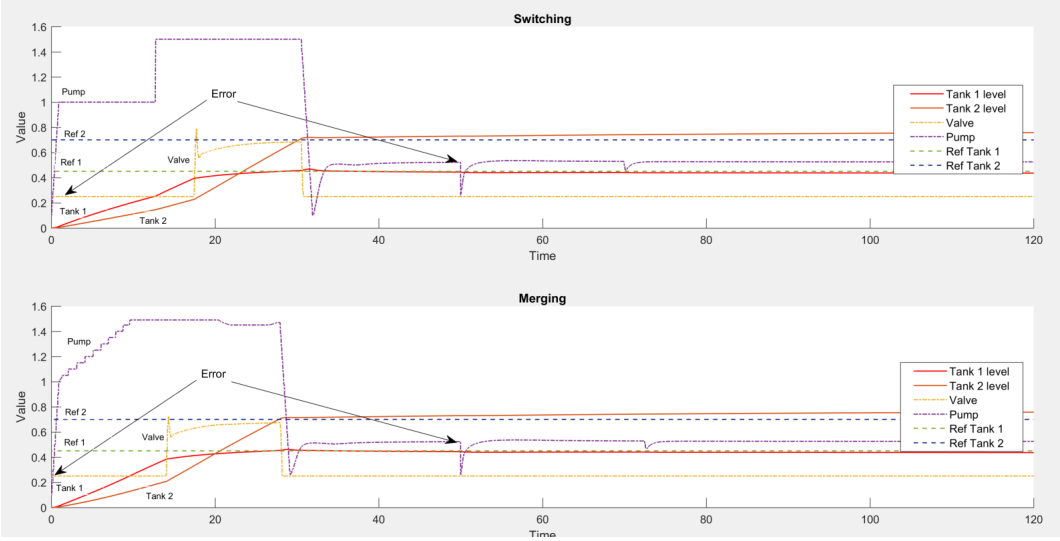


Figure A.4: Comparison of controllers for FTC with Pump and Valve fault

Figure A.4 shows the results for inducing a valve fault at $t = 0$s and a pump fault at $t = 50$s. The merging approach starts modifying the control signal immediately (given the valve fault) and restores set-point levels in tanks 1 and 2 by $t = 30$s; the impact of the pump fault is also quickly corrected.

In contrast, the switching approach does not perform a discrete switch to correct the valve fault until $t = 10$s, when the impact of the fault on tank levels exceeds the

specified tolerance. The pump fault correction is also later than that for the mixing approach.

6.2 Impact of Delays and Fault Isolation Errors

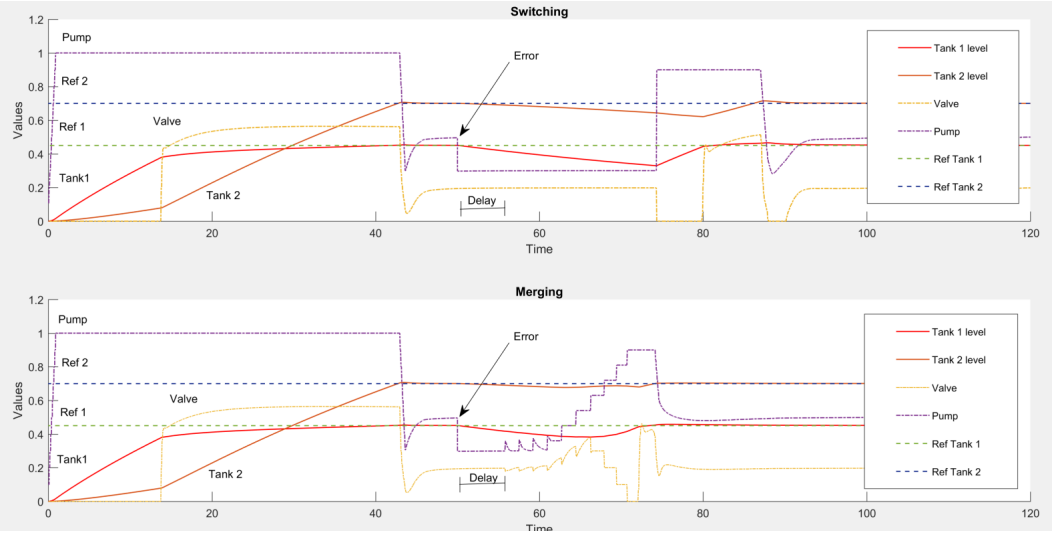


Figure A.5: Impact of delay in FDI on switching and merging controllers for FTC with Pump fault.

Figure A.5 shows the results of a delay of 5s on computing FDI results. This basically delays the possibility of switching and creates a greater impact on the system due to the fault.

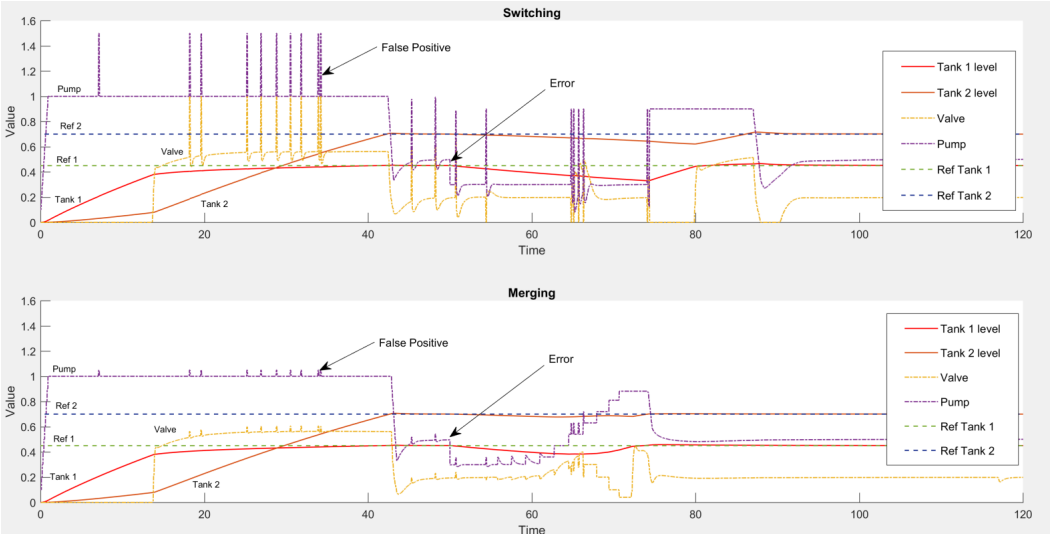


Figure A.6: Impact of inaccuracy in FDI on switching and merging controllers for FTC with Pump fault.

Figure A.6 shows the results of an inaccuracy in correctly isolating the fault on FDI results. Such inaccuracy must be taken into account since FDI is never perfect. These

results have a large impact on the possibility of switching, which creates a significant impact on the system control due to the thrashing of the control. In contrast, the impact on mixing control is much less.

7 Discussion

Our results show a clear difference between discrete switching and mixing supervisory control in the presence of faults. The mixing supervisory control is more robust to faults and to errors in the FDI outputs. Further, the mixing supervisory control responds faster to faults than does the discrete switching approach. This improvement results from the dynamic blending of all controllers, such that even small errors are taken into account in the output signal. The supervisory controller is actively compensating for faults with a blend that is proportional to the faults seen, hence a small fault will be compensated immediately with a small portion of the output coming from the fault controller. Should the error persist the portion of the fault controller will become increasingly larger until it composes the entire output signal. However, in comparison to the switching approach, the change of controller is gradually increasing which means by the time the fault threshold has been reached the system is already moving towards a stable state and will reach this earlier than the switching approach. Even in the case where a big fault occurs which is greater than the threshold, the merging approach will simply act the same as the switching approach and let the dedicated fault controller take 100% of the output signal, so it will always perform better or equal to the switching approach.

We plan to extend this work in several ways. First, we plan to compare the active FTC methods presented with passive robust and adaptive MPC control. Second, we plan to use machine learning to adaptively tune our system to unseen anomalies/faults. We are also investigating more complex application domains, such as multi-copter drones.

Paper B

Fault-Tolerant Control for Unseen Faults using Randomized Methods

Abstract

Real-time tolerance to novel faults is often needed to avoid catastrophic outcomes. However, no real-time solutions currently exist for novel faults: e.g., isolating the underlying fault and implementing an appropriate control may be too time-consuming or require *a priori* fault information. For such circumstances, we propose a novel approach that uses fast randomized control switching to stabilize a system, without any state estimation. We first show theoretically that, for a properly tuned controller, a randomized approach guarantees mean-square stability and is guaranteed to converge to an optimal solution. We then empirically demonstrate the efficacy of this approach on trajectory following in a quadcopter UAV.

1 Introduction

Systems subject to faults typically adopt fault-tolerant control (FTC) methods to ensure that they can accomplish a range of tasks [68]. A typical FTC method first identifies the fault given some anomalous behaviour, and then uses a pre-defined fault controller to ensure task completion [69] by switching control from a nominal to a fault controller.

One drawback to FTC for real-time systems is the computational effort and time required for system (e.g., fault) identification may cause control disruptions and poor system performance. Second, FTC requires *a priori* controller design for all faults $\mathcal{D}$, and hence cannot deal with novel $\mathcal{D}$.

In this article we adopt an approach that addresses these two drawbacks. We model the possible changes to a system using a hybrid systems model Ψ that can operate in a finite set $\Upsilon = \{\upsilon_1, \upsilon_2, \cdots, \upsilon_V\}$ of modes. Ψ can evolve according to a stochastic system switching rule $\sigma(k)$ that captures the random nature of mode switching, e.g., the random onset of faults. We model a controller than can switch control signals in response to system mode changes, using rule $\gamma(k)$. Given the onset of anomalies (computed using a system monitor), *without performing system identification* we perform random control mode switching.

To date, research has focused on stability properties for a stochastic hybrid system Ψ, e.g., [70, 71]. In addition to stability properties, our interest is in the stabilizability and the performance of a randomized switching controller relative to an ideal (reference) controller, given the significant computational advantages of the former. We derive theoretical results for a suitable performance metric, and demonstrate the properties of the randomized approach on a quadcopter. Our results provide theoretically-sound, computationally-simple techniques for controlling a vehicle that is encountering (unforeseen) faults for which either no control laws exists or the controls would take too long to compute before a crash occurs.

Our contributions are as follows:

- We formalize a system subject to random faults in terms of a discrete-time dual switching system;
- we show that a randomized switching controller can stabilize a system given appropriate controller tuning;
- we show that the loss of a randomized switching controller can get arbitrarily close to the minimal loss ξ^* of a reference controller, as time $k \to \infty$;

- we validate our theoretical results using a quadcopter UAV.

2 Related Work

Fault-tolerant control has been addressed using many approaches, including methods based on system structure [25], fuzzy systems [72] or data-driven methods [73]. All these methods assume that the set of faults is known *a priori*, in constrast to the approach we propose.

This paper extends work done in switched control by examining randomized blended approaches. Several authors have examined stability and stabilizability properties of discrete-time dual switching linear systems, e.g., [70, 74, 75]. [70] deal with stability properties of discrete-time dual switching linear systems, with deterministic switching for control inputs. [74] examine stochastic models for both system and control switching, but do not cover issues of relative performance of different controllers. [75] address performance of stochastic control switching, but not for blended control.

3 Notation and Preliminaries

3.1 System Model

Table B.1: Notation

Symbol	Key
$\boldsymbol{x}(k)$	state vector
$\boldsymbol{u}(k)$	control vector
$\boldsymbol{y}(k)$	observation vector
$\sigma(k)$	process switching function
Λ	process probability matrix
$\boldsymbol{\upsilon}(k)$	process mode vector
$\boldsymbol{\alpha}(k)$	process parameter vector
$\gamma(k)$	control switching function
Π	control probability matrix
$\boldsymbol{\varphi}(k)$	control mode blend distribution
$\tilde{\mathcal{T}}_{1:T}$	reference task over time $1:T$
$\xi_{1:T}$	loss function over time $1:T$

Systems that switch between operating modes exhibiting different continuous dynamics are known as switched systems [76]. The transition from one mode to the other is given by switching variables, which evolve according to a switching rule. We can design switching rules to model system mode changes due to stochastic

disturbances [77], or due to control inputs [71]. If we model stochastic switching variables as being driven by an underlying Markov chain, a linear switched system turns into a Markov jump linear system [78].

We formalize our approach using a system Ψ that is subject to both (a) stochastic jumps governed by the process $\sigma(k)$, and (b) switches dictated by the control signal $\gamma(k)$. We define a discrete-time dual switching linear system Ψ, a class of switched linear systems with two (either stochastic or deterministic) switching variables, as follows:

Definition 3 (State-Space model Ψ)**.** *A discrete-time dual switching linear system Ψ is described by*

$$\boldsymbol{x}(k+1) = A_{\sigma(k)}^{\gamma(k)}\boldsymbol{x}(k) + B_{\sigma(k)}^{\gamma(k)}\boldsymbol{u}(k) \tag{B.1}$$

$$\boldsymbol{y}(k+1) = C_{\sigma(k)}^{\gamma(k)}\boldsymbol{x}(k) \tag{B.2}$$

where k is the discrete time index, $\boldsymbol{x}(k) \in \mathbb{R}^n$ is the state, $\boldsymbol{u}(k) \in \mathbb{R}^m$ is a control input, $\boldsymbol{y}(k) \in \mathbb{R}^p$ is the performance output, $\sigma(k)$ and is a time homogeneous Markov process taking values in the set $\Upsilon = \{\upsilon_1, \upsilon_2, \cdots \upsilon_V\}$, with transition probability matrix Λ, $\gamma(k)$ is a switching signal taking values in a finite set $\mathcal{U} = \{\boldsymbol{u}_1, \omega_2, \cdots \boldsymbol{u}_M\}$, with transition probability matrix Π. More precisely, the entry $\lambda_{ij} \geq 0$ of Λ represents the probability of a transition from mode υ_i to mode υ_j, namely $\lambda_{ij} = P\{\sigma(k+1) = \upsilon_j | \sigma(k) = \upsilon_i\}$. Λ is a right-stochastic matrix.

A matrix, e.g., $A_{\sigma(k)}^{\gamma(k)}\boldsymbol{x}(k)$, defines the dynamics for Ψ at time k given system mode $\sigma(k)$ and control mode $\gamma(k)$. If we define $\rho(k)$ as the process probability distribution at time k, then its evolution is governed by the difference equation $\rho(k+1)' = \rho(k)'\Lambda$, where $\rho(0) = \rho_0$. If we further assume that Λ and Π are irreducible and aperiodic Markov matrices, then the Markov process admits a unique stationary (strictly positive) probability distribution; for example, the distribution $\bar{\rho}$ satisfies $\bar{\rho}' = \bar{\rho}'\Lambda$. The state dynamics of the overall system Ψ is characterized by a set of tuples $(A_i^j, B_i^j, C_i^j),\ i \in \Upsilon,\ j \in \mathcal{U}$.

Figure B.1 depicts a schematic for our approach. On the left we see the controller switching, where the supervisory controller (via a randomised switching function $\gamma(k)$) generates a distribution $\boldsymbol{\varphi}(k)$. On the right the system switching function $\sigma(k)$ randomly selects a plant model Ψ_i corresponding to mode υ_i.

We characterise the evolution of Ψ not just using a sequence of discrete modes, but quantify the magnitude of each mode using a parameter α. We define the mode vector $\boldsymbol{\upsilon}(k) = \{\upsilon_1(k), \cdots, \upsilon_L(k)\}$ the set of modes defining Ψ at time k. Associated with

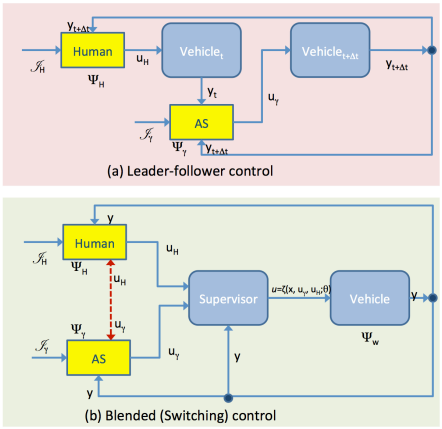


Figure B.1: System and controller framework

the mode vector is the parameter vector $\boldsymbol{\alpha}(k) = \{\alpha_1(k), \cdots, \alpha_L(k)\}$. For example, for systems with fault-modes the mode parameter $\alpha(k)$ can capture the degree of degradation of a fault. Table B.1 summarizes our notation. We denote a sequence $\{Z(1), \cdots, Z(K)\}$ using $\{Z(k)\}_{k\in\mathbb{N}}$. In our notation we denote vectors by bold-faced symbols, matrices by capitalised symbols, and temporal indices by k.

3.2 Running Example

We use as our running example a quadcopter. As shown in Fig. B.2, motor M_i ($i =$

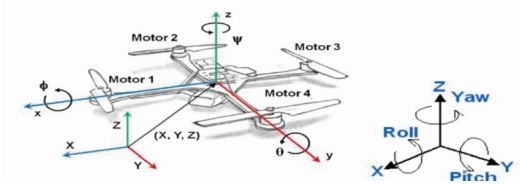


Figure B.2: Schematic of quadcopter

$1, 2, 3, 4$) produces thrust force in rotor i, which can be modulated by the control system along the vertical body axis; see, e.g. [2], for full quadcopter system details.

We define a linear model for the quadcopter in terms of Ψ (Definition 3), where[1]

- the state variables in $\boldsymbol{x}$ include 3D positions of the quadcopter in the inertial frame, x_I, y_I and z_I respectively, and Euler angles for roll (ϕ), pitch (θ) and yaw (ψ), together with their derivatives;
- the control vector $\boldsymbol{u} = [f_1 \ f_2 \ f_3 \ f_4]$ denotes the lifting forces for each rotor ($i = 1, ..., 4$);

[1] We suppress time indices for ease of exposition.

- the observation vector is $\boldsymbol{y} = [\phi\ \theta\ \psi\ x_I\ y_I\ z_I]$.

For simplicity, we assume that the state model Ψ can switch randomly from nominal to fault states: i.e., $\Upsilon = \{\upsilon_n,\ \upsilon_f\}$. The nominal state υ_n assumes fault-free operation, while the fault state υ_f encompasses actuator faults that can occur in any combination of rotors. The fault degree for rotor fault-mode υ_f is in the range [0,1], such that 0 denotes failed and 1 totally healthy. For example, transition matrix Λ below defines faults occurring with probability 0.01, and faults being permanent once they occur.

$$\Lambda = \upsilon_n(k+1)\upsilon_f(k+1)\upsilon_n(k)0.990.01\upsilon_f(k)01$$

In our control model, we assume that we tune 5 possible control modes: one for nominal operation ($\boldsymbol{u}_0$), and one for rotor fault, denoted $\boldsymbol{u}_i$ for $i,\ i = 1, 2, 3, 4$. A discrete switching controller can thus switch among these 5 possible control modes. In the simplest case, a uniform distribution for control switching defines a switching matrix Π is a 5×5 matrix with all entries of 0.2.

3.3 Performance Measures

This section describes our task and how a controller measures its performance on that task.

Definition 4 (Task)**.** *A task $\mathcal{T}_{1:T}$ is a sequence of observed world states from time 1 to T: $\mathcal{T}_{1:T} = \{\boldsymbol{y}_1, , \cdots, \boldsymbol{y}_T\}$.*

We focus in this article on controlling the system to execute a reference task, $\tilde{\mathcal{T}}_{1:T}$, by executing a sequence of controls $\{\boldsymbol{u}\}_{k\in\mathbb{N}}$. The control performance is measured by a loss function, defined below.

Definition 5 (Loss Function)**.** *A loss function $\xi(\boldsymbol{u}, \boldsymbol{\alpha}) : \boldsymbol{u}(k) \times \boldsymbol{\alpha}(k) \to [0, 1]$ assigns a loss $\in [0, 1]$ to a control $\boldsymbol{u}(k)$ executed given a state characterised by $\boldsymbol{\alpha}(k)$.*

We define the cumulative loss of an executed task $\mathcal{T}_{1:T}$ as

$$\xi^{\boldsymbol{u}}_{1:T} = \sum_{i=1}^{T} \xi\left(\boldsymbol{u}(i), \boldsymbol{\alpha}(i)\right) \text{ for } \boldsymbol{u} \in \mathcal{U}. \tag{B.3}$$

Given this loss function, we can characterize a notion of relative loss as follows.

Definition 6 (Relative Loss)**.** *Relative loss is the difference between the loss $\xi^{\boldsymbol{u}}_{1:T}$ of a controller sequence $\boldsymbol{u}_{1:T}$ and the loss $\xi^{\boldsymbol{u}^*}_{1:T}$ of an ideal (reference) controller sequence*

$\boldsymbol{u}^*_{1:T}$:

$$\Delta(\boldsymbol{u}_{1:T}, \boldsymbol{u}^*_{1:T}) = \| \xi^{\boldsymbol{u}}_{1:T} - \xi^{\boldsymbol{u}^*}_{1:T} \| . \tag{B.4}$$

Definition 7 (Task Optimisation)**.** *Given a reference task $\tilde{\mathcal{T}}_{1:T}$ and initial conditions $\boldsymbol{x}_0$, compute a control sequence $\boldsymbol{u}_{1:T}$ that minimizes relative loss of $\boldsymbol{u}_{1:T}$.*

Example: Quadcopter Trajectory Optimisation Task We illustrate loss functions using a trajectory optimization task. A trajectory is a temporally-indexed set of coordinates in 2D or 3D, denoted $\boldsymbol{\zeta}(k)$. We denote the reference (desired) trajectory as $\boldsymbol{\zeta}^*(k)$, and the executed trajectory as $\tilde{\boldsymbol{\zeta}}(k)$, with corresponding control sequences $\boldsymbol{u}^*_{1:T}$ and $\tilde{\boldsymbol{u}}_{1:T}$, respectively.

Definition 8 (Trajectory Relative Loss)**.** *We can represent the relative loss of trajectory following as a mean-square difference between reference and executed trajectories, i.e., $\Delta(\tilde{\boldsymbol{u}}_{1:T}, \boldsymbol{u}^*_{1:T}) = \sum_{k=0}^{T} \| \boldsymbol{\zeta}^*(k) - \tilde{\boldsymbol{\zeta}}(k) \|^2$ for a trajectory over time points $k = 0, \cdots, T$.*

We formulate an optimization version of the trajectory following problem, where we seek to find the control sequence $\boldsymbol{u}^*_{1:T}$ that minimizes the trajectory loss over time points $k = 0, \cdots, T$: $\boldsymbol{u}^*_{1:T} = _{\boldsymbol{u} \in \mathcal{U}} \Delta(\boldsymbol{u}_{0:T}, \boldsymbol{u}^*_{0:T})$. Figure B.3 shows an example of a trajectory optimization task for a quadcopter, where we must follow the diamond-shaped trajectory shown.

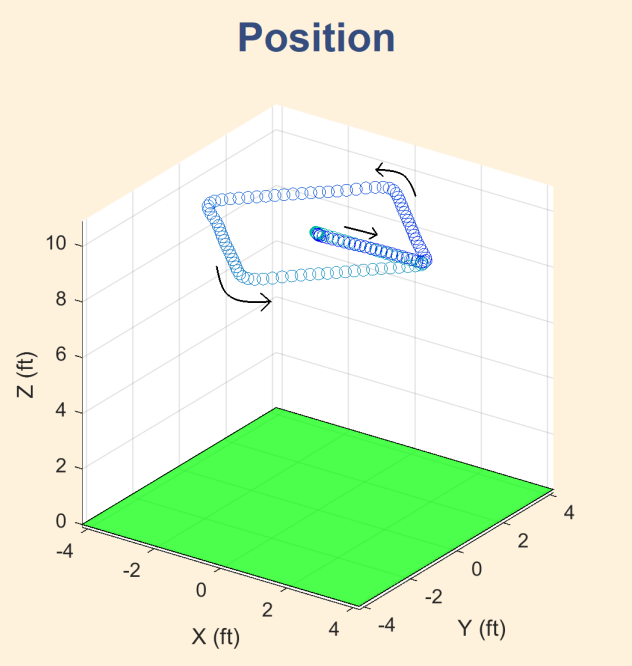


Figure B.3: Trajectory for quadcopter

3.4 Blended Controller Models

This section describes the details of our switching control models. We first describe blended (or mixing) control [51], and then extend this to randomized blended switching. Blended control is an adaptive control method that creates smooth controller switching (due to switching among proportional controller inputs rather than discrete controllers) and adapts well to uncertainty in model parameters [79]. However, the design of the blend function is complex, and the switching function requires as input the current state estimate. For example, [2] describes a blended control approach for quadcopter fault-tolerant design.

Blended Control: We assume that we have a controller that is tuned to the mean parameter for each system mode, i.e., given M control modes we have M tuned controllers. We use a blended control approach to be able to accommodate the continuous-valued parameter space.

Definition 9 (Blended Control)**.** *Given a collection of controllers $\mathcal{U} = \{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m\}$ and a control blend distribution vector $\boldsymbol{\varphi}$, a blended control is a weighted combination $\vec{\boldsymbol{u}} = \sum_i \varphi_i \boldsymbol{u}_i$ such that: (1) $\forall i$, $\boldsymbol{u}_i \in \mathcal{U}$, $0 \leq \varphi_i \leq 1$, and (2) $\sum_i \varphi_i = 1$.*

Without loss of generality, we can characterise a blended controller solely in terms of the weight vector $\boldsymbol{\varphi} = \{\varphi_1, \cdots, \varphi_m\}$, given a fixed set of controllers $\mathcal{U}$.

Example 1. *In our quadcopter we assume that each rotor can be faulty, resulting in a loss of effectiveness of that rotor, which we parameterize using $\alpha_i \in [0, 1]$ for rotor i, where 0 denotes disabled and 1 denotes fully functional. The vector $\boldsymbol{\alpha} = \{\alpha_1, \cdots, \alpha_4\}$ specifies the fault parameter vector. Assume that rotors 1 and 2 both have fault parameter 0.95, and the other rotors are normal. For control vector $\boldsymbol{u} = \{u_0, \cdots, u_4\}$, where u_0 is the normal controller and $u_i < \ i = 1, ..., 4$ is the fault controller, we can apply a blend vector $\boldsymbol{\varphi} = \{\varphi_0, \cdots, \varphi_4\} = \{0.6, 0.2, 0.2, 0, 0\}$, i.e., we stabilise the quadcopter using a combination of the normal controller with fault controllers for rotors 1 and 2.*

A randomized switching function $\gamma(k)$ switches control at each discrete step in a random fashion. For blended control, the transition probability matrix is a 3-dimensional matrix Π, whose elements are denoted by $\varrho_{ijl} \triangleq P(\varphi_l(k+1) = u_j | \varphi_l(k) = u_i)$, which is the probability of a transition from the l^{th} element of control mode u_i to the l^{th} element of control mode u_j. Here, $\varphi(k)$ is a probability distribution of blended control mode assignment at time step k.

A *blended control sequence* corresponding to a task $\mathcal{T}_{1:T}$ can be represented as

$\{\boldsymbol{\varphi}\}_{1:T} = \{\boldsymbol{\varphi}(1), \cdots, \boldsymbol{\varphi}(T)\}$ since the controller vector $\boldsymbol{u}$ is fixed. A reference control sequence is a loss-minimal control sequence for executing a task $\mathcal{T}_{1:T}$.

Definition 10 (Reference Blended Control Sequence)**.** *Given a collection of controllers $\mathcal{U} = \{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m\}$ and a control blend distribution vector $\boldsymbol{\varphi}$, a reference control sequence is a loss-minimal control sequence for $\mathcal{T}_{1:T}$:*

$$\{\tilde{\boldsymbol{\varphi}}\}_{1:T} =_{\boldsymbol{\varphi}_{1:T}} (\xi_{1:T}) \tag{B.5}$$

4 Theoretical Results

This section summarises our theoretical results. We prove two classes of result: (1) "standard" control theory results concerning the stability and stabilizability of a randomized controller; and (2) results concerning how well a randomized controller performs a tasks with respect to a reference controller. We summarise our results as follows:

A. *Stability*: given a system Ψ subject to stochastic fault, we show that $\exists$ system matrices (A and B of Definition 3) such that Ψ converges to a stable state, which we define in terms of mean-square stability (MSS) [Sec. 4.1].

B. *Stabilizability*: given a system Ψ subject to stochastic fault, $\exists$ a switching function γ that generates a control sequence $\{\gamma\}_{k\in\mathbb{N}}$ that guarantees MSS [Sec. 4.2].

C. *Performance given known modes* Υ: we can synthesize a randomized blended control sequence whose performance converges to the performance of a reference controller [Sec. 4.3].

D. *Performance given unknown modes* Υ': we can synthesize a randomized blended control sequence whose performance converges to the performance of a reference controller if the worst-case loss is bounded for any unknown mode $\breve{\upsilon} \in \Upsilon'$ [Sec. 4.3].

4.1 Stability Properties

Assessing stability of system Ψ given switching due to system fault process $\{\sigma(k)\}_{k\in\mathbb{N}}$ is a robust stability problem. The standard approach defines stochastic stability, in

terms of mean-square stability (MSS), which states that the expectation of the system state norm asymptotically converges to zero.

Definition 11 (Mean Square Stability)**.** *A system Ψ exhibits mean square stability (MSS) if for any initial condition $(\boldsymbol{x}_0, \upsilon_0)$, and any switching sequence $(\sigma_k)_{k\in\mathbb{N}}$,*

$$\lim_{k\to\infty} E(\| \boldsymbol{x}(k) \|^2) \to 0. \tag{B.6}$$

We first consider a system with no control inputs, i.e., we can represent Ψ as $\boldsymbol{x}(k+1) = A_{\sigma(k)}\boldsymbol{x}(k)$. Using this definition, we can state the conditions for stability in terms of the system matrix $A_{\sigma(k)}$, independent of the control switching process, using a result from [78].

Lemma 4.1 (Mean Square Stability)**.** *A system Ψ exhibits mean square stability (MSS) if $\exists$ a system matrix $A_{\sigma(k)}$ such that for any initial condition $(\boldsymbol{x}_0, \upsilon_0)$, and any switching sequence $\{\sigma(k)\}_{k\in\mathbb{N}}$, equation B.6 holds.*

4.2 Stabilizability Properties

Assessing stability of system Ψ given switching due to control inputs $\{\gamma(k)\}_{k\in\mathbb{N}}$ is a stabilization problem and requires assessing the properties of a switching control law. Lemma 4.2 states that one can design Ψ such that MSS holds.

Lemma 4.2 (Mean Square Stabilizability)**.** *A system Ψ (Definition 3) is mean square stabilizable if there exists a switching control law*

$\{\gamma(k)\}_{k\in\mathbb{N}}$ defined as a time homogeneous Markov chain such that, for any initial condition $(\boldsymbol{x}_0, \gamma_0)$, equation B.6 holds.[2]

4.3 Performance Computation

This section provides a theoretical basis for assessing the performance of randomised switching, with regard to an idealized controller, or reference controller. We will show comparative performance for two cases: (1) known modes, and (2) unknown modes.

4.4 Performance: Known Mode Set Υ

This section shows results for relative performance of reference and randomized controllers when the system mode switching covers a set Υ of known system modes.

[2] Proofs for Lemmas 2, 3 and 4 are in the Appendix.

When the system modes are known and appropriate controllers are tuned for these modes, we can show that the loss of a randomized control system converges (in expectation) to that of a reference control system.

Lemma 4.3. *$\exists$ a system Ψ (Definition 3), a randomised controller sequence $\{\tilde{\boldsymbol{\varphi}}(k)\}_{k\in\mathbb{N}}$, and a reference controller sequence $\{\boldsymbol{\varphi}^*(k)\}_{k\in\mathbb{N}}$ such that mean square convergence (MSC) of $\{\boldsymbol{\varphi}(\breve{k})\}_{k\in\mathbb{N}}$ and $\{\boldsymbol{\varphi}(k)\}_{k\in\mathbb{N}}$ as $k \to \infty$ is guaranteed for any known mode $\upsilon \in \Upsilon$.*

4.5 Performance: Unknown Mode Set

In the case when the system modes $\breve{\upsilon} \in \Upsilon$ may be unknown, we must introduce a notion of bounded loss to provide reasonable guarantees for any controller. By this we ensure that, for any controller, the instantaneous loss for any unknown system mode $\breve{\upsilon} \in \Upsilon$ does not cause total failure of the system Ψ. We thus add an extra task constraint to Definition 7, i.e., a bound ϵ on the instantaneous loss that corresponds to a system crash, e.g., due to a fault for which no recovery action is possible.

Definition 12. *Bounded instantaneous loss is a loss such that: $\xi(\boldsymbol{u}(k), \boldsymbol{\alpha}(k)) < \epsilon$ for $k = 1, ..., T$ and $\epsilon \in [0, 1]$.*

Given this additional constraint, we can prove the following result using Lemma 4.3:

Lemma 4.4 (Unknown Mode MSC)**.** *$\exists$ a system Ψ (Definition 3), a randomised controller sequence $\{\tilde{\boldsymbol{\varphi}}(k)\}_{k\in\mathbb{N}}$, and a reference controller sequence $\{\boldsymbol{\varphi}^*(k)\}_{k\in\mathbb{N}}$ such that mean square convergence (MSC) of $\{\boldsymbol{\varphi}(\breve{k})\}_{k\in\mathbb{N}}$ and $\{\boldsymbol{\varphi}(k)\}_{k\in\mathbb{N}}$ as $k \to \infty$ is guaranteed if the worst-case loss is bounded for any unknown mode $\breve{\upsilon} \in \Upsilon'$.*

5 Empirical Analysis

This section summarizes results of an empirical study of randomized blended control (RBC) using a quadcopter on a trajectory following task. We focus on two topics, namely: (1) stability and stabilizability given faults; and (2) performance relative to reference controller C_R. We used an ideal (perfect) controller as the reference; to make this controller more realistic, we studied its performance when varying delays are introduced to mimic inference-time for fault isolation, which we encode as a delay τ. *Relative loss* is the difference in flight-path deviation between the actual and reference paths.

In our experiments, we induce a rotor fault at $t = 15$s in the quadcopter, and then

examine the ability of our controllers to recover from the fault and to maintain a stable trajectory. We conducted experiments for rotor fault degrees of υ = 0.98 and 0.91 with similar results. Figure B.4 (c-f) shows the results for $\upsilon = 0.91$ and delay values τ. On the left side of each graphic is a plot of the flight path of the quadcopter and on the right plots for *Error Rate of Change* ϖ (yellow), *Current Tracking Error* ρ(red) and *Current Blend* φ (blue) against simulation time. These figures show how a fault causes a deviation in the flight path, which is reflected in ϖ and ρ, as well as the impact of varying τ (delay due to fault isolation).

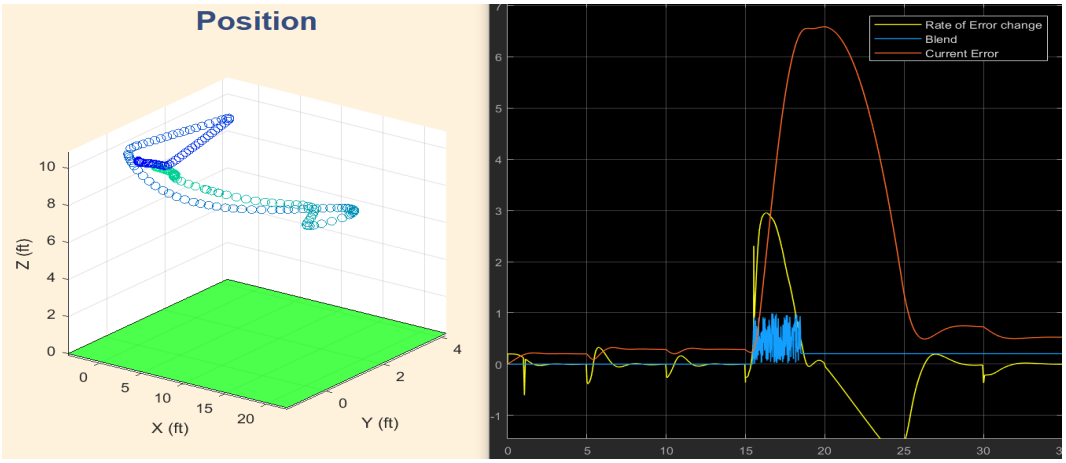
We examined two types of RBC: fault-based RBC (Figure B.4 (a)), where RBC is invoked when the deviation exceeds a threshold (i.e., we stabilize a fault for $t = 15 - 19$); and clock-based RBC (Figure B.4 (b)), where RBC is performed throughout, switching at each tick of a clock with period κ ($\kappa = 0.5$s here). The smaller the value of κ the better is the clock-based RBC. For large faults (as in this example), the clock-based RBC recovers faster and with smaller deviation than does fault-based RBC.

Our benchmark C_R with no delay ($\tau = 0$) performed the best (by definition). Figures B.4 (c-f) show that flight deviations increase with τ, with a delay of $\tau = 2s$ causing a crash before the fault controller can be applied. The randomized architectures both performed extremely well, having a deviation similar to C_R with $\tau = 0.5s$, and only 30% worse than C_R with $\tau = 0$. This indicates that RBC is competitive with state-of-the-art controllers, as one can trade off FDI inference time with fast switching times in RBC.

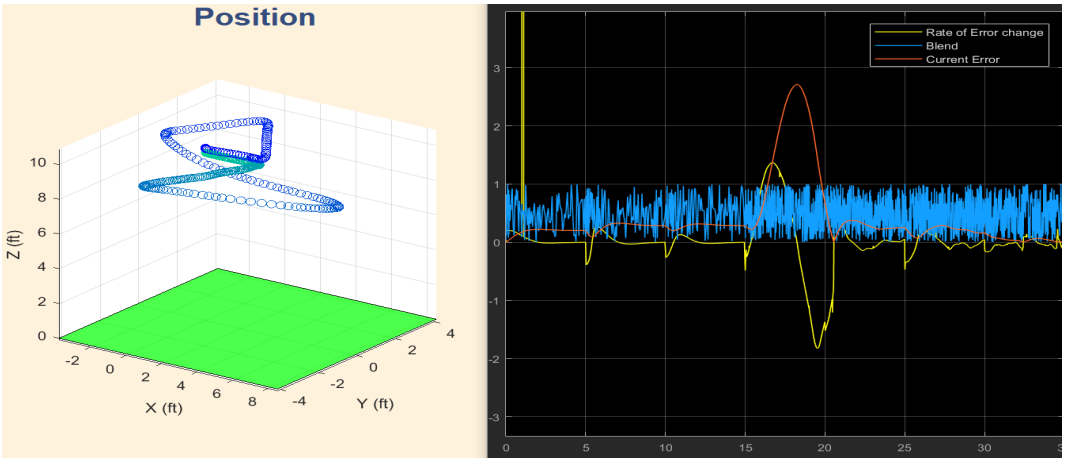
6 Conclusions

This article has shown that randomized blended control is a theoretically sound approach for stabilizing a system that is encountering novel faults for which no control laws are known. More precisely, we have shown that there exists a collection $\mathcal{U}$ of controllers that can be randomly blended to recover from a novel fault $\mathcal{D}$, without identifying the state description of $\mathcal{D}$. This result extends robust control, which guarantees stability for $\mathcal{D}$ that is strictly bounded, or other control methods (e.g., adaptive, sliding-mode) for which $\mathcal{D}$ must be known *a priori*. Our results generalize naturally to environment disturbances other that system faults.

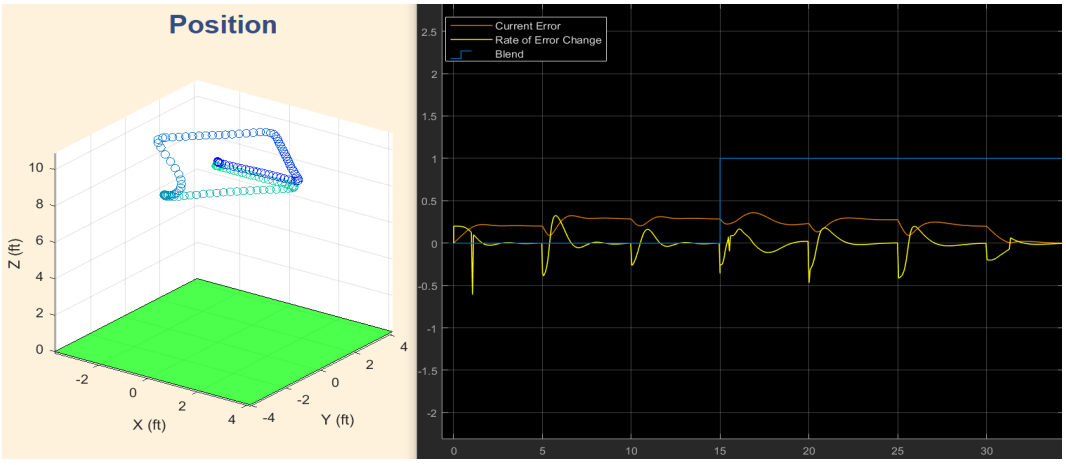
We have demonstrated empirically that: (a) the randomized approach can be used recover from unanticipated faults without fault isolation; and (b) the randomized approach can perform trajectory following in the presence of disturbances with small loss of optimality compared to a reference controller.



(a) Fault-based Randomized Blended Control

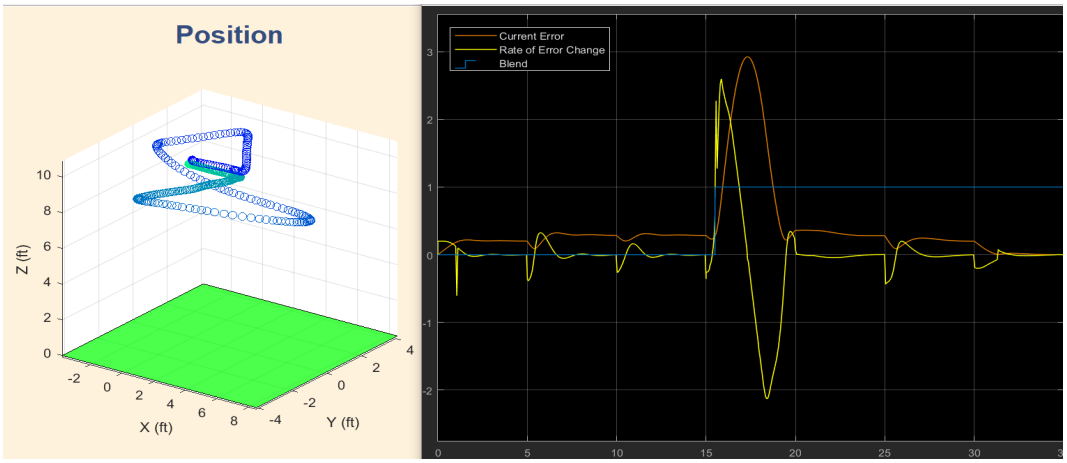


(b) Clock-based Randomized Blended Control

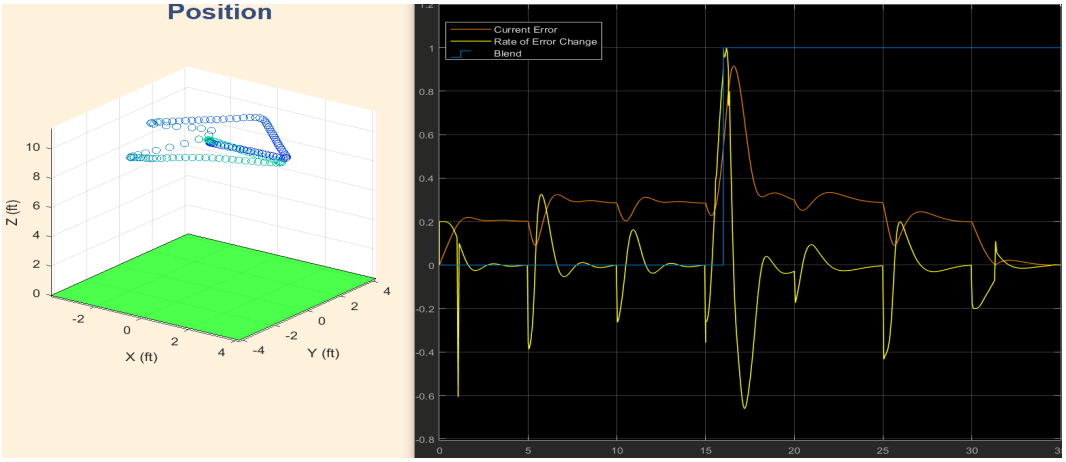


(c) C_R with $\tau = 0s$

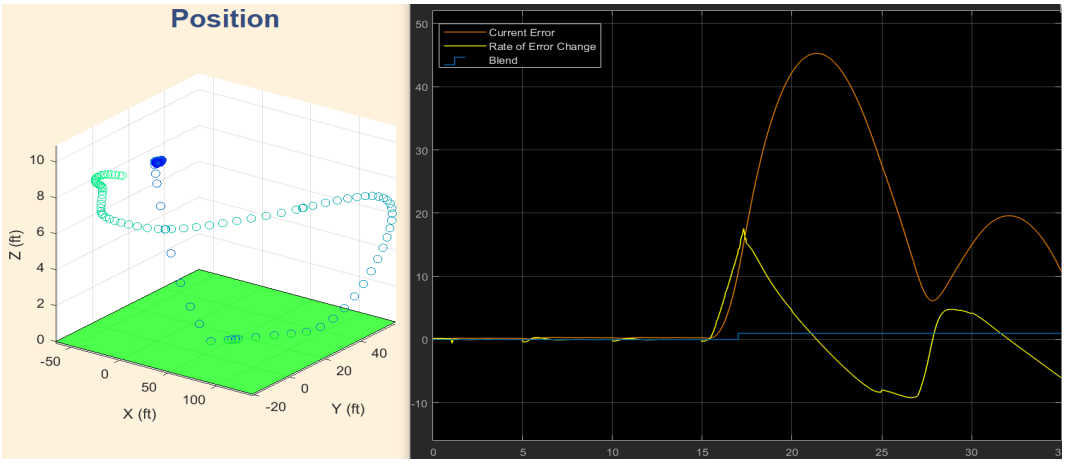
Figure B.4: $\upsilon = 0.91$ at $t = 15s$, a) Randomized fault-based Blending, b) Randomized clock-based, c)-f) Optimal blend C_R delayed by varying τ



(a) C_R with $\tau = 0.5s$



(b) C_R with $\tau = 1s$



(c) C_R with $\tau = 2s$

Figure B.5: $\upsilon = 0.91$ at $t = 15s$, a) Randomized fault-based Blending, b) Randomized clock-based, c)-f) Optimal blend C_R delayed by varying τ

7 Appendix

This section describes key proofs, all of which cannot be included due to space constraints.

Lemma 4.2 (Proof): We rewrite the state equation of Definition 3 such that $\boldsymbol{u}(k) = K^{\gamma(k)}_{\sigma(k)}\boldsymbol{x}(k)$, and defining the matrix $\tilde{A}^{\gamma(k)}_{\sigma(k)} = A^{\gamma(k)}_{\sigma(k)} + B^{\gamma(k)}_{\sigma(k)}K^{\gamma(k)}_{\sigma(k)}$. This enables us to rewrite the state equation of Definition 3 as $\boldsymbol{x}(k+1) = \left[A^{\gamma(k)}_{\sigma(k)} + B^{\gamma(k)}_{\sigma(k)}K^{\gamma(k)}_{\sigma(k)}\right]\boldsymbol{x}(k) = \tilde{A}^{\gamma(k)}_{\sigma(k)}\boldsymbol{x}(k)$. □

Performance Given Known Modes

Our supporting Lemmas are as follows: We first characterize the relative loss of two randomised switching sequences:

Lemma 7.1. *The relative loss $\Delta(\tilde{\varphi}_{1:T}, \varphi_{1:T})$ for two randomised switching sequences, $\varphi_{1:T}$ and $\tilde{\varphi}_{1:T}$, can be expressed such that $\Delta(\tilde{\varphi}_{1:T}, \varphi_{1:T}) \propto W^2(\tilde{\varphi}_{1:T}, \varphi_{1:T})$, using a W^2 measure as per Definition 2 of [75].*

Lemma 7.2 then shows that the relative loss for randomised control is equivalent to the distance W^2 between the reference and randomised distribution sequences.

Lemma 7.2. *For a task $\mathcal{T}_{1:T}$ and randomised and reference distribution sequences, $\{\tilde{\boldsymbol{\varphi}}(k)\}_{k\in\mathbb{N}}$ and $\{\boldsymbol{\varphi}(k)\}_{k\in\mathbb{N}}$ respectively, the relative loss is given by $\lim_{k\to\infty} W^2(\tilde{\varphi}_{1:T}, \varphi_{1:T}) = 0$.*

Proof: From Lemma 7.1, we can express the relative loss for $\{\tilde{\boldsymbol{\varphi}}(k)\}_{k\in\mathbb{N}}$ and $\{\boldsymbol{\varphi}(k)\}_{k\in\mathbb{N}}$ as $W^2(\tilde{\varphi}_{1:T}, \varphi_{1:T})$. We can view the reference controller as a Dirac delta distribution at each $k \in \mathbb{N}$, since its value is a single control signal. Since we are computing the W^2-distance between a Dirac delta distribution and a distribution, we know (by Proposition 1 of [75]), that $\lim_{k\to\infty} W^2(\tilde{\varphi}_{1:k}, \varphi_{1:k}) = 0$. □

We use the next lemma to show the necessary and sufficient conditions for mean square convergence (MSC) given a randomised controller.

Lemma 7.3. *Given a system Ψ and system switching sequence $\{\sigma(k)\}_{k\in\mathbb{N}}$ with switching probability $\pi(k)$, Ψ is MS-stable if and only if the matrix $A^j_{\sigma(i)}$ at time step i (given controller switching mode i) satisfies*

$$\lim_{k\to\infty}\left[\prod_{i=k}^{1}\left(\sum_{j=1}^{m}\pi_j(i)(A^j_{\sigma(i)}\otimes A^j_{\sigma(i)}),\right)\right]\to 0.$$

Proof: We start by using a result (Theorem 1) from [75], which provides the following

identity for W^2:

$$W^2(k) = [I_n]^T \Gamma(k)[\hat{\mu}(0)\hat{\mu}(0)^T + \hat{\Sigma(0)}, \tag{B.7}$$

$$\text{where} \quad \Gamma(k) = \left[\prod_{i=k}^{1}\left(\sum_{j=1}^{m}\pi_j(i)(A^j_{\sigma(i)} \otimes A^j_{\sigma(i)})\right)\right],$$

I_n is the $n \times n$ identity matrix, and $\hat{\mu}(0)$ and $\hat{\Sigma(0)}$ are the mean and the covariance of the distribution of the pdf $\pi(k)$.

We now must justify the necessary and sufficient properties of our claim.

$\Rightarrow$: If $\lim_{k\to\infty}\Gamma(k) = 0$, this implies that $W^2(k) \to 0$ as $k \to \infty$, which in turn implies that $W \to 0$.

$\Leftarrow$: We prove this part of the claim by contradiction. Suppose that $\lim_{k\to\infty}\Gamma(k) \neq 0$. Then the distance W never reaches 0 (by equation B.7), which then implies that MS stability does not hold. Hence we have a contradiction. □

Lemma 4.3 (Proof): This Lemma follows from Lemmas 7.1, 7.2 and 7.3. If we assume that a reference controller guarantees convergence, then Lemma 7.2 states that the randomized controller is MSC, and Lemma 7.3 shows that the system Ψ must be definable in terms of the matrix A. □

Performance Given Unknown Modes

Lemma 4.4 (Proof): We start by assuming a bound on worst-case loss; this guarantees that the disturbance will not cause even a reference controller to be unable to stabilize the system. If the reference controller guarantees stabilizability, then by Lemma 7.3 the randomized controller will also guarantee stabilizability. Further, by Lemma 4.3 we know there must exist a system (defined in terms of matrix A) such that MSC is guaranteed. □

Paper C

Online Reinforcement Learning for Trajectory Following with Unknown Faults

Abstract

Reinforcement learning (RL) is a key method for providing robots with appropriate control algorithms. Controller blending is a technique for combining the control output of several controllers. In this article we use on-line RL to learn an optimal blending of controllers for novel faults. Since one cannot anticipate all possible fault states, which are exponential in the number of possible faults, we instead apply learning on the *effects* the faults have on the system. We use a quadcopter path-following simulation in the presence of unknown rotor actuator faults for which the system has not been tuned. We empirically demonstrate the effectiveness of our novel on-line learning framework on a quadcopter trajectory following task with unknown faults, even after a small number of learning cycles. The authors are not aware of any other use of on-line RL for fault tolerant control under unknown faults.

1 Introduction

One of the most important uses of reinforcement learning (RL) is for controlling robots, using reinforcement learning control (RLC). It has been shown, e.g., [80] that RLC can provide a model-free method for learning control of a robot, e.g., a quadrotor.

Model-free RLC assumes that the control system starts with no model and solves the Bellman equation based on running experiments with appropriate rewards, to create a matrix of values that serves as the model. Although model-free RLC can prove accurate, its main disadvantage is a long convergence time. Since the policy space for robotic interactions can be extremely large, RLC requires a large number of iterations to achieve convergence. In addition, if the plant is unstable (as is that of a quadcopter), or safety is an issue, using RLC can prove difficult in practice.

The alternative approach is to use model-based RLC, which is also known as iterative learning control (ILC). ILC refines the reference or input signals of a desired maneuver based on data from previous executions. This can be used to update model parameters or extend an existing model.

We assume that we have modelled the quadcopter with a linear approximation of the underlying non-linear flight dynamics. Further, we assume that unmodeled dynamics (in our case, faults) can be represented as linear multiplicative term to the actuation dynamics. Given planned trajectory inputs $\boldsymbol{u}_k$, each ILC iteration can be decomposed into two steps: (1) disturbance estimation, where a Kalman filter computes the current estimate of the disturbance ; and (2) input update,where we compute an improved quadrotor input, $\boldsymbol{u}_{k+1}$. Using this framework, the input can be abstracted at any level; e.g., from robot thrust and angular velocities [81] to the position commands for trajectory following [82, 83, 84]. Experiments show that few iterations (on the order of 10-20) are needed to characterize repeatable disturbances and improve tracking performance.

The majority of applications of RL assume that learning is done off-line. However, there are many situations in which a robot will encounter novel situations and needs to adapt to those situations. We address that scenario in this article. In particular, we examine a quadcopter that has been pre-programmed with a set of controls and uses control blending to operate within a known control environment. However, for novel scenarios the robot must adapt. We introduce novel actuator faults into a quadcopter, and use ILC to learn new control laws.

We have defined the quadcopter to have a hierarchical control architecture. The lower-level controllers use PID methods to control each of the quadcopters three axis of

movement. The higher-level controller uses blends of the lower-level controllers to control flight trajectories. We have pre-defined a set of control laws for nominal and fault modes. We then subject the quadcopter to unseen fault and then allow the quadcopter to repeat the novel conditions for the unseen faults to learn new high-level control laws.

This article proposes the first use of ILC for learning novel fault-tolerant control laws. We empirically demonstrate that we can learn new controls from a small number of learning trials.

2 Related Work

This work builds on extensive prior work in RL, fault-tolerant control (FTC) and Fault Detection and Isolation (FDI).

A significant body of work exists for RL for robotics applications, dating from [80]. This work is related to work on trajectory following, e.g., [82, 83, 84]. For this class of application, several instances of learning quadcopter control have been achieved [85]; however we are not aware of prior work that uses Reinforcement Learning to learn the optimal blending of controllers and achieve fault tolerant control.

In the area of FTC [25], a significant body of work has been developed and applied to real-world systems. [49] presents a recent overview of FTC, and [50] presents FTC with relation of system safety. Traditional methods for FTC employ a bank of observers coupled with dedicated controllers, and perform discrete switching. This approach enables designers to tune the system to dedicated faults, but the speed of the system hinges on the speed of FDI. More recent approaches use mixing controllers, e.g., [51, 52], which blend the outputs of multiple controllers and are less reliant on FDI.

Fault-Tolerant Control (FTC) can be divided into two types, passive and active. Passive fault tolerant controllers are designed off-line against predefined models for certain operating conditions and have no ability to react to unanticipated faults. Passive FTC enables fast adaptation to faults, within the predefined operating conditions. Active FTC uses on-line data to reconfigure the controller to stabilise the plant. For a comprehensive study between the two approaches see [46]. Both active and passive FTC rely on specifying the space of faults that the system will encounter. For passive FTC, approaches such as blending of controllers tuned to nominal and failure modes are used to maintain system stability, e.g., [2]. Analogously, active FTC can rely on

being able to detect pre-specified faults, such as using a bank of observers, with each observer tuned to a particular fault, e.g. [47]. For complex systems, it is impossible to pre-specify all faults, since there are too many fault combinations to consider, and it may be impossible to know all possible faults a priori. As a consequence, it is imperative that a system designer understand the space of possible faults and their impact on a system. Very little work has been conducted on exploring the space of faults and their impact on active vs passive FTC.

3 Reinforcement Learning

Reinforcement learning (RL) [11] is a technique for learning control actions that are optimal for particular states, using interactions of an agent with the environment in which the agent obtains rewards for actions. Reinforcement learning is typcially formalized as a Markov decision process (MDP), which is a tuple $M = \langle S, U, T, R, \gamma \rangle$, where

- S is the set of possible world states,
- U is the set of possible control actions,
- T is a transition function $T : S \times U \rightarrow P(S)$,
- R is the reward function $R : S \times U \rightarrow R$,
- and γ is a discount factor such that $0 \leq \gamma \leq 1$.

Reinforcement learning learns a policy $\Pi : S \rightarrow U$, which defines which actions should be taken in each state. Q-learning [86] is a model-free reinforcement learning technique that uses a Q-value $Q(s, u)$ to estimate the expected future discounted rewards for taking action u in state s. At each step, Q-learning applies an update equation for $Q(s, u)$ given by

$$Q(s_t, u_t) \leftarrow Q(s_t, u_t) + \alpha \left(r_{t+1} + \gamma \max_u Q[s_{t+1}, u_t] - Q[s_t, u_t] \right)$$

where r_{t+1} is the reward observed after performing action u_t in state s_t, α is the learning rate ($0 \leq \alpha \leq 1$), and s_{t+1} is the state that the agent transitions to after performing action u_t. After $Q(s_t, u_t)$ converges, the optimal action for the agent in state s_t is $\arg\max_u Q(s_t, u)$.

In Q-learning and related algorithms, an agent maintains a table of $Q[S, U]$ based on its history of interaction with the environment. An experience $\langle s, u, r, s' \rangle$ provides one data point for the value of $Q(s, u)$. The data point is that the agent received the future

value of $r + \gamma V(s')$, where $V(s') = \max_{u'} Q(s', u')$; this is the actual current reward plus the discounted estimated future value. This new data point is called a return. The agent can use the temporal difference equation to update its estimate for $Q(s, u)$:

$$Q[s, u] \leftarrow Q[s, u] + \alpha(r + \gamma \max_{u'} Q[s', u'] - Q[s, u])$$

or, equivalently,

$$Q[s, u] \leftarrow (1 - \alpha)Q[s, u] + \alpha(r + \gamma \max_{u'} Q[s', u']).$$

4 RL Results

This section presents a summary of our results. We have shown that RL can be used to dynamically learn how to stabilise a robot's trajectory given previously unseen faults. We used a quadcopter to demonstrate how path-following deviations can be reduced by 63-75% following few learning epochs.

We will firstly present the final Q-Matrix (Figure C.1) that was learned. Positive and negative Q-values represent good and bad performance respectively. The large number of negative values are due to our reward function, as will be shown in Section 5.1. We say the matrix has converged perfectly if only a single positive Q-Value exists per row. This is not evident in this matrix but with adjustments to the reward function and enough learning epochs this matrix would converge. Another interesting point to note is the magnitude of the Q-Values. The first row has significantly higher values than the rest. This is because the first row represents the lowest deviation rate and this row will get trained for every fault since for the deviation rate to increase to the higher partitions it must first go through first partition. To see the improvements in trajectory following performance for a quadcopter, we compared the RLC with the original quadcopter configuration. As far as the authors are aware there is currently no baseline controller for unknown faults to compare against. In future we intend to run comparisons against other FTC architectures. We use total trajectory deviation in centimetres to gauge how much the tracking accuracy improved. Table C.1 shows this comparison for several magnitudes of unknown rotor faults. We denote the number of learning cycles of RLC using μ.

Figure C.2 shows the time it took to re-stabilize the quadcopter along its trajectory with acceptable deviation after each completed learning cycle. It is clearly visible that after a small number of learning iterations the time to re-stabilize drastically reduces

	1	2	3	4	5	6	7	8	9	10
1	5.5512	-14.4008	-7.4802	-20.3058	-21.5061	-11.5752	-19.8285	-18.9405	-6.9193	-13.7548
2	0.1300	-1.9945	-1.3927	-0.6599	-0.2186	-0.3024	-0.9682	1.8140	-0.0110	-0.9813
3	-1.9424	-2.3690	-0.4745	-0.5938	-1.6747	-1.5527	0.0527	0.2276	0.4315	-0.4031
4	-2.4353	-0.3924	-0.2406	-0.0685	-0.1756	-0.5804	-0.3610	-0.0662	-0.1040	-0.5393
5	-4.6703	1.3629	-1.2179	-0.2771	-1.8217	0.5362	-1.5284	-0.4245	-0.2020	-0.3820

Figure C.1: Matrix learned after 250 learning cycles. Rows represent different system states and each column represents a blended controller.

Table C.1: Empirical Analysis of RLC improvment in path deviation error against the original controller.

Rotor Fault	Original Quadcopter(cm)	RLC Quadcopter (cm)	μ	Improvement
4%	532.81	131.39	100	75.34%
6%	2188.1	650.92	200	70.25%
7%	5439.9	1995.1	250	63.32%

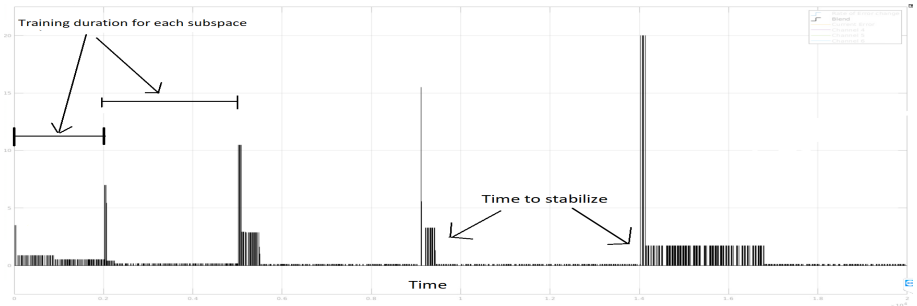


Figure C.2: Evaluation phase τ timings.

but never converges to 0. The spikes in the graph show an increase in the magnitude of a benchmark fault induced.

5 Reinforcement Learning Blended Control

This section describes the mapping of the RL approach to our experimental domain, i.e. what the RL tuple $\langle S, U, T, R, \gamma \rangle$ means for our quadcopter domain. For the experimental set-up we used an opensource Matlab implementation of a quadcopter flight simulation [87]. This implementation used 4 tuned PID (Proportional Integral Derivative) controllers to control Roll, Pitch, Yaw and Altitude (Figure C.3 Right) respectively.

Our objective with this simulation is to implement on-line reinforcement learning for faults of unknown origin and magnitude. As far as the authors are aware not much work has been done using RL to learn a blended controller for unknown faults on-line. The simulations core task is trajectory following which is defined as a list of time indexed way point locations in the form of a triplet ($\mathcal{X}$, $\mathcal{Y}$, t), presenting desired X and

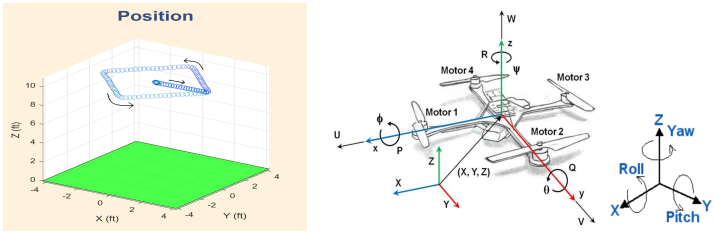


Figure C.3: Left: Simulation Path. Right: Roll , Pitch and Yaw axis of the quadcopter.

Y position at time t. Given the quadcopter's current position at time t as $(\bar{x}, \bar{y})$ the deviation from the trajectory can be computed as:

$$\delta_t = \sqrt{(X - \bar{x})^2 + (Y - \bar{y})^2}$$

The trajectory used is a square and can be seen in Figure C.3 Left. We define a single learning cycle for RL as a single execution of the shown trajectory with one unknown fault. The faults are injected at the same point in the trajectory for consistency. After every learning cycle we allow for time to fully stabilize along its trajectory so no residual effects from previous learning cycles will influence future learning quality. Since we have a cyclic trajectory we can continuously repeat the learning cycles and analyse the performance improvements over time and *on-line*. Each learning cycle can be defined in terms of four phases, which are:

A. Stabilize - Ensure nominal operating conditions.

B. Learning - Random fault in specified range and stabilizing using RL.

C. Stabilize - Ensure nominal operating conditions.

D. Evaluate - Test against a benchmark error to check improvement.

The **Stabilize** phases simply ensure the quadcopter returns to its trajectory with acceptable deviation rate and in nominal operating conditions before proceeding to the following phases. The **Learning** phase consists of random fault injection and stabilizing with RL. The **Evaluation** evaluates the current learned policy against a benchmark fault to record the improvements learned over time, Figure C.2.

5.1 RL States

The space of unknown faults is simply too large to apply learning to each fault individually. We hence classify the states S for our RL implementation in terms of the *effect* that unknown faults have on the trajectory following task. For this metric we

chose the deviation rate from the desired trajectory, which we define as:

$$\rho = \frac{d}{dt} \sum_{i=0}^{4} \delta_{t-i},$$

where δ_i is the trajectory error between current and desired position at time step $t - i$. We then define S as a partition over the parameter space of ρ into N regions of equal size.

5.2 RL Actions

Our action space U is defined in terms of blended control for which we use a linear combination of predefined controllers. In other words, given a set of M predefined controllers $\Lambda = \{\Lambda_1, \cdots, \Lambda_M\}$ and corresponding weights $\{\varphi_1, \cdots, \varphi_M\}$, our applied blended control is given by

$$\Lambda^* = \sum_{i=1}^{M} \varphi_i \Lambda_i. \tag{C.1}$$

where $\sum_{i=1}^{M} \varphi_i = 1$. We define our action space U as a partition of size P over the parameter space of φ. The granularity of this partition will dictate how many different blended controllers are used for learning.

We then define our Q-Matrix as $Q(S, U)$ and set N and P to 5 and 10 respectively. In other words we are learning the optimal blended controller for each deviation-rate sub-partition. The size of the matrix increases drastically with the size of the partitions. We hence chose small enough partition sizes to concentrate the learning into a smaller region. The transition function T maps the current state and action chosen to the new state. In our context T is already given by the simulation itself.

For RL to be able to learn *on-line* we must have some performance metric to evaluate how well a controller performed during the fault recovery for each separate learning cycle. For this metric we choose the time, τ, until the quadcopter is stabilized on its desired trajectory.

5.3 RL Rewards

Since we use τ as our performance metric we must also compute a baseline value to compare this against, which indicates positive or negative reward for the blended controller used. Since we have no prior knowledge about the faults, we compute $\bar{\tau}$,

Table C.2: $\bar{\tau}^i$ values for each partition of S after the RL simulation.

S^i	S^1	S^2	S^3	S^4	S^5
$\bar{\tau}^i$ (sec)	2.8	3.6	3.9	4.3	4.7

a running average of τ. For each subspace of S^i we define $\bar{\tau}^i$ to allow larger errors more time to stabilize. We assign credit based on the performance against the running average.

$$\mathcal{C} = \begin{cases} 1 & \text{if } \tau \leq \bar{\tau}^i \\ \text{-1} & \text{if } \tau > \bar{\tau}^i \end{cases}$$

given that $max(\rho) < S^i$. The $\bar{\tau}$ values for each fault mode after the RL simulation can be found in Table C.2.

5.4 Credit Assignment Problem

The *Credit Assignment Problem* [88] refers to the problem of identifying **which** action was the one deserving credit. In our case the problem is identifying which partition S^i should receive the credit after a learning phase. This is because ρ will naturally vary across multiple sub-partitions of S during a single learning cycle. We hence apply RL for each of the sub-partitions. The blended controller used to control the system is changed when ρ transitions from its current S^i to S^{i-1} or S^{i+1}. Figure C.4 shows various signals and the bounds of each S^i indicated as *Levels* from the quadcopter simulation during a single learning cycle. The Green signal represents the changing ρ. We indicate the transition between sub-partitions of S with red arrows. Notice the blue signal representing the applied control signal changes when the deviation magnitude changes. However this implementation makes it difficult to assign credit to a single subspace, since learning was potentially applied on multiple subspaces. We combat this by assigning proportional credit to each subspace depending on how long the value of ρ stayed in each of the parameter subspaces. That is, the credit assigned to each S^i is:

$$\mathcal{R}(S^i) = \mathcal{C} * \frac{t}{\tau}$$

where t is the duration that ρ was in the bounds of S^i and $\mathcal{C}$ and τ are as previously defined.

For this implementation we do not use γ, the future expected reward. Estimating the

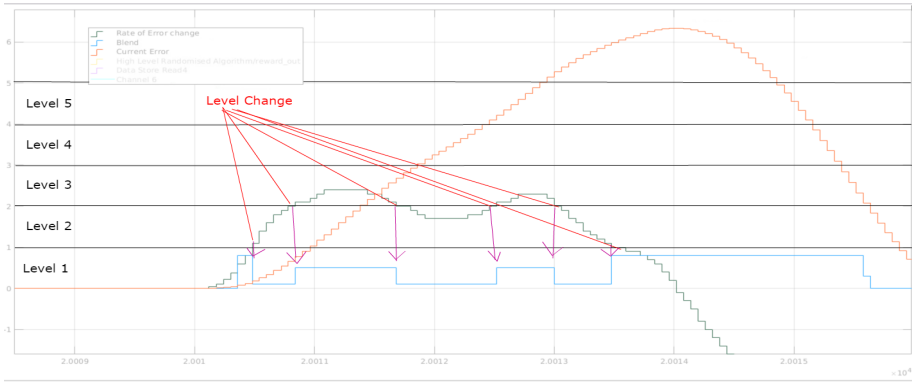


Figure C.4: Graph showing ρ (green), current error (orange) and blended controller used (blue). The subspaces $S^1, S^2, ..., S^5$ are indicated along the Y-axis using the *Level* terminology.

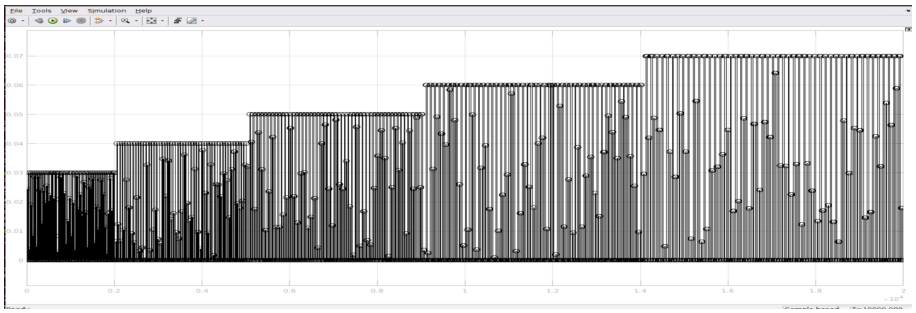


Figure C.5: Rotor Fault magnitude ι and benchmark errors Γ throughout simulation. Note: larger errors are given longer time to stabilize.

expected deviation rate ρ_{t+1} is challenging as the controller is synthesised on-line and the faults are unknown. In future work we will extend the reward function to include the γ term.

5.5 Error Generation

We focus solely on unknown rotor faults for this article for simplicity but this framework works for any unknown faults that cause trajectory deviations. More specifically, we use a multiplicative term ι on the rotor speed to represent the unknown fault. To achieve an even distribution of learning we randomize ι, such $0 \leq \iota \leq \Gamma$, where Γ is the upper limit on the fault. We incrementally increase Γ after a number of learning cycles. This will give a larger variance of deviations allowing the system to apply learning across all sub-partitions. Figure C.5 shows the rotor fault magnitudes used over 250 learning cycles. Note this figure shows the benchmark errors used as well as randomized errors for training.

Table C.3: Low level nominal and fault controller tuning for each control axis.

Controller	λ_ϕ^N	λ_ϕ^F	λ_θ^N	λ_θ^F	λ_ψ^N	λ_ψ^F
P	2	10	2	10	4	14
I	1.1	5	1.1	5	0.5	2
D	1.2	3	1.2	3	3.5	5

5.6 Simulation Details

For completeness we will give a full list of low level controllers and other parameters used for the experiments. The PID controller coefficients for nominal and fault (indicated by superscript N and F) controller for each control axis (indicated by subscript ϕ, θ, ψ) can be seen in Table C.3. It is worth noting that these controllers are not tuned for any specific fault; they are simply tuned with a more "aggressive" coefficient. Figure C.1 shows the matrix after 250 learning cycles. We set the initial value for Γ to 3% and increased this after every 50 learning iterations up to 7%. We apply the fault to the same rotor every time.

6 Conclusion

In this article we presented a novel reinforcement learning approach for on-line, real-time learning for unknown faults. We empirically demonstrated the effectiveness of this approach on a Quadcopter trajectory following task. We noticed a 63-75% decrease in the trajectory deviation due to an unknown fault after a small number of learning cycles. In this set of experiments only rotor faults where investigated but given appropriate controller pairs for blending the authors believe this approach will work for other system disturbances such as wind or sensor faults. Since we are learning on the disturbance space, instead of the fault parameter space, this learning approach will work for most faults that cause a path deviation. The novelty of this approach is that the learning phase can be conducted on-line and needs very few iterations before converging compared to traditional RL methods. Furthermore, we are not aware of any other RL based approach for FTC under novel faults. This could provide adaptive FTC capabilities to systems that are not easily fixable or recoverable such as satellites or space rovers. Our future work includes focusing on improving the learning process and attempting training on a multitude of errors. For simplicity, this article also only explores blends outputs of two controllers but in theory this is not a limitation. Larger sets of predefined controllers for blending can be trained using our described method but with a considerably larger number of training phases, which will also be explored in future work.

Paper D

Unknown Fault Tolerant Control using Deep Reinforcement Learning: A blended control approach

Abstract

It is impossible to pre-define a controller for every fault an autonomous system can experience as some faults are unknown at design time. Current fault tolerant control (FTC) architectures switch control to a pre-defined fault controller when a known fault is identified. *Blended control* implements a controller that is composed of multiple individual controllers instead of discretely switching between them. In this article we present a novel fault tolerant control architecture based on blended control that uses a high-level deep learning agent to learn the optimal blending proportions between low-level controllers for unknown faults. Faults are abstracted to the effect they have on the performance of a task while removing the inherent fault identification delays experienced by existing FTC architectures. The presented architecture is validated on a quadcopter trajectory tracking task and trained to tolerate abrupt rotor loss of effectiveness. We compare our approach against a switched architecture with the same underlying controllers and show its ability to learn unknown fault tolerance.

1 Introduction

Autonomous systems have been a major focus for the fault tolerant control community in recent years. For the application of autonomy to large scale systems they need to tolerate unknown faults. *Fault tolerance* can be defined as continuing mission performance under any fault conditions, known or unknown. Traditional switched fault tolerant control (FTC) systems rely on prior knowledge of the effects a fault has on the system dynamics for the fault detection and isolation unit (FDI) to identify the fault as well as a controller to operate under these conditions [89]. After identification Control is switched to the pre-defined fault controller. This fundamentally does not extend to unknown faults for two reasons:

A. Identification relies on prior knowledge of the effect a fault has on the system dynamics.

B. Fault controllers are designed to operate under known fault conditions only.

Identifying the effect a fault has on the system dynamics is complex and has an inherent time delay. For highly unstable systems, such as quadcopters, such a delay can be catastrophic. Blended Control is a variation of switching control that implements a control that is composed of multiple low-level controllers simultaneously instead of discretely switching between them.

In this article we present a novel hierarchical FTC framework based on blended control and a deep learning high-level controller that addresses the mentioned problems. The FDI unit and control switching function are replaced by a deep learning agent, specifically a deep deterministic policy gradient (*DDPG*) agent. This is an abstracted approach to learn the effect of a fault on the task performance rather than the system dynamics. Degrading task performance due to any faults is optimized through changing the blend weight vector which removes the need for prior knowledge of a fault and addresses problem 1. The low-level controllers are designed based on the *type* of reaction to a fault instead of for pre-defined fault conditions while providing similar control responses under nominal conditions. This allows for synthesis of new controllers for unknown faults by adapting the blend weight vector and addresses problem 2. The presented architecture provides a novel integration of existing controllers to deep learning FTC and is validated on a Quadcopter trajectory tracking task with abrupt rotor loss of effectiveness. We show the presented architecture is able to track a given trajectory closer than a switched architecture based on the same low-level controllers under rotor loss of effectiveness of 50% while also generating a improved control signal.

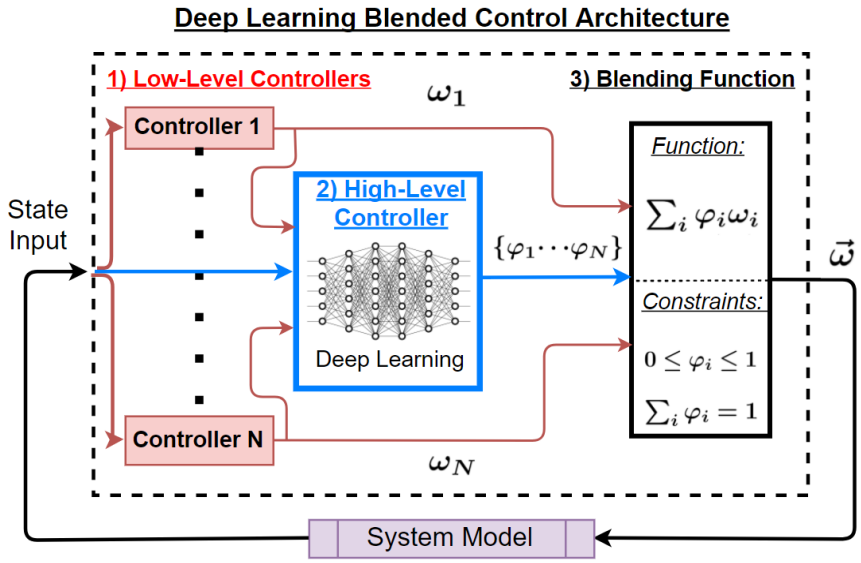


Figure D.1: Deep Learning Blended Control architecture showing the 3 main parts: Low-level controllers, High-Level Controller and Blending Function.

Our contributions are as follows:

- We present a novel hierarchical fault tolerant control architecture with the ability to learn unknown fault tolerance through blended control and a high-level deep learning agent.
- The architecture is validated on a quadcopter trajectory tracking task under unknown rotor loss of effectiveness faults. The trained controller shows robustness to the trained faults and exhibits less oscillations around the reference signal than the low-level controllers.

The remainder of this article is structured as follows: Section 2 gives a detailed overview of the presented architecture. We compare our architecture to current state-of-the-art FTC architectures and applications of deep learning for control in Section 3. We discuss deep reinforcement learning and its application to control in Section 4. The implementation and training on a quadcopter simulation is given in Section 5 followed by experimental validation in Section 6.

2 Deep Reinforcement Learning Blended Control

We will firstly give a detailed overview of the presented architecture, referred to as Deep Reinforcement Learning Blended Control (**DRLBC**), to improve the overall clarity of the article. An architecture diagram of DRLBC can be seen in Figure D.1.

The framework can be broken down into three parts which will be discussed in detail:

A. Low-Level Controllers

B. High-Level Controller

C. Blending Function

2.1 Low-Level Controllers

In hierarchical control architectures the low-level controllers generate the control signals that are directly applied to the systems actuators (motors, valves, etc). Several types of well known controllers exist for system control such as Proportional Integral Derivative (PID), Model-Preditive Controllers (MPC) or Linear Quadratic Regulators (LQR) to name a few. In this article we will restrict the low-level controllers to PID controllers which are an industry standard way of controlling automatic systems but the presented framework works with any low-level controllers. Traditionally an optimal controller is designed offline for a system model under predefined operating conditions. Fault tolerance is achieved through redundant controllers designed for known fault conditions. The number of faults a system is tolerant for depends on the number of low-level controllers predefined at design time. In this article, the set of low-level controllers are not designed for specific operating conditions but based on the *type* of reaction to a disturbance or fault and are denoted $\Omega = \{\omega_1, \cdots, \omega_N\}$. Figure D.2 shows the reaction of two differently tuned PID controllers to a fault. Gain parameters for these can be found in Table G.2. Higher gain parameters cause the controller to have a more *aggressive* response to a fault and performs better for aggressive flight maneuvers (top) while smaller gain parameters have a *smoother* response which results in more precise control but slower stabilization after faults (bottom). The blue line representing the actual state variable that is manipulated stabilizes on the black reference line for both controllers. The controllers reactions shown in Figure D.2 are used in the quadcopter simulation experiments and will be elaborated on further in Section 5.1.

2.2 High-Level Controller

In traditional switched architectures the high-level controller contains the FDI unit which identifies the effect of faults on the system. The high-level controller selects which of the low-level controllers is active at any time and provides fault tolerance capabilities by switching control to the predefined controller once a known fault is identified. In the presented architecture the high-level controller is implemented with

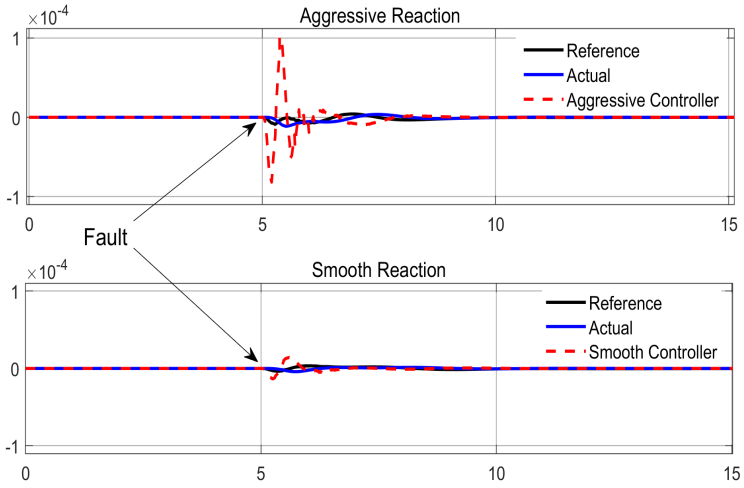


Figure D.2: Quadcopter attitude controller outputs for trajectory tracking under rotor fault. Y-Axis represents Reference and Actual position as well as controller response in red, Aggressive (Top) and Smooth (Bottom).

a deep neural network instead of a set of state observers that identify faults. In this article we will use a Deep Deterministic Policy Gradient network [90] which will be described in more detail in Section 4. The low-level controller outputs, Ω , and a subset of state variables representing the performance on a task are used as an input for the high-level controller. The objective is to learn the optimal blend of low-level controllers to maximise the systems performance on a task. Faults that impact the performance of the system can be mitigated generically in real time without having to define specific fault observers. The output of the high-level controller is defined as the blend weight vector $\boldsymbol{\varphi} = \{\varphi_1, \cdots, \varphi_N\}$ that specifies the weight of each low-level controllers in the blended control signal $\vec{\omega}$ applied to the system.

2.3 Blending Function

The blending function takes as inputs the low-level controller outputs $\{\omega_1, \cdots, \omega_N\}$ and the blend weight vector $\{\varphi_1, \cdots, \varphi_N\}$ and outputs a blended control signal $\vec{\omega}$. Formally blended control can be defined as follow:

Definition 13 (Blended Control)**.** *Given a collection of controllers* $\Omega = \{\omega_1, \cdots, \omega_N\}$ *and a blend weight vector* $\boldsymbol{\varphi} = \{\varphi_1, \cdots, \varphi_N\}$*, blended control is a weighted combination* $\vec{\omega} = \sum_i \varphi_i \omega_i$ *such that: (1)* $\forall_i, \omega_i \in \Omega, 0 \leq \varphi_i \leq 1$*, and (2)* $\sum_i \varphi_i = 1$

The two constraints imposed on the blend weight vector ensure that $\vec{\omega}$ is bound by the low-level controller outputs. Further, if the low-level controllers output the same control signal, blending to any proportions will have no effect. In Figure D.1 these constraints are shown in the blending function block for clarity but can be imposed on

the high-level controller outputs for a simpler implementation.

3 Running Example & Related Work

We will use the quadcopter with abrupt rotor loss of effectiveness (LOE) as a running example for the remainder of this article. Quadcopters are unmanned aerial vehicles that use four propellers to maneuver and have gained increased attention in the research community in recent years . These vehicles have fewer actuators than degrees of freedom, and hence are called under-actuated: only four actuators (propellers) are used to control six variables, the coordinates x, y, and z, and the roll, pitch, and yaw angles of the quadcopter, denoted ϕ, θ, and ψ, respectively. Hierarchical PID-based control is a standard way to control quadcopters. The dynamical equations of a quadcopter are complex, due to the highly coupled state-space. Due to space limitations, we give a brief summary of quadrotor dynamics and details of how rotor faults are represented, and refer the reader to [91] for details.

We define the dynamics of the quadcopter in the non-linear state space form

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}) + g(\boldsymbol{x})(1 - \varsigma)\boldsymbol{u}(t), \tag{D.1}$$

where $\boldsymbol{x} = [x\ \dot{x}\ y\ \dot{y}\ z\ \dot{z}\ \phi\ \dot{\phi}\ \theta\ \dot{\theta}\ \psi\ \dot{\psi}]^T$ is the state vector, control input $\boldsymbol{u}(t) = [U_1\ U_2\ U_3\ U_4]^T = \varrho(\upsilon_1\ \ \upsilon_2\ \ \upsilon_3\ \ \upsilon_4)$, ϱ is a non-linear function in the angular velocity of motor i, and we denote a multiplicative fault model with parameter $0 \leq \varsigma_i \leq 1$ for $i = 1, ..., 4$, where ς_i = 0 corresponds to nominal function and ς_i = 1 to total failure.

Quadcopters have been shown to be able to maintain flight even after the complete loss of one or more rotors [92]. Several applications of deep learning for quadcopters exist which mostly focus on learning the direct control mapping of state space to motor commands [93, 94, 95]. This usually requires large amount of training data and complex fine tuning of the reward functions to accurately represent the desired behaviour.

The application of deep learning for FTC of a quadcopter has been achieved by learning a complementary controller that adjusts the nominal controller output during a rotor fault [96]. The success of this approach compared to other adaptive control strategies is attributed to the continuous output of the neural network and removal of the FDI unit to identify when a correction to the nominal controller is needed.

A large amount of work has been done on FTC through hierarchical control architectures [89, 97]. An extension to the traditional switched architectures is blended

Table D.1: PID parameters for aggressive and smooth reactions to faults from a quadcopter simulation.

	$\mathcal{P}$	$\mathcal{I}$	$\mathcal{D}$
Aggressive	12	5	5
Smooth	5	1.1	3

control. Blended control has been proven to be safe as the applied control signal will always be bound by the underlying controllers and will at worst perform like a switched architecture [27]. Blended control has been successfully applied for FTC of a quadcopter with partial rotor failure in [2] but has received little success outside of this due to the complexity of generating adequate blending weights.

3.1 Comparison to DRLBC

We contrast the presented DRLBC architecture to current state of the art approaches. The deep learning algorithm is applied to an abstracted high-level task while relying on existing control mechanism to control the vehicle. This reduces the overall complexity of the learning task as nominal system control is already established which usually takes a large number of learning iterations and fine tuning an accurate reward function to achieve. The direct mapping of state space to motor commands is a highly complex function. Every additional input to the deep learning algorithm increases the size of the space that is explored which is a reason for the long convergence times experienced by the application of deep learning to control tasks. DRLBC only uses a subset of the state space variables as an input which reduces this space drastically. If the underlying controllers provide safe responses with varying performance on a task, DRLBC guarantees a safe exploration space for the agent during training as any blend weight vector used will create a control signal bound by the safe controller outputs.

The presented architecture is able to generically identify degrading performance since no fault specific observers are used in the high-level controller. Faults have an effect on the performance an autonomous system is able to achieve on its task. For example abrupt rotor LOE on a quadcopter will cause overall instability and reduce the trajectory tracking accuracy the quadcopter is able to achieve. By choosing inputs for the high-level controller that represent the performance on a given task the deep learning agent is able to identify when system performance degrades and learns to optimize this through the selection of an adequate blend weight vector. For a trajectory tracking task such a performance measure could be the current trajectory tracking error as it captures the overall task the system is executing. To be able to distinguish the effects different faults have on the overall system we extend the set of high-

level controller inputs with a subset of the state variables that represent the changing conditions of the system. The angular state of a quadcopter is heavily effected by rotor faults. This would be an adequate input for the high-level controller to learn the effect a rotor fault has on the system performance without specific observers monitoring these states. This abstraction allows the system to learn how to maintain control during unknown faults or even when the fault is not identifiable from any state variables.

With specifically tuned fault controllers blended control is mostly limited to the correction of partial faults. The calculations of the correct blending weights for partial fault control is complex as the FDI observers need to be able to identify the magnitude of the fault and then calculate the adequate blend weights. DRLBC is not limited to partial fault tolerance as the low-level controllers are not tuned to be fault specific. Blending low-level controllers based on the type of control that is required allows for the deep learning agent to synthesise a new controller when system performance degrades due to any fault. The complexity of defining the blending weights is left to the deep learning agent. Theoretically this framework can learn fault tolerance on-line given an adequate training environment but this is beyond the scope of this article.

4 Deep Reinforcement Learning (DRL)

The standard setup for reinforcement learning is a decision maker called agent that interacts with an unknown environment E in discrete time steps for achieving a goal. The information exchanged between the agent and its environment is reduced to three signals: one signal to represent the choices made by the agent $a_t \in \Re^N$ (the actions), one signal to represent the basis on which the choices are made $x_t \in \Re^M$ (the observations), and one scalar signal that represents the agent's goal $r_t \in \Re$ (the reward) [98]. Here, we assumed the environment is partially-observed ($x_t = s_t$ where s_t is the state vector).

The agent makes a decision based on a policy π that maps the states to a probability distribution over the actions $\pi : S \rightarrow P(A)$. The environment is modeled as a Markov decision process defined by a four tuple: $\{S, A, T, R\}$. The transition function $T : S \times A \times S \rightarrow [0, 1]$ allows to estimate the probability of reaching state s' at $t + 1$ given that action $a \in A$ was chosen in state $s \in S$ at time t, $p(s'|s, a) = Pr\{s_{t+1} = s'|s_t = s, a_t = a\}$. The reward function estimates the immediate reward $R \sim r(s, a)$ obtained from choosing action a in state s.

The goal of the agent is to learn a policy that maximizes the future discounted reward $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$ over a time period T without explicit knowledge about

the shape of the reward or the dynamics of the environment. Therefore, solving a reinforcement learning problem means, roughly, finding the policy function that maximizes the expected reward over the long run. One approach to find the best policy is to derive it from the so-called action-value function that approximates the expected reward for any state and action pair. Hence, the optimal action-value function Q must be learned

$$Q^*(s_t, a_t) = \max_{\pi} \mathbb{E}_{r_{i\geq t}, s_{i>t}, a_{i>t} \sim \pi}[R_t | s_t, a_t] \tag{D.2}$$

Deep neural networks have been successfully used as function approximators for learning the optimal action-value function. The Deep Q Network (DQN) algorithm was first proposed to deal with continuous state spaces [99]. However, DQN can only handle low-dimensional action spaces mainly because of the curse of dimensionality. Then, Lillicrap et al. (2015) proposed an off-policy actor-critic algorithm using deep function approximators that can learn policies in high-dimensional, continuous action spaces [90]. Their algorithm, called Deep Deterministic Policy Gradient (DDPG), promotes stability and efficiency by training the network off-policy with samples from a replay buffer and using a target Q network to give consistent targets during temporal difference backups.

Learning a set of deep neural networks for direct fault-tolerant control of a quadrotor is a complex problem. The two main challenges are the design of an appropriate reward function and the time-consuming exploration process required given the high-dimensional action and observation space required. We leave a deeper investigation into these challenges for future work. The goal of using reinforcement learning in this work is to design an agent that maps from an observation vector to the optimal blended weight vector depending on the system state.

5 Quadcopter Implementation and Training

Hierarchical PID-based control is a standard way to achieve quadcopter trajectory tracking. A trajectory is a temporally-indexed set of coordinates in 2D or 3D, denoted $\boldsymbol{\zeta}(k)$. We denote the reference (desired) trajectory as $\boldsymbol{\zeta}^R(k)$, and the executed trajectory as $\tilde{\boldsymbol{\zeta}}(k)$. The goal of a trajectory tracking task can be defined as minimizing the Trajectory Loss.

Definition 14 (Total Trajectory Loss)**.** *We can represent the total trajectory loss as a difference function between reference and executed trajectories, i.e.,* $\mathcal{L}_{0:T} = \sum_{k=0}^{T} \parallel \boldsymbol{\zeta}^R(k) - \tilde{\boldsymbol{\zeta}}(k) \parallel$ *for a trajectory over time points* $k = 0, \cdots, T$.

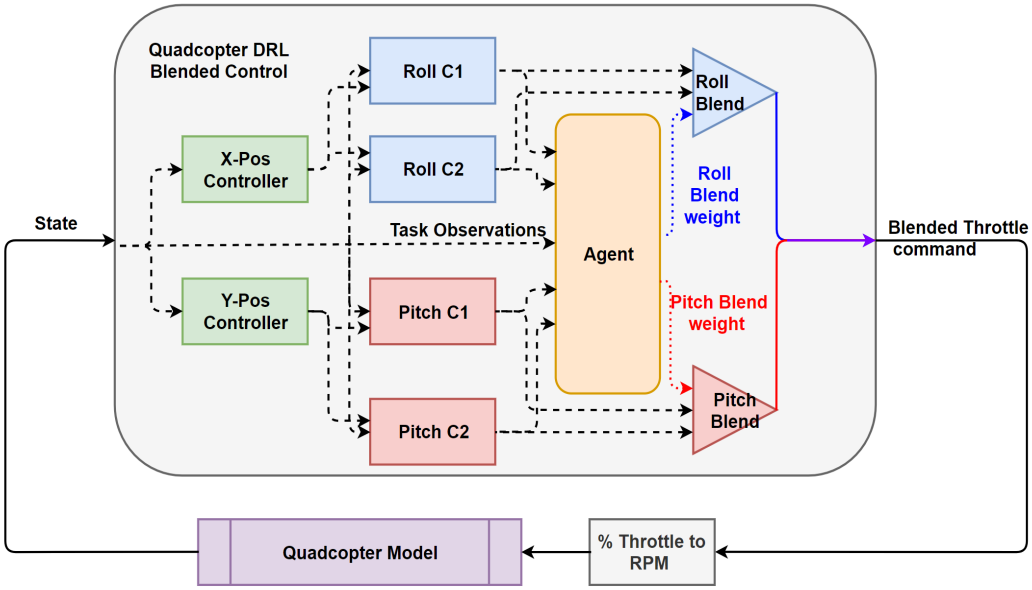


Figure D.3: Quadcopter Deep Reinforcement Learning Blended Control architecture.

For simplicity we will focus on 2D (x, y) trajectory tracking. Figure E.1 shows the full architecture diagram implemented on the quadcopter which will be discussed similarly to Section 2, omitting the blending function details as they do not change.

5.1 Low-Level Controllers

A PID controller for each x and y axis generates the desired Roll (ϕ) and Pitch (θ) reference angular state needed for the quadcopters to execute the desired trajectory. The low-level roll and pitch controllers translate the desired angular states into motor throttle commands which are applied to the vehicle. We disregard ψ, the rotational state of the quadcopter, as it is not relevant for the 2D position. By changing the gain parameters of the roll and pitch PID controllers we change how the quadcopter responds to a divergence of the reference trajectory. To achieve a blended control architecture at least one extra controller is needed for each of the angular states for which the control is to be blended. As previously mentioned an *aggressive* and a *smooth* controller, referred to as C1 and C2 respectively, are used and for simplicity both roll and pitch use the same controller tuning. Table G.2 show the detailed gain parameters used and Figure D.2 shows the difference in response to an abrupt change in position due to some unknown fault.

5.2 High-Level Controller

A standard actor-critic DDPG network structure is used to generate the blend weight vector. Both actor and critic take in the same observation vector which we define as:

$$[\phi \;\; \theta \;\; \delta_X \;\; \delta_Y \;\; C1_\phi \;\; C2_\phi \;\; C1_\theta \;\; C2_\theta]$$

where ϕ and θ are the current angular states, δ_X and δ_Y represent the current trajectory tracking error and the remainder the low-level controller outputs of C1 and C2 for ϕ and θ respectively.

Since there are only two controllers for each control axis being blended and the second blended control constraint from Definition 29 enforces $\sum_i \varphi_i = 1$, the actor output can be defined simply as : [φ_ϕ φ_θ]. The full blend weight vector $\boldsymbol{\varphi}$ can then be calculated as :

$$[\varphi_\phi \;\; (1-\varphi_\phi) \;\; \varphi_\theta \;\; (1-\varphi_\theta)]$$

This reduces the neural network output to one variable per control axis compared to current state of the art approaches learning the direct control mapping of *all* control signals needed to control the system. We give a brief overview of the network architecture but details are omitted due to space restrictions. The standard DDPG network was used without special modifications. The actor network is defined by three fully connected layers separated by ReLU (Rectified Linear Unit) layers. Finally a hyperbolic tangent layer with output size 2 is used to naturally enforce the blended control constraints, bounding φ_ϕ and φ_θ between 0 and 1. The critic network has two paths, one for the observation vector and the other for the actor output which are joined after two and one fully connected layer respectively for each path. All fully connected layer contains 32 neurons in this implementation.

5.3 Training Details

For training we use a simple sinusoidal reference path of 10 meters for the X and Y position executed over 15 seconds. We define the performance of the quadcopter on the trajectory tracking task as the **average trajectory loss** over the 15s simulation. The performance of C1 and C2 under nominal conditions is 48.88cm and 48.86cm respectively. The quadcopter was trained over 3000 episodes and we define other relevant training parameters used in Table E.2.

Table D.2: Training parameters used

Parameter	Value
Discount factor	0.99
Initial learning rate of the critic	0.01
Initial learning rate of the actor	0.025
Batch size	5
Replay buffer size	5
Training steps of an episode	150
Number of episodes	3000

5.3.1 Rotor Fault Generation

We use abrupt rotor loss of effectiveness as the fault to learn which has briefly been defined in the quadcopter model in Equation F.7. For the purposes of training we extend the definition of the rotor faults to a triplet:

$$[\varsigma \ \ \gamma \ \ t]$$

where ς defines fault magnitude, γ discretely selects the rotor and t the time of occurrence. We define the sampling interval $0.2 < \varsigma < 0.5$ indicating a loss of angular rotor velocity of 20-50%. Each fault parameter is sampled randomly to provide a varied set of training data and the probability of a fault occurring at time t is set to 5%. Each fault is applied for 0.1s which is one time step.

5.3.2 Reward function

Nominal control is already established through the low-level controllers and the objective of the agent is to learn to tolerate any fault by maintaining nominal task performance. The total trajectory loss over the training path under nominal conditions, defined $\mathcal{L}^N_{0:T}$, makes for a natural baseline to compare the performance achieved under fault conditions against.

Definition 15 (Trajectory Tracking Reward)**.** *Given total trajectory loss under faults $\mathcal{L}^F_{0:T}$, the obtained reward is defined as:*

$$\mathcal{R}_{0:T} = \mathcal{L}^N_{0:T} - \mathcal{L}^F_{0:T}$$

for time points $k = 0, \cdots, T$.

where $\mathcal{L}^N_{0:T}$ is the average total trajectory loss of C1 and C2 under nominal conditions. The reward function is defined independently of the faults applied to the system

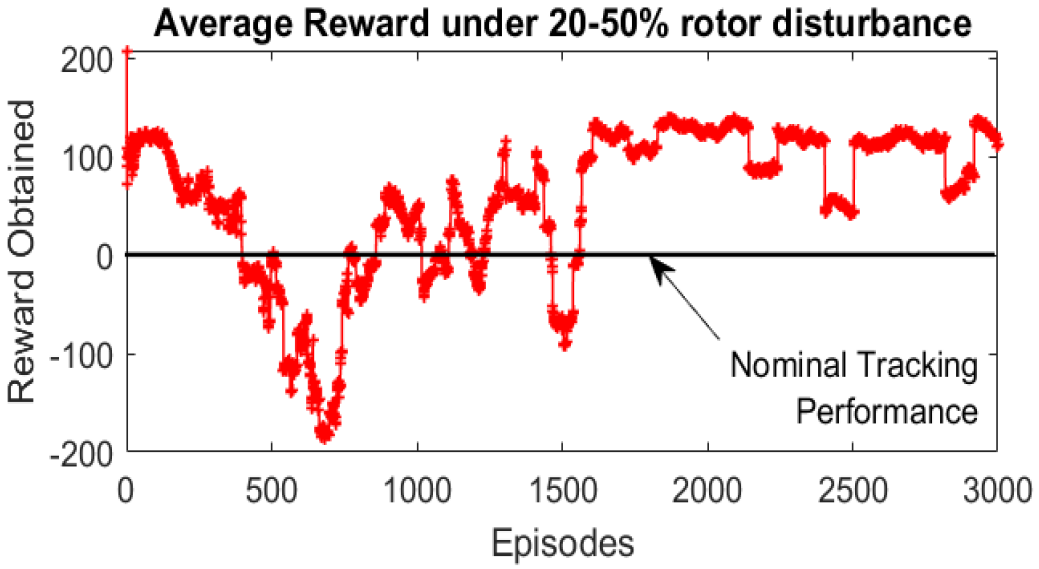


Figure D.4: Average reward obtained over 3000 episodes shown against nominal tracking performance.

which make it extensible for any fault that effects the trajectory, for example wind disturbances, but this is beyond the scope of this article. The reward function allows for a positive and negative rewards indicating improved or degraded system performance which training should maximise.

5.4 Training Results

Figure F.2 shows the average reward obtained by the agent over 3000 episodes calculated over 100 training episodes. This figure shows that the reward stabilizes to a positive value indicating that the agent performed better than nominal control. We partially attribute this to the way the reward is calculated. During rotor faults the quadcopter can actually move closer to the reference trajectory due to the steady state error experienced during tracking. This has a positive effect on the overall trajectory loss. Additional factors for this are explored during the experiments section. Since the average reward is strictly positive after 1500 episodes we reason the agent has learned to stabilize the fault correctly without having a dedicated controller predefined for the rotor fault condition.

6 Experiments

We compare the trained blended control framework against a traditional switched architecture based on the same low-level controllers. We set C2 as the nominal

controller as it performs slightly better under nominal operating conditions and smooth control is more desirable. After experimentation we found that C1 was more robust to rotor faults which make it an adequate fault controller in a switched architecture. A switch is triggered when the trajectory deviation is more than 2m, since the steady state error is around 1.8m this gives a small margin of error for deviations before identifying a fault.

We design two experiments on a 10m diamond path over 60s. We compare the average tracking performance of C1 and C2 on their own as well as the trained DRLBC framework and the switched control architecture. The cumulative tracking error is dependant on duration of flight and the shape of the path. We hence use the average tracking error as the performance measure reported in this article as it gives a better indication of general performance. The first experiment tests the performance under nominal operating conditions while the second compares performance under 50% rotor faults at every 5 second interval. The rotor selection is kept random and results shown are averaged over 10 independent runs to account for this.

6.1 Experiment 1: Nominal Control

Table D.3 shows the average trajectory loss for the 4 tested controllers. DRLBC performs *between* the performance of C1 and C2 which is exactly as expected. The switched architecture performs exactly as C2 since no fault switch to C1 is triggered. This shows if all low-level controllers produce a similar output signal, blending to any proportions will have little effect and nominal performance can be maintained even with a constantly changing blend weight vector.

Table D.3: Experiment 1: Average Tracking Accuracy under nominal conditions.

Controller	Average Tracking Error
C1	48.88cm
C2	48.86cm
DRLBC	48.87cm
Switched Control	48.86cm

6.2 Experiment 2: Rotor Faults of 50% at 5s intervals

Experimental results are shown in Table D.4. We firstly investigate the performance of C1 and C2 under rotor faults. The *aggressive* C1 controller performs better (50.1cm avg. error) as it reacts to faults with a higher response due to the higher PID gain parameters. This means C1 is more robust to rotor faults and validates the selection

Table D.4: Experiment 2: Average Tracking error under 50% rotor loss of efficiency.

Controller	Average Tracking Error
C1	50.1cm
C2	60.33cm
DRLBC	50.3cm
Switched Control	55.58cm

as a fault controller for the switched architecture. C2 performs with greater deviations (60.33cm avg. error) which is expected since the controller reactions to faults are smoother, hence taking longer to stabilize and increasing the overall error.

The switched architecture was able to utilize some of the benefits of the aggressive controller after the deviation had crossed a threshold and a fault was identified. With 55.58cm tracking accuracy it performs close to halfway between the low-level controller performances. Although this is an improvement to C2 the delay in the FDI unit still has major implications for the overall executed trajectory. The most important part in successfully stabilizing a rotor fault is the speed of reaction due to the highly unstable dynamics of a quadcopter.

The blended architecture performs comparable to C1. Given that the agent is bound between the low-level controllers the optimal result that could have been attained from the training is 50.1cm, C1s performance. Figure D.5 shows the blend weight vector applied during a sample run of this experiment. Most of the time both controllers are used to control the system to some degree. It is interesting to point out that the agent did not converge to only use C1 as one would expect since it is the better performing of the underlying controllers. We attribute this to an improved blended controller output compared to the individual PID controllers. Figure D.6 shows the over and undershooting responses of the C1 and C2 PID controllers as they stabilize on the reference signal after a rotor fault. The blended control output, shown in red, stabilizes faster and smoother than the low-level controllers can on their own which has a positive effect on the overall tracking performance.

The output of C1 and C2 are both not optimal to stabilize the system but the agent is able to use them to synthesise an improved response that is more robust to oscillations around the set-point and stabilizes quicker. This also plays a factor in the largely positive reward seen in Figure F.2 as this shows the agent has successfully learned to produce a less oscillating response signal than the underlying controllers while still being able to utilize the more aggressive responses from C1 to stabilize rotor faults.

To highlight the performance difference we provide Figure D.7 which shows the

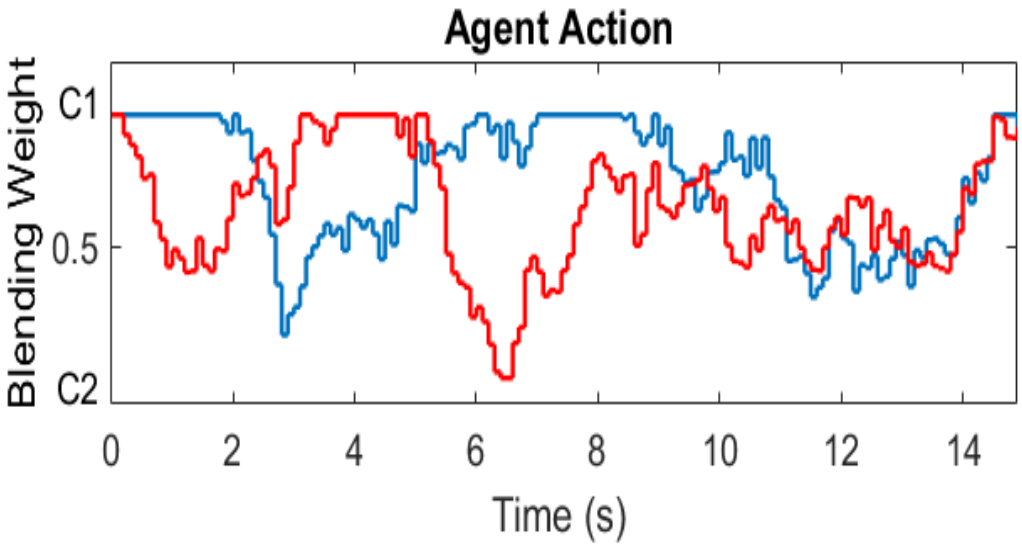


Figure D.5: Agent actions under rotor faults for blending Roll (Blue) and Pitch (Red).

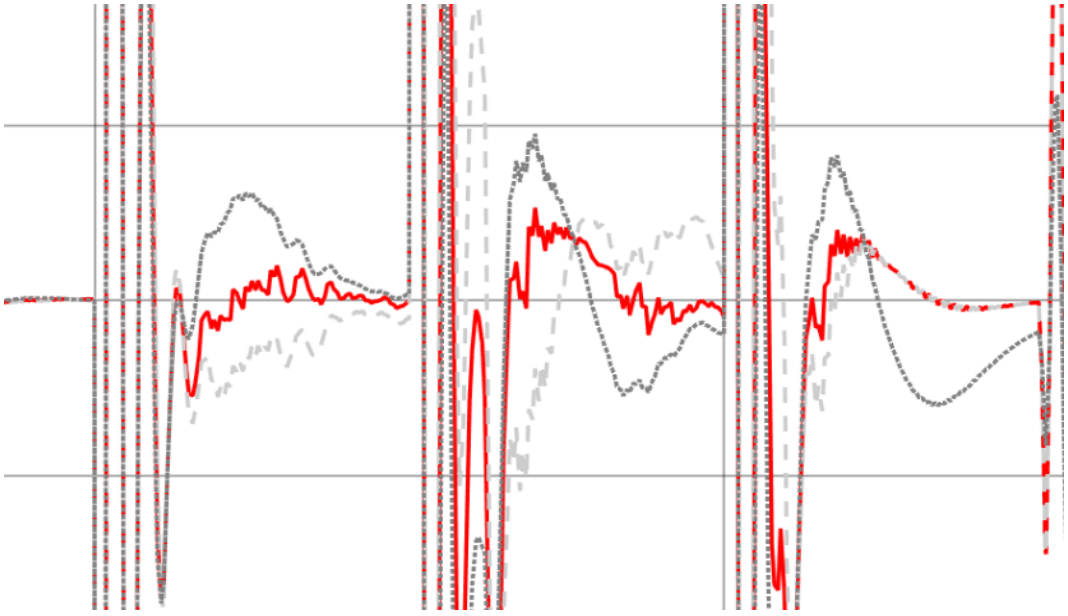


Figure D.6: Grey: Low-level controller responses and Red: Blended signal through DRL. The blended signal is able to stabilize around the set-point quicker.

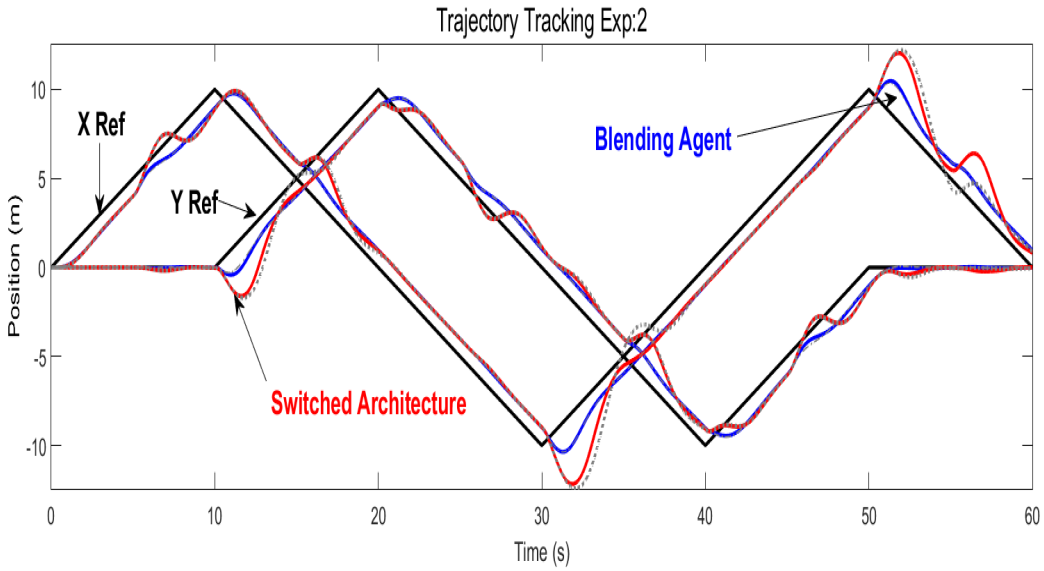


Figure D.7: Trajectory under rotor failure for different control architectures. Red: Switched Architecture, Blue: Blended DRL Architecture. Black: X and Y Reference.

executed trajectories from a sample run of this experiment for DRLBC (Blue) and switched (Red) architectures. The effect of the rotor faults can be clearly seen as abrupt deviations from the X and Y reference paths (Black) at 5 second intervals.

7 Conclusion

In this article we presented a novel fault tolerant control architecture with the ability to learn unknown fault tolerance. This was achieved through the reliance on existing low-level control mechanism and an abstract application of deep learning to the high-level control task. Using blended control, this architecture exploits a new way to integrate deep learning algorithms to well known hierarchical control architectures. Low-level controllers are designed based on the type of response they provide under fault conditions rather than for specific fault conditions. The FDI unit and it's associated delays are replaced with a DDPG agent that generically identifies degrading performance on a task in real time and learns to optimize for the new conditions. This architecture was implemented on a quadcopter trajectory tracking task under rotor loss of effectiveness faults for which no identification or pre-defined optimal control mechanism exists. We validated the effectiveness of the approach through training and experimentation showing the blended controller is able to synthesize an improved control signal that handles the unknown fault as well as improve oscillations around the reference value. The size of the learning problem is greatly reduced compared

to current state of the art approaches to learn direct control of state space to control signals. We showed the presented approach can track a trajectory under rotor faults more accurately than a switched architecture with the same low-level controllers.

Future work includes the training for several faults simultaneously and different application domains. More complex blending functions or different neural network designs pose a large frontier for exploration for the research community which can provide more new ways for control systems to learn to adapt to unknown faults and drive the application of autonomy to large scale systems.

Paper E

Deep Reinforcement Learning and Randomized Blending for Control under Novel Disturbances

Abstract

Enabling autonomous vehicles to maneuver in novel scenarios is a key unsolved problem. A well-known approach, Weighted Multiple Model Adaptive Control (WMMAC), uses a set of pre-tuned controllers and combines their control actions using a weight vector. Although WMMAC offers an improvement to traditional switched control in terms of smooth control oscillations, it depends on accurate fault isolation and cannot deal with unknown disturbances. A recent approach avoids state estimation by randomly assigning the controller weighting vector; however, this approach uses a uniform distribution for control-weight sampling, which is sub-optimal compared to state-estimation methods. In this article, we propose a framework that uses deep reinforcement learning (DRL) to learn weighted control distributions that optimize the performance of the randomized approach for both known and unknown disturbances. We show that RL-based randomized blending dominates pure randomized blending, a switched FDI-based architecture and pre-tuned controllers on a quadcopter trajectory optimisation task in which we penalise deviations in both position and attitude.

1 Introduction

Enabling agents to act autonomously is a significant challenge, with many unsolved tasks. One unsolved task is enabling a system to operate safely in novel scenarios, since it is impossible to pre-compute controllers for all disturbances (e.g., faults or external disturbances like wind), let alone scenarios that cannot be predicted during design-time.

We focus on systems for which real-time control (RTC) is important. In particular, we use a quadcopter [91] as our running example. A quadcopter is a well-studied unmanned aerial vehicle that use four propellers to maneuver. These vehicles have fewer actuators than degrees of freedom, and hence are called under-actuated. Improper design of RTC can lead to crashing. Current state of the art quadcopter control utilizes a cascading PID-architecture [100].

Two approaches for RTC have been developed. The *control-theoretic approach* uses model-based methods to develop controllers, and relies on state estimation to compute a controller appropriate to a given state. The *AI-based approach* uses model-free methods, and relies on machine learning to estimate control parameters for given operating conditions.

Both approaches have strengths and weaknesses. The *control-theoretic approach* can generate controls with precise guarantees, but state estimation introduces latency into applying controls, and typically these model-based solutions require *a priori* knowledge of all potential states. The *AI-based approach* requires time to learn controls for novel scenarios, so cannot respond in real-time to such situations.

We propose an approach that is based on Weighted Multiple Model Adaptive Control (WMMAC), a state-of-the-art passive adaptive control technique that blends the outputs of a set of low-level controllers. WMMAC was first introduced in [27] as an improvement of discrete switching-based multiple model approaches. Blending avoids the control oscillations that are problematic in other switching methods, e.g., discrete switching [101] and sliding-mode control [102], since all control assignments consist of a blend of multiple controllers so switching is inherently less abrupt.

Most blending algorithms rely on (a) a priori controller specifications and (b) identification mechanisms (e.g., Kalman-filters) to estimate the probability of a known disturbance and adjust the blending proportions accordingly. Few blending algorithms for unknown disturbances (where identification is impossible) exist; for real-time controllers, delays or miss-identification of faults can be catastrophic.

One recent WMMAC approach for stabilizing systems subject to novel disturbances uses a randomized blending (RB) distribution over all controllers without performing state estimation [103]. The weakness of this approach is that it uses a uniform distribution for controller sampling, which leads to sub-optimal control in comparison to an ideal optimal controller.

This article proposes a stable, real-time controller for novel disturbances that does not require on-line fault detection and diagnosis. We extend randomized blending (RB) control with reinforcement learning (RL) [98] to design an agent that learns how to adapt the blending distributions depending on the system state and environmental conditions. We use deep reinforcement learning to teach an agent how to parameterize the RB distribution depending on the scenarios encountered. This approach improves on traditional RL methods for real-time control (e.g., [93, 95, 104]) as the RB guarantees stability for the exploration phase of novel scenarios, whereas no stability guarantees exist for any existing RL-based controller in novel scenarios. Further, the randomization inherent in control execution provides an excellent basis for RL exploration, as it provides good coverage of the control space under the novel situation.

Our contributions are as follows.

- We present a novel real-time control system based on WMMAC with randomized blending and deep reinforcement learning. The agent is trained to learn the optimal randomized blending distribution offline for known faults given a static controller set. We show the trained framework is able to maintain control for unknown disturbances by shifting the randomized blending distributions accordingly in real time.
- We demonstrate our approach using a trajectory-following task for a quadcopter, by comparing its path- and attitude-deviation performance against that of two hand-tuned controllers, a (non-learning) randomized approach, and a traditional switching controller. We show that a neural-network based high-level controller trained for two fault conditions, abrupt rotor faults and attitude sensor noise, outperforms the other architectures under unknown disturbance conditions including wind-gusts, position noise and a combination of all known and unknown faults.
- The proposed fault-tolerant control scheme does not require an online fault-detection step.

The paper layout is as follows: Section 2 outlines various relevant control architectures and deep reinforcement learning. We introduce the generic Deep Reinforcement

Learning Randomized Blended Control (DRLRBC) architecture in section 3 and the general quadcopter model in section 4. Section 5 gives the implementation and training of DRLRBC on a quadcopter followed by experimental evaluation and conclusion in section 6 and 7 respectively.

2 Background & Related Work

This section discusses relevant background and control architectures explored in this article. We introduce manually-tuned controllers and then some hierarchical control frameworks. We are interested in comparing different high-level frameworks that use the same low-level controllers.

(1) Manual-tuning is widely done but balancing the system behaviour for different operating scenarios during the tuning process is extremely complex. Sub-optimal system controllers that exhibit unwanted behaviour during specific operating conditions are common. In this article two manually tuned controllers are designed to exhibit a specific behaviour, referred to as *C1* and *C2*. The low-level controller set for any high-level control architecture compared in this article will always consist of *C1* and *C2*.

(2) Discrete Switching is a hierarchical control architecture based on a set of low-level system controllers tuned for specific operating conditions. A large amount of work has been done on hierarchical control architectures [89, 97]. A high-level controller identifies changes in operating conditions such as faults and discretely switches control between the system controllers using a switching function. A drawback of this architecture for novel disturbance scenarios is that no pre-defined controller exists as controller design requires *a priori* knowledge of the operating conditions. By definition the performance of the pre-defined controllers is inherently unknown for novel scenarios, but one controller must maintain system control which leads to unknown performance.

(3) Weighted Multiple-Model Adaptive Control (WMMAC) uses a high-level controller to blend the inputs of a set of low-level system controllers, each of which is defined for a specific operating condition. Formally:

Definition 16 (WMMAC). *Given a collection of controllers* $\Omega = \{\omega_1, \cdots, \omega_m\}$ *and a control distribution vector* $\boldsymbol{\varphi} = \{\varphi_1, \cdots, \varphi_m\}$*, the blended control signal is given by a weighted combination* $\vec{\omega} = \sum_i \varphi_i \omega_i$ *such that: (1)* $\forall_i, \omega_i \in \Omega, 0 \leq \varphi_i \leq 1$*, and (2)* $\sum_i \varphi_i = 1$

Constraints (1) and (2) ensure that the blended control signal is bound *between* the low-level controller outputs. In [105] the stability of WMMAC is proven for a discrete time stochastic plant and [106] discusses a systematic distribution of controllers for MMAC. A major drawback of the WMMAC approach is that computing blend weights relies on FDI techniques, which are dependent on *a priori* knowledge about the disturbances. Further, detection and isolation may be inaccurate and takes inherently takes time, which can be problematic for RTC.

(4) Randomized Blended Control (RBC) is a randomized version of WMMAC. RBC avoids the estimation phase of WMMAC, and uses randomization to estimate a blend distribution by sampling uniformly over the space of all low-level controller weights. [103] show that, given unknown disturbances, RBC can stabilize a system, and that its performance for known scenarios converges to the performance of the optimal low-level controller.

Control of multiple-model systems using mixing has been shown to be stable for situations in which the (fixed) unknown parameter set of the plant is assumed to lie in the convex hull of the control parameters of the multiple models [107]. This has recently been extended to the case of systems with an unknown varying parameter set [108]. Using these notions, [103] also show stability of the randomized approach, bounded by the convex hull of the available controllers.

(5) RL-based control. We now introduce our approach, which extends the *Non-Learning (NL)* RBC with an architecture based on Deep Reinforcement-Learning (DRL), which we call Deep Reinforcement-Learning RBC (DRLRBC). DRL [11] has successfully been able to learn accurate controllers for complex systems such as quadcopters and cars [109, 93]. Further, although optimal control and RL have been developed in different communties, they are both capable of solving the same optimal control task [9]. One weakness of RL-based control approaches is the training-time required to learn control of the system for each of several operating conditions, and the complexity of defining an accurate reward function. Currently, no approach exists that can provide stability guarantees under novel disturbances. This article extends the RBC approach with deep RL in order to learn a controller weight distribution that is tailored to the observed environmental/fault conditions. To simplify the training process, we use for our training data a restriction of the entire state space of the quadcopter, namely attitude loss and low-level controller outputs.

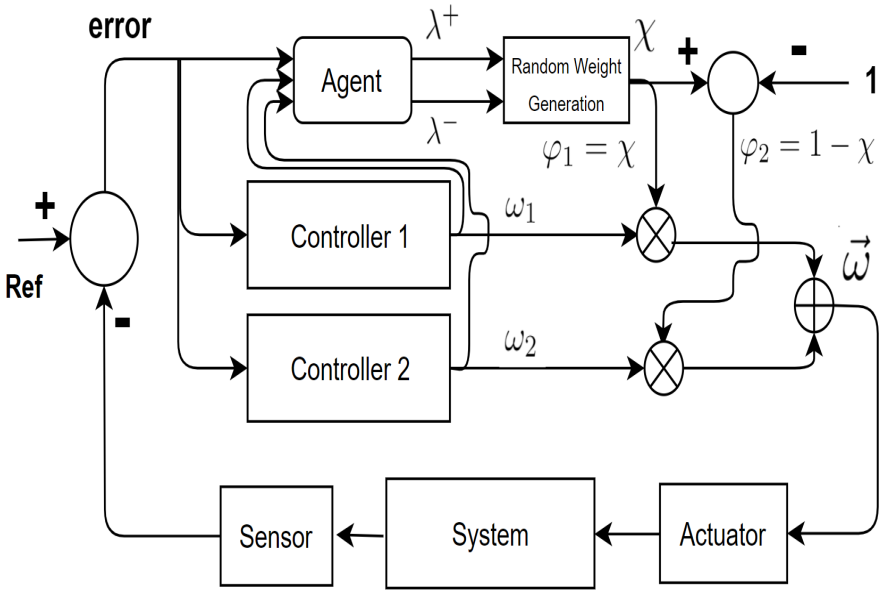


Figure E.1: The DRLRBC architecture (with two controllers).

3 Deep Reinforcement Learning Randomized Blended Control

We will give a generic overview of the presented architecture, referred to as Deep Reinforcement Learning Randomized Blended Control (**DRLRBC**). An architecture diagram showing a two controller example can be seen in Figure E.1. We will discuss the architecture in four parts: (1) Low-Level Controllers, (2) Performance Estimate, (3) Blending Function and (4) High-Level Controller.

3.1 Low-Level Controllers

In this article, we will restrict the low-level controllers to PID controllers, which are an industry-standard way of controlling automatic systems [91]. Fault tolerant control architectures tune low-level controllers for different operating conditions to achieve fault tolerance under those conditions [97]. Although MPC and LQR controllers have been applied to quadcopters, they are typically designed around a fixed operating point and require extensive modelling effort [100]. The investigation into these controllers is planned future work and beyond the scope of this article. The set of low-level controllers outputs is denoted $\Omega = \{\omega_1, \cdots, \omega_N\}$. Figure 1 shows an example architecture with two controllers.

3.2 Performance Estimate

We define $\delta = [\mathcal{P}, \mathcal{E}]$, where $\mathcal{P}$ indicates overall system performance and $\mathcal{E}$ the current control effectiveness. These measures are system- and task-dependent and give the agent a real-time measure of how well the task is being achieved.

3.3 Blending Function

In developing DRLRBC we must make additional modifications to RBC. We adapt the RBC input parameters to allow an external input Λ, referred to as the *Randomization Bounds Vector (RBV)*, to control the range from which φ_i is sampled. Λ is a set of tuples $[\lambda_i^-, \lambda_i^+]$, one tuple for each low-level controller $\omega_i \in \Omega$, indicating the range from which the randomized blend weight φ_i is sampled. More precisely :

Definition 17 (Bounded Randomized Blending)**.** *Given a collection of controller outputs* $\Omega = \{\omega_1, \cdots, \omega_m\}$ *and Randomization Bounds Vector* $\Lambda = \{[\lambda_1^-, \lambda_1^+], \cdots, [\lambda_m^-, \lambda_m^+]\}$*, the control distribution vector* $\varphi = \{\varphi_1, \cdots, \varphi_m\}$ *is randomly sampled such that for the weighted combination* $\vec{\omega} = \sum_i \varphi_i \omega_i$ *the following constraints hold: (1)* $\forall_i, \omega_i \in \Omega, \lambda_i^- \leq \varphi_i \leq \lambda_i^+$*, (2)* $\sum_i \varphi_i = 1$ *,and (3)* $0 \leq \lambda_i \leq 1$

By introducing a bound on the range from which random blend weights are sampled from, the space of combinations of controllers can be explored while still relying on the stability guarantees provided by RBC under novel disturbances.

3.4 High-Level Controller

In the presented architecture the high-level controller is implemented using a deep neural network as they are known to be excellent function approximators. Neural networks have continuous observation and action spaces which remove the inherent FDI delay experienced by traditional switched systems.

The observation vector for DRLRBC can generically be defined as: [δ , Ω]. The agent's action output is the Randomization Bounds Vector, Λ. So the goal of the agent is to learn the mapping: $[\delta, \Omega] \rightarrow \Lambda$ that optimizes the performance of the system on a given task. A more detailed example of observation vector and action space will be given in section 5.

3.5 Architecture comparison

We contrast the presented DRLRBC architecture to current state of the art approaches. The deep learning algorithm is applied to an abstracted high-level task while relying

on existing low-level system control mechanism to control the vehicle. This reduces the overall complexity of the learning task as nominal system control is already established which usually takes a large number of learning iterations and fine tuning an accurate reward function to achieve. The direct mapping of state space to motor commands is a highly complex function. Every additional input to the deep learning algorithm increases the size of the space that is explored which is a reason for the long convergence times experienced by the application of deep learning to control tasks. DRLRBC uses a measure of task performance to direct how influential each controller is in the applied signal, which is a much smaller problem to learn.

WMMAC enables FTC with respect to a pre-defined set of (partial) faults, but at the expense of (a) tuning a controller for each fault and (b) using FDI to isolate the fault magnitudes prior to controller allocation. In contrast, our approach does not require *a priori* knowledge of any faults or FDI for fault isolation; our randomized controller assignment (as tuned to actual faults via RL) handles the FTC.

The framework we propose aims to estimate the optimal blended control signal given a static pre-defined controller set under situations it was not trained for. Since we are calculating a convex combination of controller outputs, the overall range of the blended signal is limited to between the low-level controller outputs.

4 Quadcopters

Quadcopters are unmanned aerial vehicles that use four propellers to maneuver and have gained increased attention in the research community in recent years. These vehicles have only four actuators used to control six variables, the coordinates x, y, and z, and the roll, pitch, and yaw angles of the quadcopter, denoted ϕ, θ, and ψ, respectively. The dynamic equations of a quadcopter are complex, due to the highly coupled state-space. Due to space limitations, we give a brief summary of quadrotor dynamics and details of how rotor faults and wind disturbances are represented, and refer the reader to [91] for details.[1]

We define the dynamics of the quadcopter in the non-linear state space form

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}) + g(\boldsymbol{x})(1 - \varsigma)\boldsymbol{u}(t), \tag{E.1}$$

where $\boldsymbol{x} = [x\ \dot{x}\ y\ \dot{y}\ z\ \dot{z}\ \phi\ \dot{\phi}\ \theta\ \dot{\theta}\ \psi\ \dot{\psi}]^T$ is the state vector, control input $\boldsymbol{u}(t) =$

[1]The MATLAB simulation codebase necessary to run the experiments in this article is available under github.com/YvesSohege/IFAC20-Simulation.

$[U_1\ U_2\ U_3\ U_4]^T = \varrho(\upsilon_1\ \ \upsilon_2\ \ \upsilon_3\ \ \upsilon_4)$, ϱ is a non-linear function in the angular velocity of motor i, and we denote a multiplicative fault model with parameter $0 \leq \varsigma_i \leq 1$ for $i = 1, ..., 4$, where ς_i = 0 corresponds to nominal function and ς_i = 1 to total failure. The wind is generated as a drag force that acts on the body frame ε.

Quadcopters have been shown to be able to maintain flight even after the complete loss of one or more rotors under specific conditions [92]. The standard way to achieve trajectory tracking is by employing a cascading PID controller structure, which is the approach we consider in this article. Position controllers generate the required attitude reference to execute the trajectory. Roll and Pitch attitude controllers generate the required motor commands to attain the attitude. Blended control is only applied on the attitude controllers, not on the position controllers.

Several applications of deep learning for quadcopters exist, which mostly focus on learning the direct control mapping of state space to motor commands [93, 94, 95]. This usually requires significant training data and complex fine-tuning of the reward functions to achieve the desired behaviour.

The application of deep learning for FTC of a quadcopter has been achieved by learning a complementary controller that adjusts the nominal controller output during a rotor fault [96]. The success of this approach compared to other adaptive control strategies is attributed to the continuous output of the neural network and removal of the FDI unit to identify when a correction to the nominal controller is needed.

The task of the Quadcopter in this work is focused on **Trajectory Tracking**. A trajectory is a temporally-indexed set of coordinates in 2D or 3D, denoted $\boldsymbol{\zeta}(k)$. We denote the reference (desired) trajectory as $\boldsymbol{\zeta}^R(k)$, and the executed trajectory as $\tilde{\boldsymbol{\zeta}}(k)$. The goal of a trajectory tracking task can be defined as minimizing the Total Trajectory Loss:

Definition 18 (Total Trajectory Loss)**.** *We can represent the total trajectory loss as a difference function between reference and executed trajectories, i.e.,* $\mathcal{L}_{0:T} = \sum_{k=0}^{T} \parallel \boldsymbol{\zeta}^R(k) - \tilde{\boldsymbol{\zeta}}(k) \parallel$ *for a trajectory over time points* $k = 0, \cdots, T$.

Given the Total Trajectory Loss over the T time-steps, the current trajectory loss at time t can be defined as $\mathcal{L}_t$. Similarly, we introduce a second performance metric to measure the attitude tracking error, $\mathcal{A}_{0:T}$, as a loss function between reference attitude $\boldsymbol{\Delta}^R(k)$ and actual attitude $\tilde{\boldsymbol{\Delta}}(k)$ which is omitted due to space constraints. For simplicity we will focus on 2D (x, y) trajectory tracking but the approach extends to 3D trajectories with minimal changes. We do not apply blending on the rotational ψ controller of the quadcopter as this does not effect a 2D trajectory.

5 Quadcopter Implementation and Training

In this section we will describe the implementation of the DRLRBC architecture on a Quadcopter MATLAB simulation as well as the training details of the agent.[2]

5.1 Low-Level Controllers

In this article, we will use two controllers for the roll and pitch attitude of the quadcopter. Each axis will have a C1 and a C2 controller with different tuning which can be seen in Table G.2. Since a quadcopter is symmetric we can use the same tuning for both axis. By changing the PID gain parameters, we change how the quadcopter responds to different situations. The controllers were hand-tuned to nominal operating conditions under which they perform identically in terms of trajectory loss. This is highlighted experimentally in Table E.4 Exp1, in Section 6. However C2 has a higher proportional gain which allows it to respond more aggressively to abrupt disturbances than C1. We also purposely add slight oscillations around the reference when tuning C2 by slightly lowering the derivative gain.

Table E.1: PID parameters for low-level controller tuning used.

	$\mathcal{P}$	$\mathcal{I}$	$\mathcal{D}$
C1 (Smooth)	2	1.2	1.3
C2 (Aggressive)	4	1.5	1.2

5.2 Performance Estimation

We use the current x and y trajectory error, denoted δ_x and δ_y respectively, as a measure of how well the system is performing overall on its task. Blended control is being applied on two axis of control and each needs a measure of effectiveness. For this we select the current roll and pitch attitude error, denoted δ_ϕ and δ_θ respectively. We can hence define the full performance estimation output for the Quadcopter as $\delta = [\delta_x \quad \delta_y \quad \delta_\phi \quad \delta_\theta]$. These metrics allow the agent to judge how good control is for the current scenario and adapt it to maximize the performance without explicit knowledge of the operating conditions.

5.3 Blending Function

Since this article focuses on blended control of a controller **pair** a simplification can be made to Equation 17. By relying on Constraint (2) of Equation 17 which forces the

[2]Real-world flights are beyond the scope of this article.

sum of both controller weighting to be 1, after randomly generating the first weight, the second can simply be obtained by subtraction from 1. This simplification is indicated using χ in Figure E.1 and reduces the size of Λ by half.

5.4 High-Level Controller

An actor-critic DDPG network structure is used to generate the randomized bounds vector Λ. Both actor and critic take in the same observation vector which we define in full detail as $[\delta_x \ \ \delta_y \ \ \delta_\phi \ \ \delta_\theta \ \ \omega_1^\phi \ \ \omega_2^\phi \ \ \omega_1^\theta \ \ \omega_2^\theta]$, where ω_1^ϕ indicates C1 for the roll axis, ω_2^θ is C2 for pitch and so forth.

Through the additional simplification introduced to the Blending Function the agents action output can simply be defined as $[\lambda_\phi^- \ \ \lambda_\phi^+ \ \ \lambda_\theta^- \ \ \lambda_\theta^+]$, indicating the lower and upper bounds of the randomized blending ranges for ϕ and θ controllers respectively. We give a brief overview of the neural network architecture. The actor network is defined by three fully connected layers separated by ReLU (Rectified Linear Unit) layers. Finally a hyperbolic tangent layer with output size 4 is used to naturally enforce the blended control constraints, bounding the agent action space between 0 and 1. The critic network has two paths, one for the observation vector and the other for the actor output which are joined after two and one fully connected layer respectively for each path. All fully connected layer contains 32 neurons in this implementation.

5.4.1 Training Conditions

We use random rotor loss of effectiveness (LOE) and noise on the attitude sensors of the quadcopter as training conditions. All rotor faults in this article are of magnitude 10%. To vary the training conditions we set the time a rotor fault can occur randomly, with every time-step having a probability of spontaneously losing angular rotor velocity of 10% for one time-step. This has an impact on both attitude and trajectory. The noise is modelled as random white noise applied to roll and pitch state, ϕ and θ respectively, and is defined simply by its magnitude.

5.4.2 Reward function

We define the reward function in terms of Total Trajectory and Attitude Loss. The agents goal is to maximize its reward function so we define the reward obtained by the agent as:

$$\mathcal{R}(t) = -(|\delta_x(t)| + |\delta_y(t)|) - (|\delta_\phi(t)| + |\delta_\theta(t)|)$$

which represents the sum of trajectory and attitude loss at time t. We negate this to

allow the agent to maximize the reward. This reward function allows the agent to learn how to improve the trajectory and attitude tracking performance regardless of operating conditions. For training we use a straight line path of 30 meters executed over 30 seconds. The quadcopter was trained over 3000 episodes and average reward is calculated over 50 episodes. The agent converged after around 1500 episodes. We define other relevant training parameters used in Table E.2 and refer the reader to the github repository provided in section 4 for further details.

Table E.2: DDPG Training parameters used

Parameter	Value
Discount factor	0.99
Initial learning rate of the critic	0.01
Initial learning rate of the actor	0.025
Batch size	32
Replay buffer size	10000
Training steps of an episode	300
Number of episodes	3000

6 Experiments

We empirically compare the simulated performance of the presented control systems on a number of scenarios, including known and novel disturbances. We do not consider faults that would cause catastrophic failure. Since the agent was trained on abrupt rotor loss of effectiveness and attitude noise, we classify these as the known disturbances. In addition wind gusts and position noise are tested and we classify these as novel disturbances. We compare the two baseline controllers, C1 and C2, with three high-level control architectures which utilize C1 and C2 to improve control, namely DRLRBC, Non-Learning RBC and Switching. We define the switching condition for the switched architecture as a trajectory deviation of 10% above nominal tracking error. C1 is selected as the nominal controller and C2 as the fault controller. We use a diamond shape path with diagonal length of 10 meter starting and ending in the centre over 60 seconds with a sample time of 0.1s. Since all disturbances are modelled with randomness we present average results taken over ten runs on each experiment.

The full list of experimental conditions can be found in Table E.3. We test nominal conditions, three known disturbances (rotor loss of effectiveness, attitude noise and both) and three unknown disturbances (position noise, wind gusts and all faults). Magnitude of disturbances (rotor and wind) is given followed by disturbance trigger probability at any time step. E.g: *10% error (30%)* indicates every time-step has a

Table E.3: Experimental Disturbance Details.

Exp #	Att. Noise	Pos. Noise*	Rotor LOE	Wind Gusts*
Exp1	-	-	-	-
Exp2	0.02	-	10% error (10%)	-
Exp3	-	-	10% error (30%)	-
Exp4	0.05	-	-	-
Exp5	-	-	-	0-10m/s (30%)
Exp6	-	0.05	-	-
Exp7	0.02	0.02	10% error (10%)	0-10m/s (10%)

30% chance of triggering a 10% fault.

6.1 Experimental Results

We evaluate the performance on each experiment using the average Total Trajectory Loss and Total Attitude Loss of each control system over 10 independent runs. We present the complete set of results in Table E.4. On average across all experiments the presented architecture is able to outperform all other control systems in terms of attitude and trajectory tracking accuracy.

6.1.1 Known Disturbances

The training conditions included rotor faults and attitude noise. DRLRBC is able to significantly outperform all other control systems for trajectory accuracy for all three experiments showing the agent successfully learned to improve its performance under known disturbance conditions. Under heavy attitude noise C1 is able to track the attitude slightly more accurately than the DRLRBC architecture but at the expense of trajectory loss.

6.1.2 Unknown Disturbances

Positional noise and wind gusts are used to test the control systems under unknown scenarios. DRLRBC performs second best across all 3 experimental conditions in terms of trajectory accuracy showing a comparable performance. In terms of attitude tracking DRLRBC is able to perform best under wind gusts and perform comparable to C1 and NL-RBC under the other two scenarios.

Table E.4: Experiment list showing Total Attitude Loss (in radians) and Total Trajectory Loss (in meters) for the compared control systems. Best performing in bold text.

	Total Attitude Loss (rad)					Total Trajectory Loss (m)				
Exp #	DRLRBC	C1	C2	NL-RBC	Switched	DRLRBC	C1	C2	NL-RBC	Switched
Exp1	**0.68**	1.23	0.78	0.88	1.23	48.98	**48.79**	**48.79**	**48.79**	**48.79**
Exp2	**30.43**	42.08	43.71	35.1	41.08	**50.88**	98.51	59.31	55.71	56.20
Exp3	**19.98**	43.74	44.62	25.59	30.96	**50.60**	127.42	60.77	57.98	65.38
Exp4	33.16	**30.62**	42.07	34.39	34.90	**55.17**	57.87	56.36	56.21	63.98
Exp5	**10.98**	17.5	14.30	12.62	12.35	52.07	64.70	**51.67**	53.42	53.72
Exp6	16.73	**15.03**	23.01	16.29	20.89	57.91	57.97	**57.15**	58.39	57.67
Exp7	29.14	36.07	42.14	**27.43**	28.17	55.85	79.28	60.49	**55.66**	56.89
Average	**20.16**	26.62	30.09	21.76	24.23	**53.07**	76.36	56.36	55.16	58.66

7 Conclusion

In this article, we presented a novel hierarchical control architecture based on weighted multiple model adaptive control, deep reinforcement learning and randomized blending. The presented architecture is tested on a quadcopter trajectory tracking simulation, and trained under rotor loss of effectiveness and attitude noise. We compare the presented architecture to a non-learning randomized approach, a standard switched architecture, and the underlying controllers working individually. We showed that, averaged across all experiments, the presented architecture outperforms all other baselines in terms of both trajectory tracking and attitude tracking under *known* and *unknown* disturbances. This extends the field of fault tolerant control by providing a novel way to apply deep reinforcement learning to high-level control tasks. In future work, we plan to explore additional learning mechanisms, such as unsupervised learning and a larger set of low level controllers.

Paper F

Neural-Symbolic Fault Tolerant Control for Quadcopter Trajectory-Following Tasks

Abstract

Many fault-tolerant control (FTC) applications are moving from a traditional model-based approach to one that learns the FTC actions, termed data-driven FTC. While data-driven FTC does not require models, it requires a significant amount of nominal and fault data, and training may be expensive. We develop an architecture for combining model-based and data-driven FTC that aims to make use of the best aspects of each approach. The architecture learns a supervisory controller for switching weights across multiple model-based low level controllers. We demonstrate our approach on learning trajectories for a quadcopter that must follow a safe region even though it experiences rotor faults. We empirically show that our hybrid learning approach converges to safely follow given trajectories, whereas a purely data-driven approach requires significantly more training to converge than the hybrid approach (if it converges at all).

1 Introduction

Developing models for fault tolerant control of complex systems is challenging, in that model-based approaches suffer from models being incomplete and often not fully suited to real-world, dynamic environments. Data-driven approaches, on the other hand, require significant amounts of data, are time-consuming to train, and are limited to relatively simple systems. In this paper, we propose an approach that integrates model-based and data-driven control methods, in an attempt to leverage the best of both methods. We directly address how to integrate model-based inference and learning, known as neural-symbolic learning [110] or model-based/physics-guided learning [111].

Currently a great deal of research is directed towards this topic. The majority of work in neural-symbolic diagnosis at present focuses on using physics and deep learning in condition-monitoring of rotating machines, e.g., [112]; this approach is called physics-based preprocessing in [111]. This work uses physical principles of machinery and signal processing to assist with deep learning methods for diagnosing faults in simple machines, e.g., [113, 114].

A second approach, less common in diagnosis, is constraining deep learning with physical principles [115, 116]. Here, physics-based equations are used as an additional regularization term in the loss function of the neural networks. This approach has been used for diagnosis [117, 118] and prognosis [119].

We adopt an approach that is different to either of these methods: we decompose the fault-tolerant control (FTC) task into high-level and low-level inference, and use well-known physical controllers for the low-level control, and learning for the high-level FTC. This approach enables us to employ well-understood PID controllers with well-defined input parameters for low level control, and we then *learn to tolerate faults* using a high-level controller. Our approach is significantly simpler than methods that require learning all control parameters for FTC applications, since we reduce the number of parameters to be learned to a small number of high-level (abstracted) parameters.

We demonstrate our approach on a quadcopter that is subjected to rotor faults of various magnitudes.

Our contributions are as follows.

- We develop an architecture for combining model-based and data-driven control for FTC that aims to make use of the best aspects of each approach;

- We demonstrate our approach on learning safe trajectory following for a quadcopter that has significant faults in its rotors;
- We empirically show that our hybrid learning approach converges to safely follow given trajectories whereas a purely data-driven approach requires significantly more training to converge than the hybrid approach (if it converges at all).

This article is organised as follows. Section 2 describes the FTC approach from purely model-based and data-driven frameworks, and how we integrate these for a hybrid approach. Section 3 introduces the quadcopter and the faults we inject. Section 4 outlines the experiments that we perform in trajectory following given rotor faults. Section 5 presents our empirical results. We conclude in Section 6.

2 Approach

This section describes the general issues concerning hybrid learning for fault-tolerant control (FTC).

FTC is the task of using an anomalous observation $\boldsymbol{s}_t \in S$ of a system at time t (indicating a fault state) to generate a control $\mathbf{a}_t \in A$ that drives the system to a desired ("safe") state $\boldsymbol{s}_{t+1} \in S$. We can denote this as a function $f^{\theta} : \boldsymbol{s}_t \times \boldsymbol{\theta} \rightarrow \mathbf{a}_t$, where θ denotes the parameters of function f that drive the system to a safe state such that $f_T : \boldsymbol{s}_t \times \mathbf{a}_t \rightarrow \boldsymbol{s}_{t+1}$, with f_T representing the system dynamics also known as state transition function of a system.

2.1 Model-based FTC

The typical FTC algorithm uses a forward model to isolate a fault, and then generates a control output given that fault.

A model-based (MB) diagnosis forward model $f_T^{\boldsymbol{\theta_T}}$ maps state variables $\boldsymbol{s}_t$ and model parameters $\boldsymbol{\theta_T}$ to outputs (observable variables $\hat{\boldsymbol{y}}$): $f_T^{\boldsymbol{\theta_T}} : \boldsymbol{s}_t \times \boldsymbol{\theta_T} \rightarrow \hat{\boldsymbol{y}}_t$. Predictive modeling in a model-based approach entails calibrating model parameters $\boldsymbol{\theta_T}$ using observational data. The diagnosis process consists of using the *residual*, or difference between observed data $\boldsymbol{y}$ and predicted data $\hat{\boldsymbol{y}}$, to diagnose the fault responsible for the anomalous readings $\boldsymbol{\nu}_t$ [89].

The FTC aspect of this approach means to adapt the parameters $\boldsymbol{\theta}_{MB}$ of a control law

represented by a function f_{MB} such that

$$f_{MB} : \boldsymbol{\nu}_t \times \boldsymbol{\theta_L} \rightarrow \mathbf{a}_t, \tag{F.1}$$

where $\boldsymbol{\nu}_t$ represents the information of the diagnosed fault.

2.2 Data-Driven FTC

We formulate our data-driven approach in terms of an agent using reinforcement learning (RL). RL is a branch of machine learning concerned with the design of methods that allow an agent to learn how to solve a task by interacting with an environment. At each time step t, the agent observes the state of the environment $\boldsymbol{s}_t \in S$ and it generates an action $\mathbf{a}_t \in A$. The agent then receives information about the next state $\boldsymbol{s}_{t+1}$ of the environment and an immediate scalar reward $r_t \in \Re$. The decision made by the agent is based on a policy function $f_{ML} : S \rightarrow A$ that maps the state space to the action space.

The environment is modeled as a Markov decision process (MDP) defined as follows:

Definition 19 (Markov Decision Process)**.** *A Markov decision process is defined by a four tuple:* $M = \{S, A, T, R\}$ *where* S *represents the set of possible states that the environment can reach. The transition function* $T : S \times A \times S \rightarrow [0, 1]$ *estimates the probability of reaching state* $\boldsymbol{s}'$ *at time* $t + 1$ *given that action* $\mathbf{a} \in A$ *was chosen in state* $\boldsymbol{s} \in S$ *at decision epoch t,* $T = P(\mathbf{s}'|\boldsymbol{s}, \mathbf{a}) = P\{\boldsymbol{s}_{t+1} = \boldsymbol{s}'|\boldsymbol{s}_t = \boldsymbol{s}, \mathbf{a}_t = \mathbf{a}\}$*. The reward function* $R : S \times A \rightarrow \Re$ *estimates the immediate reward* $R \sim r(\boldsymbol{s}, \mathbf{a})$ *obtained from choosing action* $\mathbf{a}$ *in state* $\boldsymbol{s}$*.*

A task $\mathcal{T}$ is defined in this context by the tuple

$$\mathcal{T} = (R_{\mathcal{T}}, P_{\mathcal{T}}(\boldsymbol{s}_t), P_{\mathcal{T}}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \mathbf{a}_t), H) \tag{F.2}$$

where $R_{\mathcal{T}}$ is a reward function, H represents the duration of the task, and $P_{\mathcal{T}}(\boldsymbol{s}_t)$ and $P_{\mathcal{T}}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \mathbf{a}_t)$ determine the dynamic of the system for task $\mathcal{T}$.

The goal of the agent is to learn an optimal policy function $f_{ML}^{\theta_H}$ that maximizes the expected return (the cumulative sum of rewards) for a given task $\mathcal{T}$.

$$V^{f_{ML}^{\theta_H}}(\boldsymbol{s}_t) = \max_{\theta_H \in \Theta_H} V^{f_{ML}^{\theta_H}}(\boldsymbol{s}_t) \ , \ \forall \boldsymbol{s}_t \in S \tag{F.3}$$

where $V^{f_{ML}^{\theta_H}} : S \rightarrow R$ is called value function and it is defined as

$$V^{f_{ML}^{\theta_H}}(\boldsymbol{s}_t) = E\left[\sum_{i=0}^{H} \gamma^i R(\boldsymbol{s}_i, \mathbf{a}_i) | s_0 = \boldsymbol{s}_t\right] \ , \ \forall \boldsymbol{s}_t \in S \tag{F.4}$$

where $0 < \gamma \leq 1$ is called the discount factor, and it determines the importance assigned to future rewards such that it decays with time.

In deep RL, we learn a neural network model $f_{ML}^{\theta_H}$ with parameters θ_H that represents the optimal policy function which solves the task $\mathcal{T}$ such that

$$f_{ML} : \ \boldsymbol{s}_t \times \boldsymbol{\theta_H} \rightarrow \mathbf{a}_t. \tag{F.5}$$

2.3 Hybrid FTC

In our hybrid approach, we develop a hybrid model that composes a model-based and a data-driven model. We use the output of f_{MB} as an input to f_{ML}, i.e.,

$$f_{Hybrid} : \boldsymbol{s}_t \times \boldsymbol{\nu}_t \times (\theta_L \times \theta_H) \rightarrow \boldsymbol{a}_t, \tag{F.6}$$

where $f_{Hybrid} = f_{MB} \circ f_{ML}$, and $(\theta_H \times \theta_L)$ is the composite parameter space. The machine learning component will thus encapsulate the remaining unmodeled complexity of the system in a lumped form. In this approach, the model acts in a complementary fashion to enable adequate FTC.

The key is to define a compositional approach that takes advantage of each approach. The next sections shows the architecture that we use for this decomposition.

3 Quadcopters

3.1 Quadcopter Dynamics

Quadcopters are unmanned aerial vehicles that use four propellers to maneuver and have gained increased attention in the research community in recent years. These vehicles have only four actuators used to control six variables, the coordinates x, y, and z, and the roll, pitch, and yaw angles of the quadcopter, denoted ϕ, θ, and ψ, respectively. The dynamic equations of a quadcopter are complex, due to the highly coupled state-space. Due to space limitations, we give a brief summary of quadrotor dynamics and details of how rotor faults are represented, and refer the reader to [91]

for details.[1]

We define the dynamics of the quadcopter in the non-linear discrete state space form

$$\boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t) + g(\boldsymbol{s}_t)(1 - \varsigma)\mathbf{a}_t, \tag{F.7}$$

where $\boldsymbol{s}_t = [x\ \dot{x}\ y\ \dot{y}\ z\ \dot{z}\ \phi\ \dot{\phi}\ \theta\ \dot{\theta}\ \psi\ \dot{\psi}]^T$ is the state vector, control input $\mathbf{a}_t = [U_1\ U_2\ U_3\ U_4]^T = \varrho(\upsilon_1\ \ \upsilon_2\ \ \upsilon_3\ \ \upsilon_4)$, ϱ is a non-linear function in the angular velocity of motor i, and we denote a multiplicative fault model with parameter $0 \leq \varsigma_i \leq 1$ for $i = 1, ..., 4$, where $\varsigma_i = 0$ corresponds to nominal function and $\varsigma_i = 1$ to total failure.

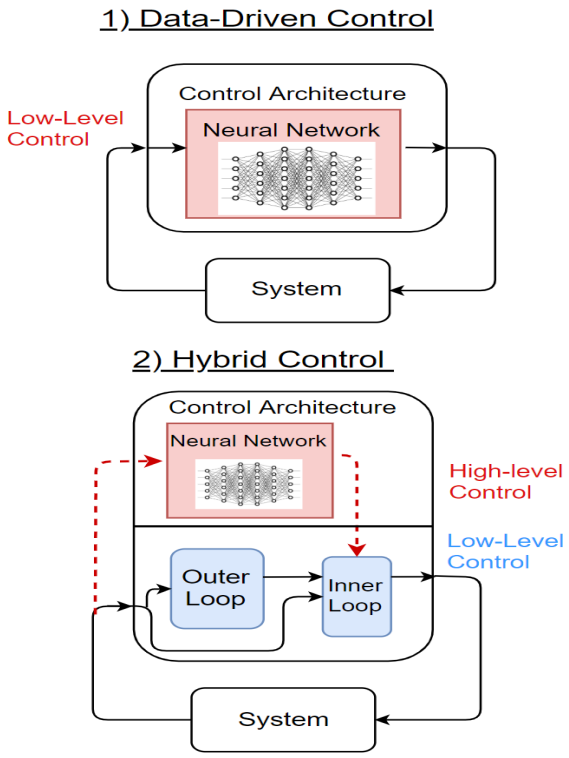


Figure F.1: Two Control architectures for quadcopters showing Data-driven control (red) and Model-based control (blue) parts.

3.2 Architecture

A quadcopter is a highly-coupled, under-actuated, nonlinear system whose control architecture can be divided into two subsystems: an attitude system and a position system. The rotational motion, also known as attitude, is independent of the position, but the translational motion is dependent on the attitude of the aircraft. Using this, we can derive the motion of the quadrotor given the position and attitude and hence define

[1]The Python-based simulation codebase necessary to run the experiments in this article is available under github.com/YvesSohege/DX20-Simulation.

inner- and outer-control loops as the attitude and position control, respectively. The physics of a quadcopter model are well understood, and an industry-standard control approach is to use PID or PD controllers for both low-level inner- and outer-loop control [91]. Fault tolerance is typically achieved by tuning additional PID controllers and using a residual-based high-level (supervisory) controller that switches between active controllers.

Figure F.1 (top) shows a data-driven architectures that can be used for quadcopter control, where a neural network learns all control parameters in a single black-box model. The issue with this learning task for complex domains is the size of the parameter space and of the data required. Figure F.1 (bottom) shows a hybrid approach, where for low-level control we use a model-based PID architecture that decomposes the system into inner- and outer-loop sub-controllers, using methods common to control theory, e.g., [120]. A high-level data-driven controller then learns to tune the low-level architecture.

When replacing any part of the lower-level control loop with data-driven approaches, several challenges arise: (1) whereas physics based stability proofs exist to ensure real-world safety, data-driven controllers do not have such proofs of stability and stability cannot be empirically guaranteed outside of the training scenarios; (2) training is usually conducted in high-end simulation environments and then transferred to a real quadcopter, which creates a well known simulation to reality gap; (3) data-driven controllers must be re-trained when the environment changes.

To address these drawbacks, we reformulate our system as a hierarchical control architecture with a subset of high- (θ_H) and low-level (θ_L) parameters, i.e., such that $\theta = \theta_H \cup \theta_L$. We use our physics-based controller to take care of low-level control and hence only the parameters of the high-level controller must be learned. Our task is thus reduced to learning θ_H and tuning θ_L. The benefit of this approach is that (1) we can use a low-level, physics-based controller whose properties as well understood and whose parameters, θ_L, are relatively easy to tune using well known methods for different scenarios, and (2) we can learn over a significantly smaller parameter space ($|\theta_H| \ll |\theta|$) for a controller whose physics is less well understood.

3.3 Comparison of Learning Tasks

This section compares the parameter spaces of our learning tasks, i.e., purely data-driven vs. hybrid learning parameter spaces. The full state space is given by the state vector $\boldsymbol{s} = [x\ \dot{x}\ y\ \dot{y}\ z\ \dot{z}\ \phi\ \dot{\phi}\ \theta\ \dot{\theta}\ \psi\ \dot{\psi}]^T$, which is a 12-tuple. If we want to achieve

trajectory tracking, we need to add additional information about the target location, i.e., $[x_{target}, y_{target}, z_{target}]$. Hence the full length vector has 15 parameters, each with a continuous-valued range of possible values.

3.3.1 Model-Based

In a purely model-based FTC approach that uses PID control at the low level and a supervisory controller, we must manually tune the PID controllers and the supervisory controller. Further, if we pre-define the fault controllers for the rotor faults, we must tune these controllers as well. The drawback to using specific fault controllers is the cost of tuning each of these controllers, as well as the limitation of the approach to single-fault scenarios, since it is impossible in typical control frameworks to "merge" the outputs of these controllers [91]. To overcome this, a technique known as Blended Control [103] which uses a convex combination of controller outputs is used in this article. However, even for this latter approach one cannot adapt to changing environmental conditions or novel faults, as one can by dynamically learning controllers. PID controllers require three gain parameters to be tuned, for which established mechanisms exist [97].

3.3.2 Data-Driven

For the data-driven task, we must learn the direct motor commands applied to the rotors of the quadcopter, i.e $\boldsymbol{a}_t = [U_1\ U_2\ U_3\ U_4]$. We limit the speed of each of the four rotors to 10000 rpm, creating an action space of four actions with range [0-10000]. In the case of optimal attitude control, there is little tolerance and flexibility as to the sequence of control signals that will achieve the desired attitude [95]. For example, to achieve a stable hover all four motors much spin at exactly the same speed, which is trivial to define for model-based methods but difficult for a data-driven controller to learn due to the large continuous state and parameter space.

3.3.3 Hybrid

For the hybrid task, the supervisory controller uses a weighted combination of pre-defined PID controllers [121]. This Randomized Blended Control (RBC) architecture samples the blending weights used for the convex controller combination from a probability distribution [103]. The learning task in the hybrid approach is to *learn an optimal probability distribution* for RBC which can be defined by mean and standard deviation, $\theta_H = [\mu, \sigma]$. This reduces the size of the action space of the agent to two parameters with range [0,1]. We tune the parameters for the low-level model-based

controllers, θ_L, using standard mechanisms prior to training the high-level controller.

Learning randomized blended control is thus a significantly smaller problem (in terms of parameter space) than data-driven methods; it also has an inherently safe action space, i.e., there is no way the agent's actions can crash the quadcopter unless one of the low-level controllers performs an unsafe action and even then there is only a small random chance that this action is fully selected by the high-level controller.

4 Hybrid Quadcopter Control

In this section, we will describe the implementation of the presented approach on top of an open source Python-based Quadcopter simulation [122]. The description will be broken down into the model-based low-level control architecture, the learning-based high-level controller and how the two components integrate together through randomized blended control.

4.1 Model-based Low-Level Control

Researchers have successfully learnt the dynamics model of quadcopters through DRL but they are not comparable to traditional controllers yet [123, 95]. Our decomposition of the FTC task enables the integration of learning-based low-level control mechanisms into our approach, once they become a competitive solutions. For nominal trajectory tracking only a single roll and pitch controller is needed. Fault tolerance is provided by additional PID controllers tuned for general fault conditions such as a high-gain controller that responds more aggressively to wind disturbances or rotor faults. In this article, we focus on roll and pitch attitude control and hence add a high-gain roll and pitch PID controller for more aggressive maneuvers. For clarity we referred to this controller as C1 and for the nominal conditions (lower-gain) controller as C2.

Table F.1: PID parameters for low-level model-based control architecture.

Axis of control	$\mathcal{P}$	$\mathcal{I}$	$\mathcal{D}$
X-Position	300	0.04	450
Y-Position	300	0.04	450
Z-Position	7000	4.5	5000
Roll - C1	24000	0	12000
Pitch - C1	24000	0	12000
Roll - C2	4000	0	1500
Pitch - C2	4000	0	1500
Yaw	1500	1.2	0

4.2 Learning-based High-Level Control

We use an actor-critic neural network implemented using the stable-baselines framework [124]. This network consists of two hidden layers of 64 neurons. The observation space of the agent consists of a subset of the state variables as well as the target position, namely $[x \;\; y \;\; z \;\; \phi \;\; \theta \;\; \psi \;\; x_{target} \;\; y_{target} \;\; z_{target}]$. Quadcopter FTC requires real-time actions from the high-level controller, as any delays in action during a fault could, for example, cause a crash. This direct control approach avoids delays associated with fault isolation in traditional FDI methods.

4.3 Randomized Blended Control

RBC draws the weighting vector from a probability distribution defined by mean μ and standard deviation σ at each cycle of the control loop. In this work the high-level controller learns the parameters defining the underlying probability distribution used for RBC in real-time as faults occur. RBC has been shown to be stochastically stable [103], so flight stability is ensured even when it is learning to deal with non-terminal faults, i.e., faults for which no control is viable.

4.4 Training Details

We enforce a 1 meter *safety region* around the trajectory to evaluate control performance under fault scenarios. The task of the agent is to learn to distribute control in such a way that the quadcopter stays inside the safety region when experiencing faults. A negative reward is applied when the quadcopter drifts outside the safety region and a large positive reward is received when the quadcopter completes the entire trajectory inside the safety region. The trajectories used for training need to be diverse enough to expose the agent to a large variety of experiences. In this work, we use straight-line trajectories to a randomized destination point within a 7-meter bounding box. Proximal Policy Optimization (PPO) is selected as the training algorithm, with a learning rate of 0.1.[2] The agent was trained for a total of 10 million time steps, which is approximately 12000 episodes.

4.4.1 Fault Generation

Rotor faults are one of the most common faults experienced by quadcopters and the focus in this work. We investigate faults in any rotor and up to 30% loss of effectiveness

[2]Fine-tuning the parameters of a learning algorithm is a challenging task which will be investigated in future work.

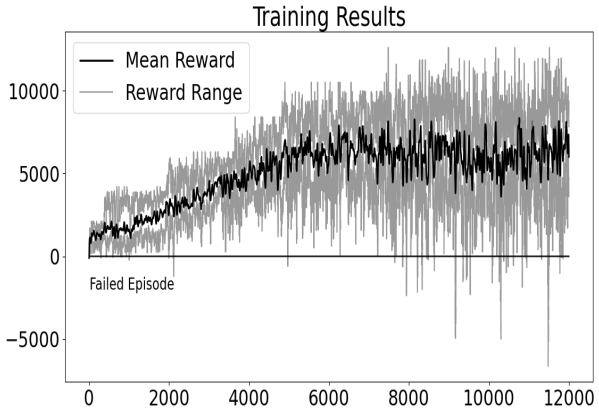


Figure F.2: Mean Cumulative Reward obtained over 5 independent training cycles of 12000 episodes. Scores below 0 indicate the quadcopter left the safe zone during the episode.

(LOE). Greater than 30% faults will cause a crash, and 20% causes severe disruption to the flight path but is marginally controllable by the higher-gain controller. Instead of exposing the agent to the entire fault space randomly, we systematically divide the fault space into *levels* representing increments of 5% in the fault magnitude. We increase the fault level when the agent does not leave the safe zone for 20 consecutive episodes. The reward obtained for completing a trajectory is proportional to the level. Hence, as the agent progresses to higher fault magnitudes the reward also increases as seen in Fig. F.2, where each data point shows the cumulative reward over the last 20 episodes. However, larger faults create more severe disturbances which is why episodes fail towards the end of the training cycle. Since the faults are of the same type but varying magnitudes, information learned on lower levels is relevant on higher levels. After 6000 episodes the mean cumulative rewards stops increasing showing that the agent converged.

4.5 Data-Driven Quadcopter Controller

To date, fully data-driven controllers are restricted to simple tasks, e.g., learning how to hover a quadcopter and simple trajectory tracking to a location [123]. Due to the lack of guarantees of stability (as exists for many model-based controllers), there exist no guarantees on the actions a learned controller will perform when presented with novel faults or unmodelled dynamics, as we do in our experiments.

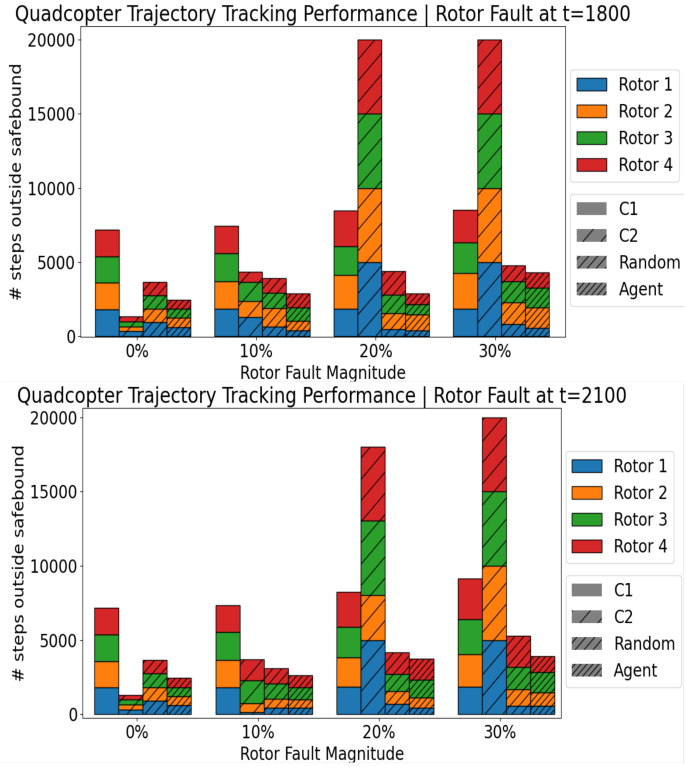


Figure F.3: Experimental results showing the amount of time a quadcopter was outside of the safety bound for rotor faults of varying magnitudes. Lower is better.

5 Experiments

This section compares the performance of the trained agent with that of (a) the high-gain and low-gain PID controllers alone as well as (b) a uniform randomized blended controller. Intuitively, we expect the agent to outperform the uniform randomized approach, since this would show that the agent has learned an improved probability distribution.

5.1 Experimental Design

To validate these hypotheses, we ran experiments on a test trajectory, a 5-meter diamond, which is significantly more difficult than the straight line training trajectories. Faults at two separate time points are investigated, $T1 = 1800$ and $T2 = 2300$. We test faults of magnitude 0%, 10%, 20% and 30% , where 0% indicates nominal control, on all rotors at each time point. As a performance metric we use the **total time outside the safety bound** so the smaller the better. Figures F.3 shows the summarized experimental results.

5.2 Parameter Spaces

Data-driven approaches typically use all of the parameters available as observation and generate the four motor commands as the action. The minimum observation vector needed is the full state vector of size 15 and most approaches use additional parameters [93]. The hybrid approach only requires a small subset of parameters that are specific to the task since flight control is already established. We use an observation space of 9 parameters and an action space of 2 parameters creating a smaller problem compared to the purely data-driven approach.

5.3 Trajectory Following Results

5.3.1 Hybrid Approach

We will now discuss the results presented in Figure F.3. There are four groups of bars for each chart representing one fault level. Each bar represents a different controller C1, C2, RBC and the trained hybrid approach, on that fault level. Each coloured stack in a bar represents which rotor experienced the fault.

The *left-most bar group* in the chart represent nominal conditions (0%). As expected, the bars in this group are identical, as no fault occurs. This also shows that the smooth controller (C2) is much better under nominal conditions as the high-gain controller naturally overshoots the way-points. This makes C1 undesirable as a controller under nominal conditions. However as the fault magnitudes increase we see that C2's performance drastically degrades. This is because the controller cannot respond aggressively enough to correct the effect of the higher rotor fault. The high-gain controller is much more robust to rotor faults as the naturally aggressive reactions can mitigate rotor faults quickly and performance is less affected.

The *third bar* in each bar group represents random blended control, which samples from a uniform probability distribution over the controllers. Uniform randomized blended control outperforms both C1 and C2 on fault magnitudes of 10% or higher. This shows how randomization is able to utilize benefits from both controllers.

The *fourth (right-most) bar* in each group represents our approach which significantly outperforms all other controllers under all fault levels. The agent is able to very quickly shift control to the aggressive controller to mitigate the fault and then continue using the nominal controller when the system stabilizes. This allows our approach to provide good control under faults that would otherwise crash the system. Under nominal conditions our approach performs comparably to the better-performing individual controller. This shows that the agent was successfully able to learn how to parameterize

the underlying probability distribution used for RBC, and thus provides an exciting new way to integrate model-based low-level control frameworks with high level data-driven controllers.

5.3.2 Data-Driven Approach

We implemented a learning-based controller where a neural network generates four motor throttle commands, which are applied directly. The full state vector and target location are set as the observation vector. Even when we used a simpler learned trajectory, a simple hover at a given altitude, and increased the training length to 50 million steps ($5\times$ as much as our approach, or 48 hours CPU-time), *the agent was unable to converge*. There are several possible reasons for this: (1) Parameter fine-tuning is needed such as the learn rate, neural network size, etc.; (2) Reward shaping - a well known open problem for complex control tasks; (3) More training time/better hardware is needed.

6 Conclusion

We compared a purely data-driven FTC approach with a neural-symbolic approach that uses an architectural integration of data-driven and model-based FTC.

The architecture uses models for low-level controllers, and learns a supervisory controller for switching weights across multiple controllers. We demonstrate our approach on learning trajectory following for a quadcopter that follows a safe region even through it experiences faults in its rotors. We empirically show that our hybrid learning approach converges to safely follow given trajectories, whereas a purely data-driven approach requires significantly more training to converge than the hybrid approach (if it converges at all). The parameter-space for the data-driven approach is significantly larger than that of the hybrid approach, resulting in this increased training challenge. Further, the hybrid approach inherits the stability guarantees of the model-based component, whereas there are no similar guarantees for the data-driven approach.

Paper G

A Novel Hybrid Approach for Fault-Tolerant Control of UAVs based on Robust Reinforcement Learning

Abstract

Although the use of unmanned aerial vehicles (UAVs) is increasing, much work remains to guarantee fault- tolerant control (FTC) properties of these vehicles. We propose a novel hybrid FTC approach that uses a learned supervisory controller (together with low-level PID controllers) with key stability guarantees. We use a robust reinforcement learning approach to learn the supervisory control parameters and prove stability. We empirically validate our framework using trajectory-following experiments (in simulation) for a quadcopter subject to rotor faults, wind disturbances, and severe position and attitude noise.

1 Introduction

Researchers have developed fault-tolerant control strategies for UAVs by using model-based, learning-based or combined techniques [125]. The performance and robustness of model-based FTC methods is highly dependent on the accuracy and comprehensiveness of the dynamic model of the system. A sophisticated model leads to a more complex control strategy, which compromises the tractability of the controller for real-time implementation. Learning-based FTC methods on the other hand allow developing control strategies by using data obtained from the system. Among learning-based FTC approaches, reinforcement learning (RL) has recently shown promising results [126, 127]. However, RL methods, e.g., [128], along with many other approaches, e.g., [129], lack control stability guarantees [125], which is of primary importance for the control of UAVs.

In this article, we propose a novel hybrid control strategy for FTC of UAVs on the task of maintaining the aircraft within a specified operation volume during flight. Our approach is based on using a hierarchical control architecture formed by a collection of model-based low-level controllers and a supervisory robust RL-based controller. We extend our prior work on FTC for quadcopters [103, 121] by applying robust RL methods, and validating our approach not just on simple linear trajectories but on maintaining flight within a volume enclosing a linear trajectory.

The proposed approach integrates key properties of model-based and learning-based methods, as follows:

A. We propose a hybrid control strategy with stability guarantees under all but unrecoverable fault conditions [103], which maintains a satisfactory performance for novel faults under the assumption that the effect of a novel fault on the task to be solved is similar to previously experienced faults.

B. We leverage robust RL approaches, and use domain randomization to train the high-level controller.

C. We propose a novel FTC approach for UAV trajectory following tasks based on optimal operational volumes during flight.

We demonstrate stability guarantees for the proposed hybrid control approach and empirically validate it in simulations of flying in safe spaces of urban scenarios with a quadcopter testbed.

2 Related Work

Commercially available UAVs present impressive trajectory tracking capabilities and typically utilize cascading control architectures that divide the task into position and attitude control [130, 131, 132]. Position controllers generate the required attitude reference and attitude controllers generate the required motor commands to follow a reference trajectory. The dynamic model of UAVs is well understood and a variety of controllers such as LQR, PID and MPC, for both position and attitude exist [132, 133, 91]. With advances in computational power, learning-based control approaches have made significant progress in UAVs control [134, 93, 94]. We next discuss the advantages and drawbacks of model-based and learning-based techniques for FTC of UAVs.

Model-based control is the most common way to control UAVs in the real-world. This approach uses physics-based models, which can be used to establish stability proofs and guarantee system safety. Using cascading LQR controllers, Mueller et al. [131] show that control of a quadcopter can be maintained even after the loss of one, two or three rotors, highlighting the power of model-based design. By adding redundant controllers designed around different operating conditions the control system is able to switch between controllers when a disturbance is identified, which is refereed to as Multiple Model Adaptive Control [89]. However, obtaining models of the system and environment for every possible operating condition a UAV can experience in a real world scenario is not feasible [103]. This is a major drawback of using pure model-based control methods as it limits fault tolerance to faults considered at design time.

Learning-based control, and specifically reinforcement learning, focuses on learning the direct control mapping of measurements to motor commands [93, 94, 127]. Reinforcement learning allows to learn policies requiring a vast amount of interactions with the system. Since real-world data is usually expensive to acquire, learning in simulation has became a popular strategy for training an RL agent. However, agents trained with a simulator tend to fail to generalize when transferred to the real environment due to a mismatch between the simulation and reality, usually called the *reality gap* or *sim-to-real* problem [135, 136, 137]. In reality, the parameters of the simulator can be different from the real-world setting because of several reasons, e.g., the modeling errors, changes in the real-world parameters over time, and adversarial disturbances. For example, varying weather conditions, sensor noise, and actuator noise commonly occur across robotics domains. Robust reinforcement learning addresses the problem of finding an optimal policy that is robust against parameter uncertainties. Robustness to changing or mis-specified environmental dynamics is an

important topic in reinforcement learning, which is crucial for overcoming the gap between simulation and reality known as Sim2Real transfer problem, i.e. a policy trained in a simulator is executed on a real-world domain. Agents tend to over-fit the domain on which they are trained because of an overestimation of the policy's performance given by a simulation optimization bias. Thus, even small perturbations of the real-world setting with respect to the training environment can have a significant negative impact on the performance of the agent.

Hybrid control methods have become popular and increasingly successful for robust quadcopter control since both learning-based and model-based control methods have clear advantages and disadvantages. A typical approach is to use all prior knowledge, in terms of models available, to design robust system controllers and then use a learning component to compensate for unmodeled details. In [129], a similar hybrid control framework is used to make a quadcopter robust to cyber-physical attacks. A low-level cascading PID architecture is used for trajectory tracking and a neural network is used to compensate for the attacks on the system. The neural network is used to compensate for the missing thrust which greatly improves the trajectory tracking capabilities under rotor fault scenarios. Other hybrid approaches learn either inner or outer control loop only and use model-based controllers for the remainder [127, 138]. While this simplifies the complexity of the task, the *direct combination* of a model-based controller output with a learning-based controller output breaks the existing stability guarantees that make model-based controllers safe. To the best of the authors knowledge, most hybrid approaches for UAV control suffer from this problem.

3 Preliminaries

This section presents the background on RL approaches developed to obtain robust policies.

3.1 Robust Markov Decision Processes

A robust Markov decision process (R-MDP) is defined using a tuple $(S, A, R, \mathcal{P}, \gamma)$, where S is the state space, A the action space, $R : S \times A \to \mathbb{R}$ represents a reward function, $\gamma \in [0, 1]$ is called discount factor, and $\mathcal{P}(s, a) \in \mathcal{M}(S)$ is an uncertainty set where $\mathcal{M}(S)$ is the set of probability measures over next states $s' \in S$. An R-MDP defines how the next state of the system is determined by a conditional measure $p(s'|s, a) \in \mathcal{P}(s, a)$ where s is the current state and a is the action selected by an agent.

The robust RL literature usually adopts the assumption that $\mathcal{P}$ is structured as a

cartesian product $\otimes_{s\in S,a\in A}\mathcal{P}_{s,a}$, which is also known as the state-action rectangularity assumption [139]. In RMDPs, this implies that nature can choose the worst-transition independently for each state and action.

In the standard MDP framework, an agent is defined by a policy $\pi : S \rightarrow p(A)$ that maps states to distributions over actions with the goal of maximizing the sum of the discounted rewards it receives over the future. The optimization criterion is the following

$$\mathcal{J}(\pi^*) = \max_{\pi\in\diamond} \mathcal{V}^{\pi}(\int) \ , \ \forall \int \in S. \tag{G.1}$$

where the value function, $V^{\pi} : S \rightarrow R$, defines the value of being in any given state s

$$V^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s,a)\right] \ , \ \forall s \in S. \tag{G.2}$$

Traditional RL algorithms require that the system dynamics and reward function do not change over time to be able to find an optimal deterministic Markovian policy satisfying I.5. This property is clearly not satisfied in the R-MDP case. Therefore, different approaches have been proposed to tackle the uncertainty around the system dynamics. We next present two possible solutions.

3.2 Optimizing for the worst-case performance

One approach to incorporate the notion of robustness in standard RL algorithms is by transforming the optimization problem of equation I.5 to account for the variability in the environment. An optimal robust value function V^* defined over the set of possible policies and possible changes in the system dynamics is defined as follows

$$\mathcal{J}_{\overline{\overline{\mathcal{R}}}}(\pi^*) = \sup_{\pi} \inf_{\mathcal{P}} V_{\pi,p}(s) \tag{G.3}$$

The policy obtained by solving this optimization problem is robust in the sense of optimizing for the worst-case expected return considering the possible evolution of the environment. Mankowitz et al. [140] follow this approach by optimizing the worst-case squared temporal difference error. Abdullah et al. [141] propose to make problem G.3 tractable by constraining the reachable states of the system according to the average Wasserstein ball around a reference dynamics.

3.3 Optimizing for the average-case performance

Robustness can be achieved by modifying the optimization problem I.5 to maximize the expected return across the distribution of environments. The optimization criterion is formulated as follows [142]

$$\mathcal{J}_{\mathcal{R}}^{\dashv}(\pi^*) = \sup_{\pi} \mathbb{E}_{p\in\mathcal{P}}\, V_{\pi,p}(s) \tag{G.4}$$

Domain randomization (DR) approaches; e.g., Tremblay et al. [143]) attempt to solve this optimization by problem by exposing the policy to be learned to a multitude of environments within the distribution of $\mathcal{P}$ in offline settings. Active Domain Randomization attempts to improve the generalization power of the agent by focusing the learning process on the parameter space which makes the dynamics of the environment difficult to control [144].

4 Hybrid Approach for FTC

The approaches to solve the R-MDP problem presented in the previous section have some limitations if applied for FTC of UAVs. Optimizing for the worst-case performance leads to conservative and even pessimistic agents with sub-optimal policies. Optimizing for the average-case performance, on the other hand, makes the task too complex for the agent, thus r resulting in low sample efficiency, which makes the training process a labor-intensive task leading sometimes to no convergence at all. In both cases, the resulting agent may perform optimally across all versions of the environment but will perform sub-optimally for each independent version. Moreover, in none of the above cases the resulting agent provides stability guarantees.

We propose a hybrid approach that provides stability guarantees: we use a hybrid control architecture as shown in Figure G.1. Through this control architecture, the agent learns to adapt to the changes in the environment by learning to sample the control actions from a set of model-based controllers. We now describe, in turn, the model-based controllers (low-level), and learning-based controller (high-level).

4.1 Model-based controllers

A cascading PID controller architecture, consisting of three position and three attitude controllers, is commonly used for trajectory tracking [132]. Fault tolerance is usually provided through additional PID controllers tuned for specific faults or disturbance conditions. For example, high-gain attitude controllers respond more aggressively to

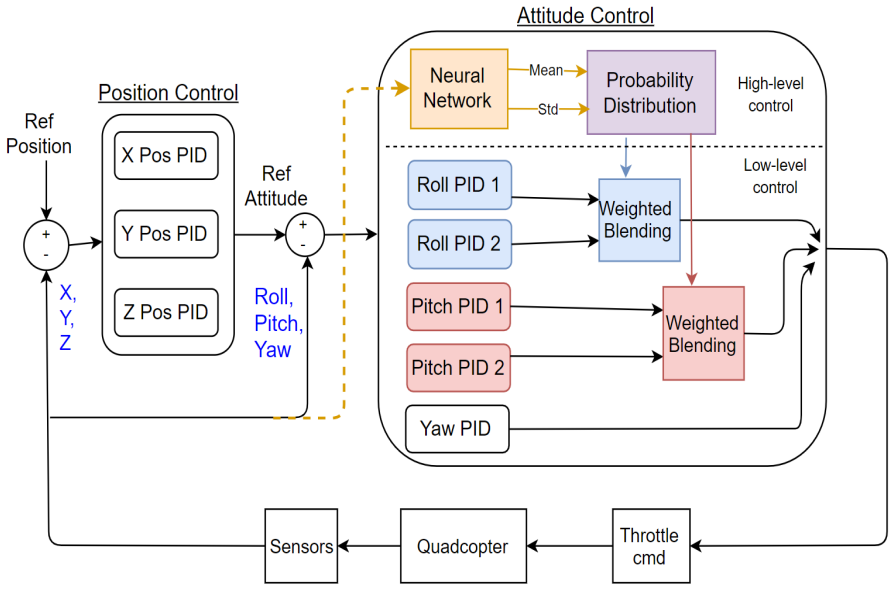
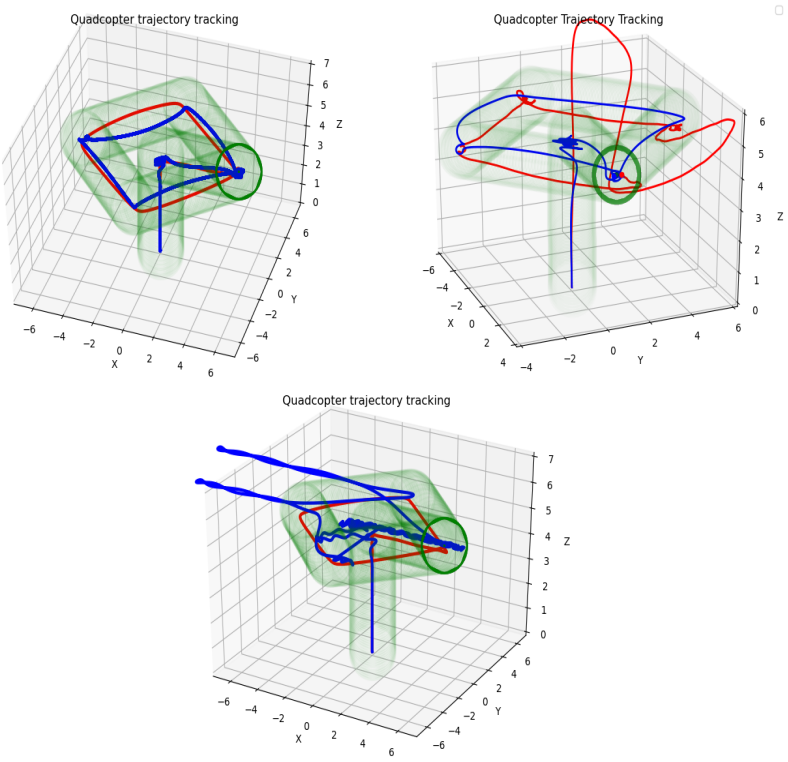


Figure G.1: Hybrid control architecture for fault tolerant quadcopter trajectory tracking. Low-level control is established using cascading PID control with additional roll and pitch controllers. A supervisory neural network controller estimates the probability distribution used to sample the blending weights.

wind disturbances or rotor faults. We focus on tuning roll and pitch controllers for different conditions. A set of controllers are tuned with **low-gains** which increases robustness to high attitude sensor noise. A different set is then tuned with **high-gains** to increase robustness to rotor faults and allow for aggressive reactions. Under nominal operating conditions both controllers must be able successfully follow a given trajectory.

We show trajectories executed by each set of controllers under the three different operating conditions in Figure G.2. The leftmost plot shows nominal operating conditions. The green area highlights the operating region the quadcopter should stay inside to avoid collision with obstacles in the environment. The middle plot shows the same trajectory under heavy attitude noise (Gaussian noise with zero mean and 0.6 rad standard deviation). The low-gain controller is able to maintain control and complete the trajectory while the high-gain controller fails due to its aggressive nature. The rightmost plot highlights the opposite for rotor faults scenarios (20 % loss of power) where the aggressive reactions of the high-gain controllers are beneficial and the low-gain controllers are unable to maintain the quadcopter within the safe region. These contrasting types of required reactions are common across different disturbances and a single controller can generally not provide both quick reaction and low overshoot [145].



.

Figure G.2: Trajectories executed under different operating conditions when using Low-gain controllers (Blue) or High-gain controllers (Red)

4.2 Learning-based controller

The proposed learning-based approach is based on the idea of sampling over the action space generated from a set of model-based controllers. The low-level controllers should be defined to accommodate the plausible variability in the distribution of environment dynamics ($\mathcal{P}$). In the case of FTC for UAVs, this variability is defined in terms of the fault scenarios to be considered as described in the previous section.

Once the set of low-level controllers is defined, the role of the supervisory agent is to map the low-level controller outputs into the final action space. We define the final action as taking a weighted sum of the low-level controller outputs. In other words, given a set of low-level controller outputs $a_1, a_2, ..., a_m$, the final control output is $\boldsymbol{a} = \sum_{i=1}^{m} \varphi_{(i)} \cdot a_i$, where each parameter $\varphi_{(i)low}$ or $\varphi_{(i)high}]$, $i = 1, ..., m$ is drawn from a $[0, 1]$ interval and $\sum_{i=1}^{m} \varphi_{(i)} = 1$. In this article, $\varphi_{(i)}$ is sampled from a Gaussian distribution defined in terms of $[\mu^i, \sigma^i]$.

The learning-based controller must learn how to tune $[\mu^i, \sigma^i]$ for each controller i depending on the environmental conditions. We use therefore domain randomization (DR) to train the learning-based controller. DR considers an extended uncertainty

set, including wind and position noise. Given a bounded set of K uncertain parameters/faults $\boldsymbol{\xi} \in \mathbb{R}^K$, DR uniformly samples from $\boldsymbol{\xi}$ at each episode such that the supervisory controller experiences a different disturbance/fault in the environment. DR enables us to generate a set of independent MDPs to cover the distribution of $\mathcal{P}$ such that the agent learns to generalize over $\mathcal{P}$. The training process is therefore similar to standard DR but the agent optimizes *across the action space of low-level controllers instead of directly optimizing across the state space of the environment*. In other words, the agent *learns an importance sampling strategy* to accommodate to the changes in the environment.

One advantage of our approach is that it allows us to learn/define the low-level and supervisory level controllers separately. This is especially useful when the low-level control problem is well understood, i.e., the controller can be defined analytically and/or is relatively easy to tune for different scenarios. On the other hand, learning how to adapt from one scenario to another is the job of the supervisory controller, which can be learned through RL algorithms such as Proximal Policy Optimization [32]. Moreover, if we define a set of low-level controllers through classic control methods with stability guarantees, the supervisory controller preserves the stability properties of the individual policies even if it is learned through RL [103]. We demonstrate this in the following section.

4.3 Stability Properties

This section describes the stability properties of our approach, and the range of environments to which the approach is applicable.

This weighted multiple-model control framework has been well studied, e.g., [146, 103]. We define a control system within this framework using $\Psi = \langle C, \Phi, E \rangle$, where we assume that: (1) we have a collection $\mathcal{C} = \{c_1, \cdots, c_m\}$ of controllers, with a distribution (or weight function) over $\mathcal{C}$, denoted $\boldsymbol{\varphi} = \{\varphi_1, \cdots, \varphi_m\}$, such that $\sum_j \varphi_j = 1$; (2) each controller c_i, with a parameter set θ_i, is tuned to optimize the performance of a corresponding model $\phi_i, i = 1, \cdots, m$ for environment $E_i \in E$, such that ϕ_i is in model set Φ and optimizes the performance in E_i.

Given this framework, a weighted control output is guaranteed to be stabilizing as described below.

Lemma 4.1. *: A WMMC system Ψ is stable if the following hold (see [147]):*

- *$\phi_j \in \Phi$, for some $j \in 1, \cdots, m$, i.e., the true plant model is included in the fixed models of the model set Φ.*

- *each model $\phi_i \in \Phi$ can generate a measured signal that is within a fixed bound $B \in \mathbb{R}^n$ of the true signal.*
- *every controller c_i is locally stabilizing with respect to its model ϕ_i and parameter set θ_i.*

Because each weighted control signal is a convex combination of controllers in the control set C, the set of all control combinations forms a convex hull C bounded by C. We can thus define the set of environments for which stable controllers exist under Ψ as follows:

Lemma 4.2. *: Given a WMMC system Ψ defined over controller/model set C/Φ, with corresponding control parameters $\Theta = \{\theta_1, \cdots, \theta_m\}$, any environment E with parameters P_E such that the optimal controller has parameters falling within the convex hull of Θ is stabilizable using Ψ.*

This second Lemma thus defines the range of environments to which our approach is applicable. We will empirically demonstrate this using a quadcopter with a set of 2 base controllers, for which we show a range of unknown fault environments are stabilizable.

5 Quadcopter Experiments

The quadcopter trajectory tracking implementation is based on an open-source python simulator [148] and integrated with stable baselines [124] to create a custom quadcopter training environment.[1] We discuss the operating conditions investigated in the experiments, the training set up for the high-level controller and experimental evaluation.

5.1 Operating Conditions

We investigate four common disturbances to the quadcopter including rotor loss of effectiveness, wind gusts, position sensor noise and attitude sensor noise. We denote a multiplicative rotor fault model with parameter $0 \leq \varsigma_i \leq 1$ for $i = 1, ..., 4$, where $\varsigma_i = 0$ corresponds to nominal function and $\varsigma_i = 1$ to total failure. Wind gusts are implemented using the Dryden turbulence model [28]. Wind direction and the rotor to experience the fault are selected randomly. We model noise using a zero-mean Gaussian distribution of varying magnitudes. Position noise is applied to x, y and z in meters and attitude

[1] A repository containing all simulation files needed to replicate the experiments is made available under "https://github.com/YvesSohege/ICRA21-QuadcopterExperiments".

Table G.1: Quadcopter domain parameters and corresponding disturbance magnitudes for each level.

Domain	**Level 1**	**Level 2**	**Level 3**	**Level 4**
Attitude Noise	0.2 rad	0.4 rad	0.6 rad	0.8 rad
Position Noise	1 m	2 m	3 m	4 m
Wind Gust	3 m/s	6 m/s	9 m/s	12 m/s
Rotor LOE	5%	10%	15%	20%

noise to ψ and θ in radians. We partition the domain parameter space into *levels* of increasing disturbance magnitudes. The full parameter ranges and for the domains and levels can be found in Table G.1.

Table G.2: PID parameters for low-level model-based control architecture.

Axis of control	$\mathcal{P}$	$\mathcal{I}$	$\mathcal{D}$
X-Position	300	0.04	450
Y-Position	300	0.04	450
Z-Position	7000	4.5	5000
Roll - C1	24000	0	12000
Pitch - C1	24000	0	12000
Roll - C2	4000	0	1500
Pitch - C2	4000	0	1500
Yaw	1500	1.2	0

5.2 Trajectory tracking task and reward definition

When a fault or disturbance occurs the quadcopter starts to deviate from its path and can no longer track a reference trajectory. However, in urban scenarios, this could lead to collision with obstacles. We therefore consider the idea of operational volumes that create a 1-meter **safe zone** around the trajectory which represents the acceptable level of deviation to avoid collisions. The goal of the agent then is to learn how to stay inside the safe zone. The agent receives a large positive reward if the goal location is reached successfully and a small negative reward (-1) for every step outside the safe zone. For an episode with δ steps outside of the safe zone, the reward is defined a $\delta * 2$ if the goal reached and $-\delta$ in case of failure. This focuses the learning on episodes where quadcopter deviates outside of the safe zone but is able to re-stabilize to reach the goal. When no deviation occurs the agent is performing well and no rewards are given, which is shown in Figure G.3.



Figure G.3: Shows reference trajectory in black with safe zone in dashed green. When the quadcopter leaves the safe zone the agent is rewarded with a -1 while positive rewards are only possible by reaching the goal.

5.3 Other Training Setup Details

We define the observation vector as $[x, y, z, \phi, \theta, \psi, X_D, Y_D, Z_D]$, where X_D, Y_D, Z_D represents the goal position. The action space is defined as $[\mu, \sigma]$. We use proximal policy optimization schulman2017proximal to train a neural network consisting of four layers of 64 neurons. Straight line trajectories are generated from an initial position on the ground to a random goal location within a 10-meter bounding box. We set a limit of 5000 steps to complete the trajectory. The domain is selected using uniform domain randomization. We train the proposed hybrid architecture for 10 million steps (5770 episodes).

5.4 Results & Discussion

The performance metric we will use to measure robustness to disturbances is the average number of steps outside of the safe zone. Firstly, we investigate the performance of the high-gain and low-gain PID controllers over entire fault space which can be seen in Figure G.5 **Top-Left** and **Top-Right**, respectively. As discussed previously, the high-gain PID is more robust to rotor faults and the low-gain performs better under attitude noise which is supported by results shown in the heatmaps. Both controllers perform poorly under heavy wind and position noise as they are not tuned for these conditions. Secondly, we are interested in how a uniform randomized blended controller performs across the fault space, seen in Figure G.5 **Bottom-Left**. This acts as a baseline to judge how much the supervisory learning component improved the performance by adapting the randomized weighting. We can see that uniform randomized blended control is able to improve the performance under small

disturbance magnitudes but is by nature sub-optimal and hence failing on any of the severe disturbances.

In contrast, our trained hybrid architecture, Figure G.5 **Bottom-Right**, is able to learn how to adjust the distribution between low-gain and high-gain controller in real time to improve performance over the entire fault space. A significant improvement can be seen under wind and position noise, for which neither PID controller performs well but the agent is able to learn how to utilize them for this new environment. The only disturbance the agent fails on is severe attitude noise which we attribute to the agent optimizing over all domains. The episode count in each cell shows how many times the agent was exposed to this environment during training.

6 Conclusion

We have described an approach that guarantees key control properties (e.g., stabilizability), in contrast to similar quadcopter control systems [129]. Our hybrid framework enables us to provide stability guarantees based on the model-based controllers, while learning a supervisory parameter space that is simpler than the parameter space encompassing the entire control task. This approach thus has the benefit of control guarantees and relatively fast parameter learning.

We have validated the FTC approach on urban trajectory-following tasks using a quadcopter. For even relatively extreme rotor faults we show that we can learn to maintain safe trajectories.

Our approach is limited to environments whose corresponding control parameters are contained in the convex hull of the baseline controller parameters. We are currently investigating how we can use RL to learn new baseline controllers for environments not covered by our initial controller set, and initial results indicate that this set of new controllers will grow slowly (and not exponentially) with respect to the complexity of the novel environment sets.



Figure G.4: Performance heatmap showing the average number of steps outside the safe zone under various disturbances. **Top - High-gain PID** is robust to rotor faults but suffers on the other domains. **Bottom - Low-gain PID** is robust to attitude noise but suffers on the other domains.



Figure G.5: Performance heatmap showing the average number of steps outside the safe zone under various disturbances. **Top - Uniform Randomized Blended Control** is unable to handle strong disturbances and **Bottom - Our Hybrid Approach** significantly outperforms the other approaches over all domains except strong attitude noise.

Paper H

Comparison of Control and Cooperation Frameworks for Blended Autonomy

Abstract

Autonomous vehicles, e.g., cars, aircraft or ships, will need to accept some degree of human control for the coming years. Consequently, a method of controlling autonomous systems (ASs) that integrates control inputs from humans and machines is critical. We describe a framework for blended autonomy, in which humans and ASs interact with varying degrees of control to safely achieve a task. We analyze multi-agent tasks in which the human and AS have identical or conflicting objectives, and in which noise is present in collaborative control. We empirically compare an algorithm based on leader-follower control with algorithms based on blended and switching control, given communication delays, noise and different collaboration levels. We validate our results on a car steering control model.

1 Introduction

Fielded autonomous systems (ASs) need to interact with humans in a variety of ways, and hence need a framework to enable that. Here, we focus on ASs that can incorporate a variable degree of human control, which we call *blended autonomy*. For example, forthcoming autonomous cars will allow drivers to set a variety of autonomy levels; for example, lower levels include cruise control and/or collision avoidance, and full autonomy incorporates all driving functionality. Analogously, UAVs flying in commercial airspace will need to follow Air Traffic Control (ATC) instructions, entailing a mechanism to integrate human voice commands into autonomously-generated flight planning.

Our objective is to develop a control-theoretic framework for blended autonomy that will allow humans to interact with a collection of ASs. This is an extension of prior work on tele-operation [149], which assumes continuous human control of a vehicle. This work also extends frameworks for automotive autonomy, e.g., [150], in that we generalize the single-driver and -vehicle case to a multi-agent case.

We propose a framework that focuses on the following key properties. First, we describe a system that can flexibly operate in multiple autonomy levels as a multi-agent system with human and AS agents that must interact to accomplish a task. Second, the human/AS interaction may have varying degrees of cooperation and information sharing among the agents; incomplete information sharing may arise due to lack of transmission of state information or communications losses. We solve our blended autonomy problem using optimal control techniques, and demonstrate the resulting properties of algorithms like Model-Predictive control (MPC) when applied to this task.

Our contributions are as follows:

- We formulate a control-theoretic framework for ASs that incorporates a varying degree of human control inputs.
- We specify the AS framework in terms of a multi-agent hybrid system, and compare three control algorithms: (1) leader-follower (e.g., as done with Stackelberg game approaches [151]), (2) blended-control, based on weighted-sum optimization [152], and (3) switching-control, with control assigned to the agent whose output is closest to the optimal output.
- We illustrate our approach using MPC methods for a car steering system jointly controlled by human and AS inputs.

2 Related Work

Teleoperation is an area in which a human operates a vehicle at a distance. Most approaches to teleoperation assume full human control (e.g., [149]). However, some recent work has included blended human/robot control, e.g., [153].

A large body of work has been published on automotive blended human/vehicle control, e.g., [154, 155, 150]. The majority of this work assumes that human and AS controls are cooperative, and that no unsafe controls are possible. Recent work that incorporates safety includes [155]. In contrast to this work on the automotive domain, we develop a framework that is generic and can be extended to more than just 2 actors (driver and AS).

Control theory has been used for developing algorithms for blended autonomy. The most common approaches are MPC [156] and optimal control [157].

Our approach explicitly adopts a multi-objective optimization, for which a large body of work exists for general problems [152] and for control applications [158].

Game theory has been used for action generation in blended autonomy. For example, [159] describes experiments with different cooperation/non-cooperation strategies for human/AS. [151] proposes an algorithm based on a Stackelberg equilibrium for cooperative steering control.

Work also exists in attempting to model the driver's intent. For example, [160] proposes the use of motion primitives for cooperative blended-control driver assistance systems. [159] proposes several human models for robot interactions. [161] examines modes for adaptation in human-AS collaboration.

3 Technical Description

This section describes our technical formulation. We present a framework that can accommodate arbitrary numbers of human and AS agents, but for simplicity we restrict our discussion to a single instance of human and AS agent. We denote human with H and AS with Υ. The actions at time t are the joint human and AS actions, denoted $u(t) = (u_H(t), u_\Upsilon(t))$, and the total space of actions is $\mathcal{U}$. We assume that we have a set θ of parameters in our system model Ψ. We can define a high-level version of our task as an optimal control problem:

Definition 20 (Blended-autonomy task)**.** *A system S solves a blended-autonomy task with objective function $\mathcal{J}$ using human and AS agents who collectively interact to*

optimize $\mathcal{J}$ within world state X_w, where human and AS agents may take on differing levels of autonomy, by selecting actions that maximize $\mathcal{J}$:

$$u^* =_{u \in \mathcal{U}} \mathcal{J}(u, \theta). \tag{H.1}$$

We will make this definition more precise in the following sections.

3.1 Multi-Agent Framework

Definition 21 (Agent ω)**.** *We define an agent $\omega \in \Omega$ using the tuple $\langle c, \Phi, \mathcal{U}, r, \mathcal{J} \rangle$, where*

- *c is the agent type, $c \in \{H, \Upsilon\}$.*
- *Φ is the agent model, denoting the agent's hybrid dynamics.*
- *$\mathcal{U}$ is the system action space, such that $\mathcal{U} = \mathcal{U}_H \cup \mathcal{U}_\Upsilon$.*
- *r is the agent's reward function.*
- *$\mathcal{J}$ is the agent's objective function.*

We assume that we can define three state types in our system, $x_w \in X_w, x_\Upsilon \in X_\Upsilon$, and $x_H \in X_H$, denoting states for the world, AS and human, respectively.

We define a blended-autonomy system (BAS) as a multi-agent hybrid system as follows:

Definition 22 (BAS S)**.** *A BAS is a multi-agent system consisting of a collection of agents $\omega_i \in \Omega, \quad i = 1, ..., n$, who attempt to achieve a collective task with system reward function R_w and objective function $\mathcal{J}_w = \varsigma(\mathcal{J}_1, ..., \mathcal{J}_n)$, where ς is a reward aggregation function for the agents. We characterize the agent interaction protocol using χ.*

3.2 Autonomous Modes

In this article we generalize the notion of mode from an indicator for operational state to also include the autonomy levels of agents. Our notion of mode corresponds to the notion of autonomy level in many application domains. For example, in the automotive domain the SAE has defined six levels of autonomy for self-driving vehicles, with level 0 denoting full human control to level 5 denoting full AS control. The levels 1 through 4 consider a blended approach where the mode defines the driving mode (e.g., highway cruising) with some autonomous capabilities (e.g., steering, cruise control).

We assume that an identical operational mode can be achieved through human or AS control. We formalize these notions in the following sections.

3.2.1 Mode Specification

We assume that a system S can operate in a discrete set of system modes $\Gamma = \{\gamma_1, \cdots, \gamma_k\}$. A mode is characterized by both the operational state and the health state of S; here we focus only on the operational state. More specifically, we define an agent's operational mode in terms of its autonomy level and state. For example, an air vehicle's operational mode can be in *cruise*, and it's autonomy mode either autopilot or human control.

We formalize modes as follows. A system S can be in one of several operational modes at any time, denoted $\Gamma_O = \{\gamma_O^1, \cdots, \gamma_O^k\}$. Given an operational mode, the system can be in one of several autonomy modes, i.e., where S is controlled by both the AS or human to varying degrees.

Definition 23 (Mode γ_k)**.** *A system S can be in one of several modes, $\gamma \in \Gamma$, which is given by the pair of operational and autonomy modes: $\Gamma \subset \Gamma_O \times \Gamma_A$.*

We use the notion of autonomy map to capture the different human/AS mode specifications that lead to task completion.

Definition 24 (Autonomy Map)**.** *We can compute the system's autonomy level using autonomy map $\varrho : \Gamma_O \times \Gamma_A \to \Gamma$.*

We can use this map to identify, for example, the AS autonomy level necessary to achieve a task given as input the human autonomy level and required mode.

We assume that Γ can take on a discrete set of values we can partition these values into safe and unsafe modes: $\varphi : \Gamma \to \{0, 1\}$, where 0 denotes unsafe and 1 denotes safe. For example, for a car with human and AS inputs, the overall autonomy level must be equivalent to full control due to inputs from human and AS for the vehicle to be safe.

3.3 Dynamical System Model

We assume that the system may operate using three different models: a world model Ψ_w, and models maintained by the AS and human, Ψ_Υ and Ψ_H, respectively.

Agent i possesses a unique dynamical model, which for a single mode $\gamma \in \Gamma$ we denote

by a discrete-time equation

$$\begin{aligned} \xi_i(k+1) &= A_i\xi_i(k) + B_iu_i(k) \\ z_i(k) &= C_i\xi_i(k). \end{aligned} \tag{H.2}$$

The system evolves according to a state transition function $T : X_w \times \mathcal{U}_\Upsilon \times \mathcal{U}_H \to X_w$. The agents can also switch system modes in a discrete manner.

Since this system is hybrid, mode switches can occur due to discrete switch commands or to continuous state evolution.

4 Multi-Agent Coordination

4.1 Assumptions

This section describes our assumptions. We assume that we have a collection of n agents, each characterized by tuple $\langle c, \Phi, \mathcal{U}, r, \mathcal{J} \rangle$. We denote the (possibly unique) i^{th} reward and objective function as $r_i,\ \mathcal{J}_i$, and Φ_i, respectively, for $i = 1, ..., n$. Φ_i is given by equations H.2, for a given mode. This framework enables us to analyze the impact of agents who may have different objectives, or agents who may have different sensors, and hence obtain different sensor outputs $y(k)$ given world state $x(k)$.

4.1.1 Reward Model

Each agent has a (potentially different) reward function, denoted as $r_\Upsilon(x, u_\Upsilon, u_H; \theta_\Upsilon)$ for the AS and $r_H(x, u_\Upsilon, u_H; \theta_H)$ for the human, each with parameters θ. The AS and human may not know each other's reward functions (or equivalently, each other's parameters θ). The human-AS team receives a real-valued reward based on the true state of the world and the combined control $u_\Upsilon \oplus u_H$.

4.1.2 Objectives

We assume that agent i has objective function $\mathcal{J}_i$, and that there exists a system-level objective function $\mathcal{J}_w$. The system-level task is the optimize $\mathcal{J}_W$ subject to $\Psi_w,\ u_H,\ u_\Upsilon$, collaboration χ, plus safety and other constraints.

We also assume that the human agent computes its control inputs u_H first, and the supervisor then needs to compute a system-level control in response to u_H. We explore situations in which the human model Φ_H is either incomplete or receives incomplete sensor inputs, so that the human actions u_H are sub-optimal. This contrasts with other

work that either assumes a perfect human model Φ_H (e.g., [155]) or learns/estimates such a model (e.g., [161]).

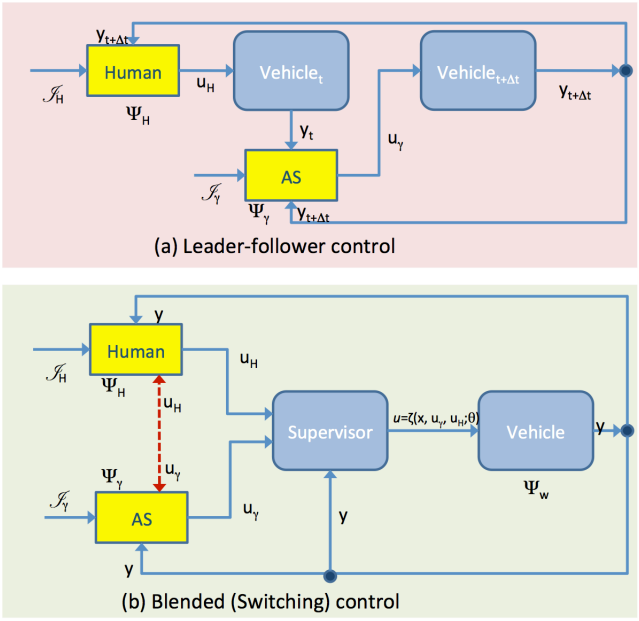


Figure H.1: Architecture for blended autonomy system: (a) shows the leader-follower approach, and (b) shows the blended approach

Figure H.1 shows the two control approaches (applied at each step t) that we compare.

4.2 Leader-Follower Control Approach

This section describes how we can use game theory to coordinate the (possibly conflicting) objectives of the agents. Figure H.1(a) shows the leader-follower approach, where the human agent applies a control $u_H(t)$ to the vehicle, and then the AS agent observes $y(t)$ and then applies $u_\Upsilon(t + \delta t)$ to the vehicle, for some small δt.

In the general case, we assume that the human and AS generate controls through solving two different optimization problems. This may arise in several situations. For example, a human air traffic controller (ATC) may specify a landing plan for a UAV that optimizes over all planes within the ATC's sector, where an individual UAV has computed a plan that optimizes its individual fuel consumption. In a different case of a car, the human driver may may be tired and issue dangerous commands while the AS may compute safe commands for the car.

We adopt the leader-follower approach from game theory the can be used to compute a Stackelberg equilibrium solution, e.g., as in [151]. In this approach, the human first

selects a control, and the AS then selects a control, and the process repeats. Given observability of control settings for both agents, we can compute a strategy profile that optimizes the agents' objective functions, given the strategies of the other agent. Given the computed strategies, we implement them using a control switching framework, in which our controller switches from u_H to u_Υ at each step.

This approach does not attempt to incorporate stochastic adaptation to or learning of the other agent's objective function or dynamical model. It contrasts with the Bounded-Memory Adaptation Model of [162], which uses a parameter α to capture an agent's inclination to adapt. For example, if human and AS disagree, the human may switch from their control u_H to the AS's control u_Υ at the next time step with probability α.

We assume that both human and AS agents minimize their respective objective function. Then, a solution of the differential game requires solving a multi-objective optimization problem. We adopt a receding horizon optimization approach, i.e., the optimal controls are calculated by solving the open-loop optimal control problems for the horizon

4.3 Blended Control Approach

Figure H.1(b) depicts the architecture for a blended autonomy system. In contrast to the leader-follower approach, the AS and human each compute controls (u_Υ and u_H) that are "integrated" in the supervisory control module in an optimal manner.

Solving a multi-objective function $\boldsymbol{\mathcal{J}}_w = \varsigma(\mathcal{J}_1, ..., \mathcal{J}_n)$ is computationally intractable, so we adopt the widely-used weighted-sum method for optimization [152]. We need to represent vectors for the collection of n agents, so we denote $\boldsymbol{\Phi} = [\Phi_1, \cdots, \Phi_n]^T$, and $\boldsymbol{\mathcal{J}}(u, \alpha) = [\mathcal{J}_1(u_1, \alpha_1), \cdots, \mathcal{J}_n(u_n, \alpha_n)]^T$. We map the multiple objective functions into a single weighted function, i.e., assigning a non-negative weight λ_i to each of the i objective functions, given by $\mathcal{J} = \boldsymbol{\lambda}^T \boldsymbol{\mathcal{J}}(u, \Phi)$, where $\boldsymbol{\lambda}^T = [\lambda_1, \cdots, \lambda_n]^T$ is the weight vector. The objective is to optimize $\mathcal{J}$ by selecting weights such that $\sum_i \lambda_i = 1$ and $1 \geq \lambda_i \geq 0, \;\; i = 1, \cdots, n$. The optimal solution is given by

$$u^* =_u \boldsymbol{\lambda}^T \boldsymbol{\mathcal{J}}(u, \Phi)$$

In general, we need to normalize the objective functions since not all objectives have the same range of values; after normalization we use standard optimization algorithms to compute optimal solutions.

[159] defines, in a qualitative manner, four key types of collaboration for interaction

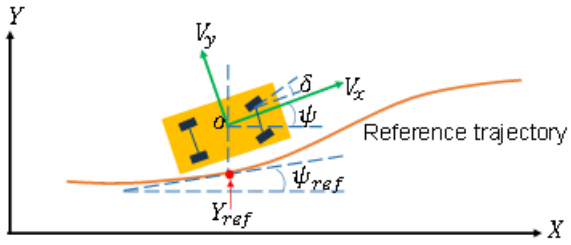


Figure H.2: Bicycle Model of a car

between human and AS: collaboration that is (1) perfect; (2) collaborative but approximately optimal; (3) subject to incomplete or corrupted communication; (4) non-collaborative.

The key question is whether every agent knows the autonomy mode and reward model for the other agents. These data can be communicated among agents, but if we have disturbances in collaboration (e.g., incomplete communication) then complete information will not be globally held. In this article we empirically compare the impact of agents having different models, objective functions, and incomplete/corrupted communications.

5 Bicycle Model

For our experiments we used the bicycle model whose lateral vehicle dynamics have two degrees of freedom, lateral position and yaw angle. The vehicle model is depicted in Figure H.2. We define our notation in Table H.1.

The lateral velocity of the vehicle v_x is constant and hence the *control input* corresponds to the front wheel steering angle δ of the vehicle, under the assumption that only the front wheels can be steered.

Definition 25 (Kinematic Bicycle Model)**.** *We define the kinematic bicycle model as [163]:*

$$\begin{aligned}
\dot{x} &= v cos(\psi + \beta(\delta)) \\
\dot{y} &= v sin(\psi + \beta(\delta)) \\
\dot{\psi} &= \frac{v}{l_r} sin(\beta(\delta)) \\
\beta(\delta) &= tan^{-1}\left[tan(\delta)\frac{l_r}{l_f + l_r}\right]
\end{aligned}$$

Table H.1: Notation used in the article

x, y	Position of the center of gravity (CoG) of the vehicle in the ground framework in (x, y)-plane
ψ, δ	Yaw and steering angle of the car body
v_x, v_y	Longitudinal and lateral speed of the vehicle in its inertial frame
M	Total mass of the vehicle
l_f, l_r	Distance from front and rear axle to CoG
$\beta(\delta)$	Slip angle at the CoG
C_f, C_r	Front and rear cornering stiffness
I_z	Polar moment of intertia
i_s	Steering Ratio

Since this vehicle model has more degrees of freedom than control inputs its classified as under-actuated.

Definition 26 (Vehicle Model Φ)**.** *The dynamics of the longitudinal, lateral and yaw motions of the whole vehicle are given in the form of state space equation H.2, where* $\varphi(k) = [v(k)\ \ \omega(k)\ \ y(k)\ \ \varphi(k)]^T$*, control* $u(k) = [y(k)\ \ \varphi(k)]^T$*, and* $A,\ B$ *and* C *are given as follows:*

$$
\begin{aligned}
A &= \begin{bmatrix} \frac{-(C_f+C_r)}{Mv_x} & \frac{-(l_fC_f-l_rC_r)}{Mv_x} - v_x & 0 & 0 \\ \frac{-(l_fC_f-l_rC_r)}{I_zv_x} & \frac{-(l_f^2C_f+l_r^2C_r)}{I_zv_x} & 0 & 0 \\ 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & 0 \end{bmatrix} \\
B &= \begin{bmatrix} \frac{C_f}{i_sM} & \frac{l_fC_f}{i_sI_z} & 0 & 0 \end{bmatrix}^T, \quad C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.
\end{aligned}
$$

We are interested in controlling the (x, y)- position and velocity v of the vehicle. The position (x, y) of the vehicle in the ground framework is given by equation H.3:

$$
\begin{aligned}
\dot{x} &= v_x cos\psi - v_y sin\psi \\
\dot{y} &= v_x sin\psi + v_y cos\psi
\end{aligned} \tag{H.3}
$$

Definition 27 (Control Problem)**.** *Given a list of waypoints* $(w_i)_{i\in I} = (x_i, y_i, v_i)_{i\in I}$*, where* (x_i, y_i) *are the successive reference positions of the vehicle in the ground frame and* v_i *the successive speed references at position* (x_i, y_i)*, the control problem consists in finding a sequence of feasible control inputs* δ *that stabilizes the system and tracks the reference trajectory given by the waypoints.*

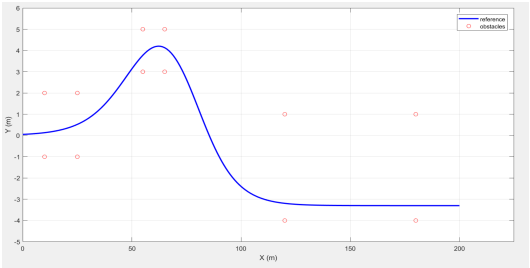


Figure H.3: Double Lane change reference trajectory

Figure H.3 shows a double lane change manoeuvre which will be used as the reference trajectory for the experiments.

6 Empirical Experiments

For our experiments we use the bicycle steering model described previously. We implemented the Leader-Follower, Oracle-Based Switching and Blended control framework for this vehicle on top of a Matlab Autonomous Steering demo. To simulate the interaction of control for the described framework in previous sections between an AS and a Human we define two Model Predictive Controllers (MPC) for u_H and u_Υ, respectively. with weight assignment for the measured variables ψ and Y being set to [1,1] for u_H and [1,0.1] for u_Υ. Intuitively the Human controller focuses more on reference tracking between ψ_{ref} and ψ instead of mostly on the y position tracking, which should give smoother tracking overall. We minimize the total tracking error $\epsilon = \sum_{i \in \mathcal{W}} |y_{ref}(k) - y(k)|$ over displacement waypoints $\mathcal{W}$. We empirically study the impact of the following across the control approaches:

- Bad u_H signals, such as a collision course or no input at all;
- Signal delays;
- Sporadic noise on the dominant control signals.

6.1 Leader-Follower

The Leader-follower (LF) framework alternates the control signals applied to the car model at intervals of $\delta = 1s$. Figure H.4 shows the LF framework executing the double lane change manoeuvre in nominal operating conditions. The figure is broken into 3 sub-charts each showing different metrics during the simulation. The top consists of the (x, y) position of the reference path (blue), actual path taken (green) and obstacles to avoid (red circles). In the middle chart the yaw angle of the car can be seen, again

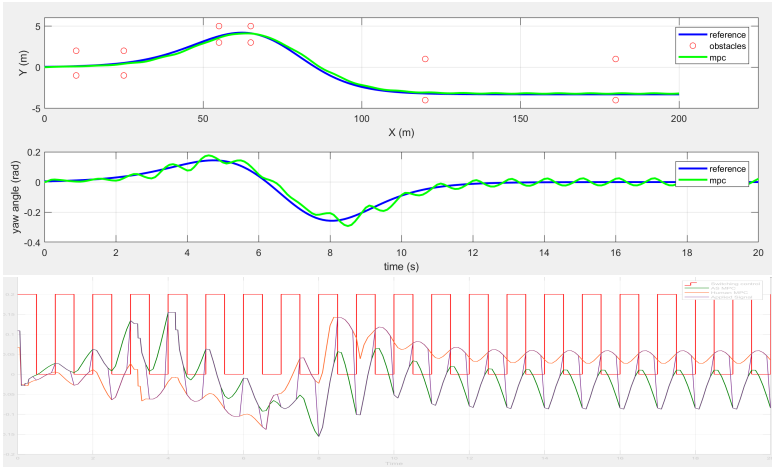


Figure H.4: Leader-Follower framework in nominal operating conditions

reference position (blue) and actual position (green). The last chart consists of the MPC output signal u_H (orange) and u_Υ (green), the applied control signal u (purple) and η (red) which signifies the current control proportions between Human controller and AS used to calculate u. The X-axis represents time and the Y-axis the magnitude of the signal.

6.2 Blending

Blending combines the u_H and u_Υ signals using weight parameter η such that: $u(k) = u_H(k)(1-\eta) + u_\Upsilon(k)(\eta)$, with $0 \leq \eta \leq 1$. We predict the tracking error ϵ between y and y_{ref} for time the current time point k. Since by nature the current tracking error ϵ will gradually increase, we can use this such that $\eta = \epsilon$ so that if control starts to diverge, u_Υ will gradually become more influential and should intuitively re-stabilize control. Since u_H plays a more dominant part in the control signal u we call this **Human Dominant Blending**. Alternatively we also experiment with **AS Dominant Blending** which is defined by $u(k) = u_H(k)(\eta) + u_\Upsilon(k)(1-\eta)$.

6.3 Oracle-based Switching

Oracle-based switching (OBS) is a framework that switches control to the agent whose output is closest to that of an oracle Θ. Θ has knowledge of the reference path $\Theta_{y_{ref}}$, obstacles Θ_{obs} as well as the car's current trajectory Θ_σ, which is defined by fitting a polynomial function of degree 4 to the last 5 (x, y) coordinates. Evaluating $\Theta_\sigma(t+\tau)$ allows us to predict the path the vehicle will take during the next τ seconds.

We define our control $u(k)$ such that the human control is applied ($u = u_H$) if any of the predicted y coordinates are more than β meters from the reference trajectory or the

vehicle is on a collision course with any of the known obstacles; otherwise $u = u_\Upsilon$. Formally, $u(k)$ is generated such that:

$$u(k) = \left\{ \begin{array}{ll} u_H, & \text{if } div \geq \beta \\ u_H, & \text{else if } col \leq \beta \\ u_\Upsilon, & \text{Otherwise} \end{array} \right\}$$

$$div = |\Theta_\sigma(t+\alpha) - Y_{ref}(t+\alpha)| \tag{H.4}$$

$$col = |\Theta_{obs_i}(y) - \Theta_\sigma(t+\alpha)(y)| \quad \forall \Theta_{obs_i} \in \Theta_{obs} \tag{H.5}$$

Here div represents the expected future divergence to Y_{ref} and col the distance to an obstacle for all known obstacles. $\beta = 0.2$ and $\alpha = [0.1, 0.2, 0.3]$;

6.4 Experiment 1: No Human input

We set $u_H = 0$ to model a Human that has fallen asleep or is otherwise unable to issue control commands. Figure H.5 (left) shows the difference in total tracking error between the frameworks for the described double lane change manoeuvre. Surprisingly the OBS and Human Dominant Blending perform very poorly. We assume this is due to the system being allowed to get into a bad state, from which it tries to recover; in contrast, the Leader-Follower and AS-Dominant Blending approaches always assign a degree of control to the AS, such that even if $u_H = 0$ the AS-Dominant Blending and Leader-Follower are only affected slightly. However, all frameworks were able to re-stabilize control and execute the manoeuvre without crashing into any obstacles.

6.5 Experiment 2: Human on Collision Course

We next investigated the effect of bad u_H. The Human MPC controller was modified to have a slower sampling rate, which gives it worse tracking and causes a collision during the manoeuvre. This experiment is designed to test how the frameworks respond to bad human input. Again, Figure H.5 (middle) shows ϵ for the four frameworks. Again all frameworks were able to avoid collision and complete the manoeuvre. The LF framework experienced a thrashing yaw angle signal, but the y position tracking is good. Human Dominant Blending and OBS trashed less but overall tracking was worse. AS Dominant Blending performed very smoothly for both signals and achieved best control. This is because the Human agent has little or no control of the system,

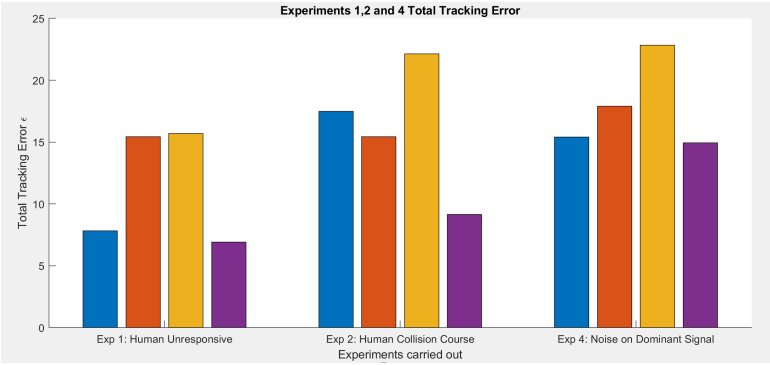


Figure H.5: Total tracking error for double lane change for Experiments 1,2 and 4. Blue: Leader Follower, Orange: Oracle-based Switching , Yellow: Human Dominant Blending, Purple: AS Dominant Blending

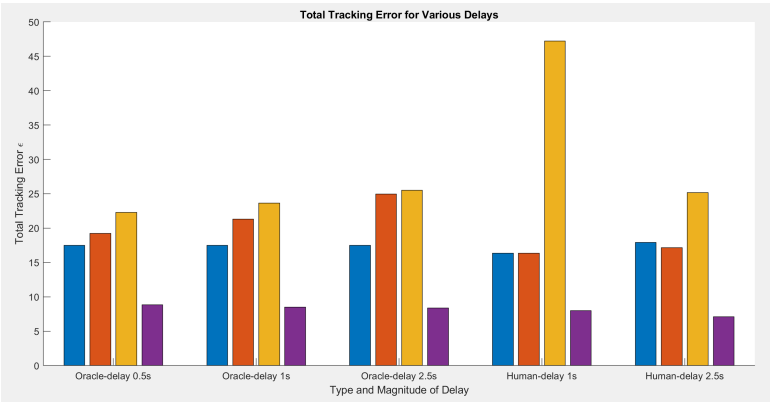


Figure H.6: Total tracking error for double lane change with given delays.

so being on a collision course does not affect the overall system. Given results from Experiments 1 and 2, we can conclude that AS Dominant blending performs the best if we cannot guarantee a good u_H input.

6.6 Experiment 3: Delays on various signals

We studied the effect of delays in the following signals:

A. Oracle computation of Θ for $\delta = [0.5, 1, 2]$.

B. AS control signal u_Υ for $\delta = [1, 2]$.

C. Human control signal u_H for $\delta = [1, 2]$.

D. Overall applied system u for $\delta = [0.5]$.

Here, δ represents the delay on the specified signal. Due to limited space we show only the overall tracking error ϵ for the specified delays. Figure H.6 correspond to oracle and human delays, Figure H.7 to the AS and Applied signal delays. Delays can

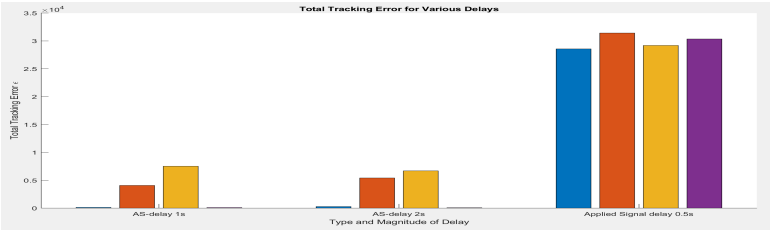


Figure H.7: Total tracking error for double lane change with given delays.

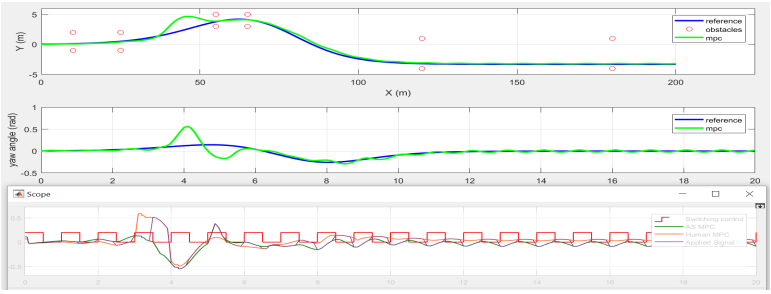


Figure H.8: Leader Follower Total tracking error for double lane change with noise injected to u_H at t=3s.

represent a variety of factors such as computation time for AS and Oracle and delayed human response and are common in real systems. Hence evaluating the resilience to delays for the frameworks is of utmost importance.

6.7 Experiment 4: Noise Rejection

Our final experiment concerns the effects of sporadic noise on the dominant controller. For this we injected a single offset of 0.5 at $t = 3s$ for a duration of 1 second into the dominant control signal. We evaluate each approach in detail and show the paths taken. Figure H.8 shows the tracking results for the LF framework.

The deviation due to noise injected is clearly visible in the Y position tracking (top). The time taken to re-stabilize on the path is about 2 seconds. This is due to the fact the LF framework periodically switches the actual signals, seen in the bottom plot, which can be analysed to find that the human had just started its control period when the noise was injected. This means the full effect of the noise were experienced for 1 seconds before the AS could intervene and we see a large deviation. We also notice slight trashing for the remainder of the simulation Y position but more significantly in the steering angle.

Figure H.9 shows the Oracle-based Switching simulation results. The deviation is a lot smaller since this framework identifies the bad trajectory and switches controllers. This is again obvious from the bottom chart in this Figure. We see the red line representing

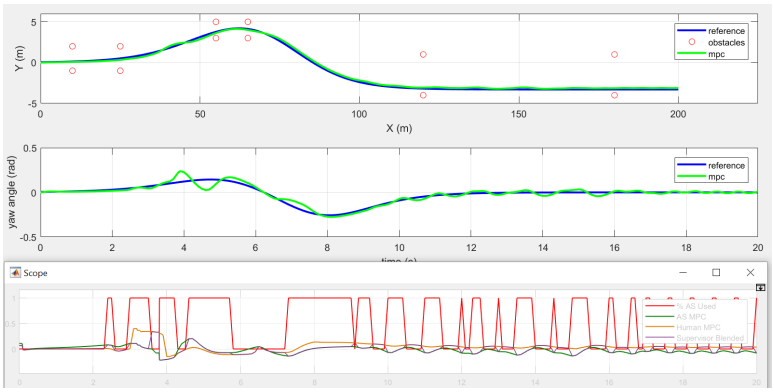


Figure H.9: Oracle-based Switching Total tracking error for double lane change with noise injected to u_H at t=3s.

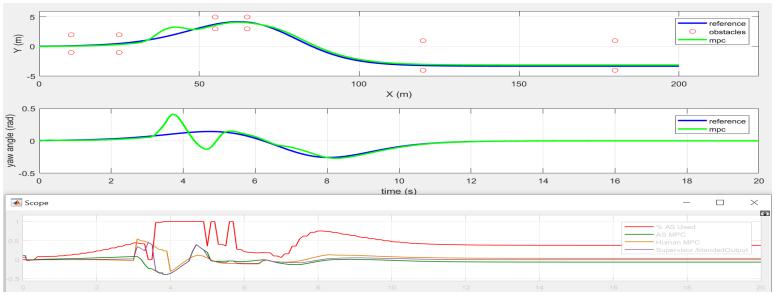


Figure H.10: Human Dominant Blending Total tracking error for double lane change with noise injected to u_H at t=3s.

the currently controlling agent switch to the AS slightly before the u_H noise was injected. We assume this was before the path deviated before the noise and hence the AS agent took control and the effects of the noise were limited. However, we still see slight thrashing in the steering angle for the remainder of the simulation.

Figure H.10 describes the Human Dominant Blending under noisy conditions. The deviation is significant but the framework is able to stabilize and execute the manoeuvre. We notice the framework started using the AS before the injection of noise which is why the effects were not as severe as for LF. Note here that there is no thrashing on the yaw angle and both tracking objectives are smooth.

This is the final framework we test again this scenario. One difference to the other Noise injection experiments is that we applied the noise to the AS Signal instead of Human signal, since having it on the Human will have no effect. Even with that change this framework performs the best overall. It is very smooth in the lead up to the noise and then recovers quickly. The deviation is still significant, however. The bottom chart shows that the blend of signals, in contrast to all other frameworks, is exceptionally smooth and it resumes close tracking within 3 seconds.

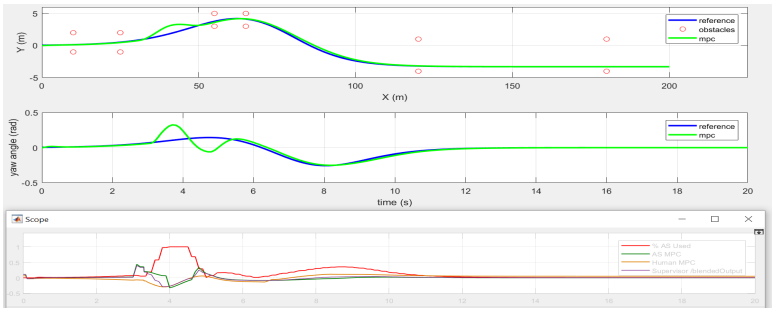


Figure H.11: AS Dominant Blending Total tracking error for double lane change with noise injected to u_{AS} at t=3s.

Finally, Figure H.5 (right) shows the comparison of total tracking errors for this experiment. Even though the LF framework experienced a large deviation and trashing, it still tracks accurately, since it assigns 100% of the control to a single controller, allowing that controller to get very good tracking for that time. It is difficult for Blending to achieve this level of tracking once the agents signals start to diverge, since the u will always be somewhere **between** u_H and u_{AS}, but it's rare that one agent has full control to generate an un-modified (optimal) control signal. This is an advantage the Switching frameworks have over the Blending. In contrast, this same phenomenon causes trashing on the output as both controllers compete for control of the system.

To summarize our results:

- *Bad u_H signals:* the less reliant the framework is on u_H the better it will perform. This is obvious from Experiments 1 and 2 which show AS blending performing the best.
- *Signal delays:* Signal delays can effect performance dramatically. AS Dominant Blending is the most resilient to delays on all signals expect when u gets delayed. But performance from all frameworks is catastrophic in this scenario.
- *Sporadic noise:*. Even though position tracking might look good, in reality there could be a lot of control thrashing which would effect the driving quality. AS Dominant Blending once again performed the best, even though marginally, and by far smoother than the switching frameworks in regards to yaw control.

7 Conclusions

This article has described an investigation of the impact of degree of collaboration and communication on the ability to control a multi-agent system. We have defined a blended autonomous system as a multi-agent system in which several forms

collaboration are allowed. We defined the collaboration degree in terms of a control metric, and studied the impact of different control algorithms on achieving a set-point for a hydraulic benchmark.

Our results indicate that blended control at each step dominate the performance of the switching control approach for Stackelberg game theory. The blended approach converges more quickly and is significantly less likely to generate thrashing.

This study has potential ramifications for design of blended human/AS systems, e.g., car steering control. A clear set of control over-rides will probably be necessary to avoid instability in joint human/AS vehicular control.

Paper I

Learning Sufficient Low-Level Controller Parameters for Blended Control in Non-Stationary Conditions

Abstract

Multiple model adaptive control frameworks such as switched or blended control have become a standard way to deal with non-stationary operating conditions for many autonomous systems. The traditional low-level controller parameter tuning approaches optimize a low-level system controller for each operating condition separately such that a supervisory controller can switch between the low-level controllers as the operating condition changes. The number of required parameters to cover the control parameter space sufficiently using this approach grows significantly as more conditions are considered. Blended Control is an alternative scheme where the control output is the convex combination of a set of low-level controller outputs, thus, any parameter contained in the convex hull of the low-level controller parameter set can be theoretically interpolated. The low-level controller parameter set should hence be distributed in such a way that the convex hull provides sufficient coverage of the parameter space to interpolate any required controller parameter. However, currently no convex hull-based tuning methods exist. In this article, we propose a theoretical basis for how blended control methods can complete a task in non-stationary operating conditions and present a novel learning-based algorithm that automatically constructs a convex hull in the low-level control parameter space to find a sufficient number of parameters to complete a task in such conditions. Compared to point-based estimation methods, our convex hull-based approach can reduce the size of the required controller

set. We conduct thorough empirical studies on a quadcopter trajectory-tracking task subject to faults and disturbances and show that the proposed method can automatically learn a controller set whose size is smaller than that defined through traditional tuning strategies, without incurring a significant performance loss.

1 Introduction

Autonomous systems such as self-driving cars, unmanned aerial vehicles (UAVs), and humanoid robots are becoming increasingly popular in society. These systems have vast application domains and require robustness to challenging non-stationary operating conditions, especially when operating in proximity to humans such as UAV delivery services in dense urban environments. Current fault-tolerant control methods typically require *a priori* knowledge of the system and expected operating conditions to design models and controllers specifically for those conditions [164]. However, it is impossible to pre-define a model and controller for every possible operating condition a system will experience in its operational lifetime at design time as there are simply too many to consider [103].

Multiple Model Adaptive Control (MMAC) algorithms based on a supervisory controller with multiple low-level controllers have become a standard way to deal with non-stationary operating conditions due to their decoupled and well-understood nature. The traditional approach is to use point-based estimation methods to find low-level controller parameters for a known set of operating conditions, and use a supervisory controller that can optimize the performance of the autonomous system by using a single or combination of low-level controllers.

Switching Control implements a supervisory controller that discretely switches between the low-level controllers. Achieving stability requires the existence of at least one controller tuned for each operating condition [106]. Therefore, switching-based control requires a number low-level controllers that increases exponentially with the dimension of the parameter vector and the number of operating conditions considered [165]. Given a controller defined over parameters θ, the optimum number of controllers required for fast convergence of a second-level model is 2^{θ} [166], which further increases as multiple operating conditions are considered.

Blended Control, also referred to as Adaptive Mixing Control [27], involves using a convex combination of the low-level controller outputs instead of discretely switching between them. This allows the supervisory controller to interpolate any low-level controller parameter contained within the convex hull of the parameter set of the low-level controllers [147]. The advantages of this include smoother transitions between controllers and the ability to interpolate a new low-level controller parameterized by any configuration contained within the convex hull of parameters of the low-level controllers [146]. This approach has been successfully used for partial fault conditions [2] but identifying the correct blending weights is significantly more complicated than

switching, as the fault identification step also requires the correct interpolation between controllers [147]. No work has been done to characterize the required number of low-level controllers, or the joint tuning of this collection of controllers for a blended control architecture.

The low-level controller parameters set used for blended control should be tuned specifically to generate a convex hull, as this ensures the set is affinely independent and sufficient to generate any required parameter. This is quite different from the traditional point-based controller tuning methodology developed for multiple model control systems, as the number of controllers required in a convex hull-based tuning approach does not increase with the number of operating conditions considered, given the required controller parameters for each condition are contained within the convex hull.

In this article, we present a novel learning-based algorithm that automatically finds a sufficient low-level controller set that allows a blended control architecture to stabilize a system under non-stationary operating conditions. The novelty of our approach is the construction of a convex hull in the controller parameter space instead of using point-based parameter estimation to find the low-level controller set. The extremes of the convex hull provide a blended control architecture with a sufficient parameter set. In comparison to point-based optimization methods that find a single parameter approximation for a fixed operating condition, the presented approach can optimize the low-level controllers over a continuous environment parameter range.

The contributions of this article are as follows:

A. We outline the theoretical underpinnings governing how a sufficiently large collection of low-level controllers in a blended multiple model adaptive control architecture can complete a task with non-stationary system dynamics.

B. We propose a novel learning-based algorithm for automatically generating a collection of low-level controllers that can solve particular control tasks given faults and environmental disturbances. The novelty is the convex hull-based tuning approach to find sufficient coverage of the controller parameter space for all conditions, instead of tuning one parameter for each condition separately.

C. We conduct thorough empirical studies on a quadcopter trajectory-tracking task subject to faults and disturbances. As a result, we found that the proposed method can automatically learn a controller set whose size is smaller than that defined through traditional tuning strategies, without incurring a significant performance loss.

The remainder of this article is organized as follows: Section 2 provides an overview of blended multiple model adaptive control. Section 3 gives the theoretical basis of the problem that we are interested in. Section 4 describes two learning-based approaches to find a low-level controller parameter set, followed by the presented approach. We discuss the quadcopter trajectory tracking task used for the empirical validation in Section 5 along with details of how to learn a set of attitude controllers for the non-stationary conditions. Section 6 empirically compares the controller sets found using our approach and a point-based optimization approach, followed by a conclusion in Section 7.

2 Background & Notation

MMAC architectures [164, 2] involve the decomposition of a control task into low-level system control and supervisory fault-tolerant control. A traditional approach is to design a low-level system controller, π_θ, for each operating condition $\lambda \in \Lambda$ (i.e. using multiple models) and discretely switch control between them which ensures the system is controllable under any considered operating condition.

Definition 28 (Controllability of Known Operating Conditions)**.** *Given a system defined over $\Lambda = \{\lambda_1, \cdots, \lambda_N\}$ operating modes, and a controller tuned for each mode, the system is controllable using a set of controllers $\Pi = \{\pi_{\theta_1}, \cdots, \pi_{\theta_N}\}$, such that controller θ_i is tuned to mode λ_i, $i = 1, ..., N$.*

Blended control, also referred to as Interacting Multiple Model [167] or Adaptive Mixing Control [168], is an instance of MMAC that uses the convex combination of all controller outputs. Given a system defined over N operating modes, each with a corresponding low-level controller, the supervisory controller uses an N-sized weight vector $\boldsymbol{\varphi} = \{\varphi_1, \cdots, \varphi_N\}$ to combine the outputs. The weight vector can be used to switch or interpolate between controllers providing more flexibility than discrete switching.

Definition 29 (Blended Control)**.** *Given a low-level controller set $\Pi = \{\pi_{\theta_1}, \cdots, \pi_{\theta_N}\}$ that generates an action vector $\boldsymbol{a} = \{a_1, \cdots, a_N\}$ and a corresponding weight vector $\boldsymbol{\varphi} = \{\varphi_1, \cdots, \varphi_N\}$, a blended control action is the weighted convex combination $\boldsymbol{a}^b = \sum_i \varphi_i a_i$ such that: (1) $\forall_i, a_i \in \boldsymbol{a}, 0 \leq \varphi_i \leq 1$, and (2) $\sum_i \varphi_i = 1$.*

Since Blended Control uses a convex combination of low-level controller outputs, the convex hull of the parameter set defines the possible parameters that can be interpolated. If the set contains a parameter that can be interpolated from the remaining

parameter set, it could theoretically be removed from the set without impacting the convex hull coverage. Tuning each controller parameter separately can result in a parameter set that is *affinely dependent*, i.e. some parameters in the set can be interpolated from a combination of the remaining set. We define the extreme controllers that define the convex hull as a *sufficient controller set* to ensure the system is controllable.

Lemma 2.1 (Sufficient Controller Set)**.** *Given a system defined over modes* $\Lambda = \{\lambda_1, \cdots, \lambda_N\}$ *and a controller set* $\Pi = \{\pi_{\theta_1}, \cdots, \pi_{\theta_M}\}$*, we call* Π *sufficient if*

- $\exists$ *some* $\boldsymbol{\varphi}$ *such that the blended control action* $\boldsymbol{a^b} = a^*_{\lambda_i}$*, where* $a^*_{\lambda_i}$ *is the action required to stabilize the system for each mode* λ_i*,* $i = 1, ..., N$*.*
- Π *is affinely independent, i.e. no controller in the set can be interpolated from the rest of the set.*

The first condition ensures that the controller parameter space covered by a sufficient controller set can generate the required control actions for all operating conditions. The second condition enforces that the controller parameter set only contains the extreme parameters since all other configurations can be interpolated from these. This is the key notion that point-based tuning approaches do not exploit. This is due to the point-based optimization methodology, where each operating condition is treated separately instead of considering the convex hull coverage of the parameter set for all conditions together.

Researchers have developed a wide array of control techniques for multiple model adaptive control [169], as well as methods for tuning individual low-level controllers [12, 170]. Existing methods for automated tuning, e.g., [12, 170, 171, 172] often make strong assumptions (e.g., linearity), which violate the inherent non-linearities of complex real-world systems. The obstacles of automated parameter tuning algorithms include computational complexity of parameter optimization [173], the difficulty of traversing the non-linear parameter space, as well as making gradient approximations from noisy measurements.

Significant progress in the theoretical underpinnings and practical design of optimization- and learning-based controllers that can deal with robotics systems involving e.g., multivariate dynamics, constraints, faults, and non-stationary operating conditions has been made. However, the implementation of fully automated algorithms to generate such advanced model-based controllers remains an open challenge. Further, these algorithms-as well as gradient-based optimization methods-require the environment parameters to remain fixed during the tuning process and are not able

to deal with a continuous parameter range, as often no single optimal solution for the entire range exists. The fixed models used for the tuning process are either systematically distributed over the environment parameter range to provide adequate controller coverage at the expense of using more controllers or using the worst-case parameter only [106].

Both supervisory and low-level controllers require *a priori* knowledge about the system and operating conditions, usually in the form of a model. The design of low-level controllers for unknown operating conditions, where no such prior information exists, is still considered an open problem. If the convex hull of the parameter set contains a feasible controller configuration for the unknown condition, the system is controllable, but the identification of the appropriate weight vector for such a scenario is also an open problem.

A recent extension to blended control, called Randomized Blended Control (RBC), exploits this property by using a uniformly randomized weight vector in the case of unknown operating conditions for which no control law exists. By using fast re-sampling of the weight vector, control can be distributed uniformly over the convex hull of the low-level controller set. RBC is to be a theoretically sound approach for stabilizing a system that is encountering novel operating conditions for which no control laws are known [103]. The major benefit of using randomization is the computational simplicity and speed which cannot be matched by identification based approaches. A clear drawback is that using a uniform distribution is sub-optimal and dependent on the parameter space coverage of the convex hull.

Lemma 2.2 (Controllability of Unknown Operating Conditions)**.** *Given unknown operating mode* $\lambda^U \not\subset \Lambda$ *and a controller set* $\Pi = \{\pi_{\theta_1}, \cdots, \pi_{\theta_M}\}$*, the system is controllable if* $\exists$ *some* $\boldsymbol{\varphi}$ *such that the blended control action* $\boldsymbol{a^b} = a^*_{\lambda^U}$*, where* $a^*_{\lambda^U}$ *is the action required to stabilize the system under the unknown operating mode.*

Several papers have shown that reinforcement learning can be used to learn a supervisory controller that generates an improved probability distribution used for randomized blending in real-time [174, 121, 175]. Given that a learning-based supervisory controller can be used to optimize the weights for unknown conditions, the remaining open problem is tuning the low-level controller set in such a way that the convex hull contains the parameters needed for the unknown condition. The presented learning-based framework is a step towards addressing this problem by directly constructing the convex hull in the controller parameter space based on the known operating conditions.

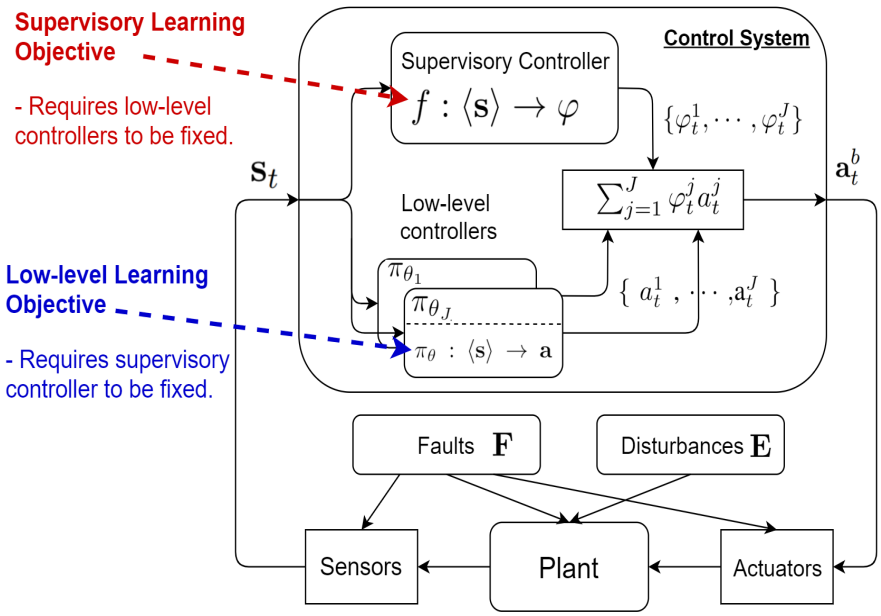


Figure I.1: Generic hierarchical control architecture.

Θ	Set of low-level controller parameters
π_{θ_i}	Control law function of controller i parameterized by θ
$\mathcal{A}$	Control action space
$\mathbf{a}_i$	Control action issued by controller i
$\mathcal{S}$	State space of the system
$\mathbf{s}_t$	State vector
$\mathbf{\Lambda}$	Set of operating modes in $\Re^n$ with n being the number of parameters of the system
λ_t	Operating mode vector
$\mathbf{E}$	Set of operating modes defined by environment parameters
$\mathbf{F}$	Set of operating modes defined by fault parameters
$\mathcal{T}$	Task
L	Loss function of a task
$\mathbf{a}_\lambda^*$	Required action for operating mode λ

3 Task: Theoretical Framework

3.1 Architecture

Fig. I.1 shows the generic architecture that we use for our approach. As shown, we have a hierarchical control system that uses a set Π of low-level controllers where

each controller π_{θ_i} is characterized by θ_i parameters and its respective control action is represented by $\mathbf{a}_i \in \mathcal{A}$. The supervisory controller assigns a system-level control consisting of a convex combination of $\mathbf{a}_i \in \mathcal{A}$.

In this article, we focus on learning the low-level controller parameters θ that generate the actions $\mathbf{a}_i \in \mathcal{A}$. In previous work [174, 121], we investigate a learning-based approach for the control parameters of the supervisory controller and hence omit discussion around finding the optimal blending weights from this article.

3.2 System Description

This section provides a top-level description of the system in which we are interested. We have a system, which consists of a plant and control system, operating in an environment. We characterize a system behavior using the tuple $\langle \mathcal{S}, \mathbf{\Lambda}, \mathcal{A} \rangle$. The state of the system is described by a state vector $\mathbf{s}_t \in \mathcal{S}$. We assume that the system can operate in a set of modes $\lambda_t \in \mathbf{\Lambda}$, which characterizes the plant/environment operating conditions, e.g., plant fault conditions or adverse environmental conditions. We represent a set of modes using a compact parameter set $\mathbf{\Lambda} \in \Re^n$, with n being the number of parameters. We can define a subset $\mathbf{E}$ of environment parameters (e.g., external wind) and a subset $\mathbf{F}$ of fault parameters (e.g., rotor faults), such that $\mathbf{\Lambda} = \mathbf{E} \cup \mathbf{F}$.

We characterize our system in terms of two parameter sets:

- a parameter set for the modes $\lambda_t \in \mathbf{\Lambda}$;
- a parameter set $\mathbf{\Theta}$ for the system controllers.

3.2.1 Stationary System Dynamics

If we assume that the system operates only in nominal plant states and the environment does not change, then we can define the dynamics using the function $\Sigma : \mathcal{S} \times \Lambda \times \mathcal{A} \rightarrow \mathcal{S}$, where

- $\mathcal{S}$ is the system state space;
- $\mathcal{A}$ is the action space;
- $\mathbf{\Lambda}$ is the mode state space.

In this scenario, we assume that $\mathbf{\Lambda}$ is fixed for any mission, i.e., we know the environment parameters *a priori* and that they do not change during a mission, and that faults do not occur.

3.2.2 Non-Stationary System Dynamics

In the situation where we allow changes in the plant and environment over time, we characterize such changes using mode transitions. Therefore, we can define the dynamics using the function $\Sigma_\lambda : \mathcal{S} \times \mathbf{\Lambda}(t) \times \mathcal{A} \rightarrow \mathcal{S}$.

3.3 Task Optimization for Stationary System Dynamics

Assume that we have a system aiming to complete a task $\mathcal{T}$ defined by the tuple

$$\mathcal{T} = (\mathsf{L}, T, H) \tag{I.1}$$

where

- L is a loss function
$$\mathsf{L} : \langle \mathbf{s}, \lambda, \mathbf{a} \rangle \rightarrow [0, 1] \tag{I.2}$$
that assigns a [0,1] loss at time t;
- $T : \mathcal{S} \times \Lambda \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a stochastic transition function that estimates the probability of reaching state $\mathbf{s}'$ at $t + 1$ when action $\mathbf{a}_t \in \mathsf{A}$ is taken by a controller of the vehicle ($T = P(\mathbf{s}_{t+1} = \mathbf{s}' | \mathbf{s}_t, \lambda, \mathbf{a}_t)$);
- H corresponds to the duration of the task.

The objective of solving a task is thus to minimize the loss L in solving task $\mathcal{T}$. We can formally state this as follows.

Definition 30 (Task optimization)**.** *Given a task $\mathcal{T}$, select a sequence of control actions $A_{1:H}$ that drive the system through a state/environment sequence $\mathcal{S}_{1:T}, \Lambda_{1:T}$ such that we minimize*

$$\sum_{t=1}^{H} \mathsf{L}(\mathbf{s}_t, \lambda, \mathbf{a}_t), \tag{I.3}$$

where $\mathbf{s}_t \in S_{1:H},\ \lambda \in \Lambda_{1:H}, a_t \in \mathcal{A}_{1:H}$.

Note that is this formulation the mode λ is fixed in the interval $1 : H$.

We assume that there exists an optimal control action at every time step, i.e.,

$$\exists\ \mathbf{a}_t^* = \arg\min_{\mathbf{a}_t \in \mathsf{A}} \mathsf{L}(\mathbf{s}_t, \lambda, \mathbf{a}_t). \tag{I.4}$$

Given a control policy $\pi_\theta : \langle \mathbf{s} \rangle \rightarrow \mathbf{a}$ characterized by the parameters $\theta \in \Theta$, the

optimization problem to solve a task for stationary system dynamics is the following

$$\theta^* = \arg\min_{\theta\in\Theta} \sum_{t=1}^{H} \mathsf{L}(\mathbf{s}_t, \lambda, \pi_\theta(\mathbf{s}_t)) \tag{I.5}$$

Problem I.5 can be solved if the following conditions are satisfied for task $\mathcal{T}$

A. $\mathsf{L}(\mathbf{s}, \lambda, \mathbf{a}) \leq C < \infty$, $\forall \mathbf{s} \in \mathcal{S}, \lambda \in \Lambda, \mathbf{a} \in \mathsf{A}$, and some bound C.

B. T and L do not change with time.

3.4 Task Optimization for Non-Stationary System Dynamics

In this section, we investigate the theoretical underpinnings allowing a blended control architecture to stabilize a system in non-stationary operating conditions (Contribution 1). A task for non-stationary system dynamics is defined by the tuple

$$\mathcal{T}^{\boldsymbol{\lambda}} = (\mathsf{L}, T^{\boldsymbol{\lambda}}, H), \tag{I.6}$$

where the behavior of the system depends on the parameter configuration vector $\boldsymbol{\lambda} \subset \boldsymbol{\Lambda}$ such that $T^{\boldsymbol{\lambda}} = P(\mathbf{s}_{t+1} = \mathbf{s}' | \mathbf{s}_t, \boldsymbol{\lambda}_t, \mathbf{a}_t)$. For example, the dynamics of a flying vehicle experiencing a rotor fault or wind conditions are different from nominal conditions. Therefore, the parameter vector encodes different system and environmental conditions.

We solve a task for non-stationary system dynamics by minimizing the loss function. However, a single control policy cannot solve such a task because the second condition of the optimization problem I.5 is not satisfied (T may change with time depending on the operating mode). Therefore, the optimization problems transforms into adapting the control policy parameters according to the operating mode. The optimization problem to solve a task for non-stationary system dynamics is the following

$$\min_{\theta\in\Theta} \sum_{t=1}^{H} \left(\mathsf{L}(\mathbf{s}_t^{\boldsymbol{\lambda}}, \boldsymbol{\lambda}_t, \pi_\theta(\mathbf{s}_t^{\boldsymbol{\lambda}}, \boldsymbol{\lambda}_t)) \right) \ \mathbf{s}_t^{\boldsymbol{\lambda}} \in T^{\boldsymbol{\lambda}}, \boldsymbol{\lambda} \subset \boldsymbol{\Lambda}. \tag{I.7}$$

Solving this optimization problem requires a control policy dependent on the operating mode. This implies that the operating mode has to be identified before attempting to solve the optimization problem. However, this is an issue for systems with small time constants that require fast adaptation mechanisms, such as unmanned aerial vehicles. Moreover, re-formulating I.7 as a policy selection problem is not feasible because the system/environment parameter space $\boldsymbol{\Lambda}$ is continuous thus a finite set of policies cannot

be designed beforehand.

As an alternative, we propose to reformulate I.7 as follows

$$\min_{\varphi} \sum_{t=1}^{H} \Big(\mathsf{L}(\mathbf{s}_t^{\boldsymbol{\lambda}}, \boldsymbol{\lambda}_t, \mathbf{a}_t^b) \Big) \;\; \mathbf{s}_t^{\boldsymbol{\lambda}} \in T^{\boldsymbol{\lambda}}, \boldsymbol{\lambda} \subset \boldsymbol{\Lambda}, \tag{I.8}$$

where $\mathbf{a}_t^b = \sum_{j=1}^{J} \varphi_j \pi_{\theta_j}(\mathbf{s}_t^i)$, $\sum_{j=1}^{J} \varphi_j = 1$, $\varphi_j \in [0, 1]$, and the boundary of the policy parameter space Θ_J is defined by the set of control parameter vectors $\delta_{\Theta_J} = \{\theta_1, \theta_2, ..., \theta_J\}$. Therefore, solving the policy blending problem translates into finding a supervisory controller that learns a mapping $f : \langle \mathbf{s}_t \rangle \rightarrow \varphi$ to determine the optimal blending vector φ for each operating mode $\boldsymbol{\lambda}$. An optimal solution for I.8 can be found if the following condition holds

$$\exists\, \theta_j \subset \Theta_J | \lim_{t \to \infty} (\mathsf{L}(\mathbf{s}_t^{\boldsymbol{\lambda}}, \lambda_t, \pi^*_{\theta_{\boldsymbol{\lambda}}}) - \mathsf{L}(\mathbf{s}_t^{\boldsymbol{\lambda}}, \lambda_t, \pi_{\theta_j})) = 0, \forall \boldsymbol{\lambda} \subset \boldsymbol{\Lambda}, \tag{I.9}$$

where $\pi^*_{\theta_{\boldsymbol{\lambda}}}$ is the optimal control policy for operating mode $\boldsymbol{\lambda}$. This conditions means that the controller parameter space Θ_J should contain the optimal control parameters corresponding to any operating mode in $\boldsymbol{\Lambda}$. If the policy parameter space is sub-optimal, then the equation in condition I.9 equals a constant value instead of zero.

The solution to I.8 depends on having the optimal control parameter space Θ_J. We formulate the following optimization problem for obtaining Θ_J

$$\arg \min_{\Theta_J \subset \Theta} [\xi \sum_{t=1}^{H} \mathsf{L}(\mathbf{s}_t^{\boldsymbol{\lambda}}, \boldsymbol{\lambda}_t, \mathbf{a}_t^b) + (1 - \xi) |\delta_{\Theta_J}|], \forall \boldsymbol{\lambda} \in \boldsymbol{\Lambda}, \tag{I.10}$$

where ξ is a regularization parameter, $\mathbf{a}_t^b = \sum_{j=1}^{J} \varphi_j \pi_{\theta_j}(\mathbf{s}_t^{\boldsymbol{\lambda}}, \lambda_t)$, $\sum_{j=1}^{J} \varphi_j = 1$, $\varphi_j \in [0, 1]$, and φ is uniformly distributed over the $(J-1)$-dimensional simplex.

Given that Θ_J can be formed based on the set of control parameters that determine its boundary, we formulate two methodologies for finding the set $\delta_{\Theta_J} = \{\theta_1, \theta_2, ..., \theta_J\}$. We call the set of controller parameters that form δ_{Θ_J} low-level controller set. Both proposed methodologies rely on having a physics simulator of the real system available and work under the assumption that a single parameter change defines each operating mode. For a real system, this implies that multiple faults are not considered.

3.4.1 Low-level controller set optimization: a parameter coverage approach

Given the set of operating modes $\boldsymbol{\Lambda} \in \Re^n$, this approach works under the assumption that the bounds of each parameter $\lambda_{min_i} \leq \lambda_i \leq \lambda_{max_i}$ in $\boldsymbol{\lambda}$ are known. This can be interpreted as knowing the minimum and maximum magnitude of each parameter that

defines the change of operating conditions. For example, for a loss of effectiveness fault in the motor of an unmanned aerial vehicle, it implies knowing the minimum and maximum loss of effectiveness.

Knowing the range for each of the n parameters that define $\boldsymbol{\lambda} \in \Re^n$, we independently generate L instances for each λ_i by uniformly sampling from its respective interval. Having a set of operating modes, $\boldsymbol{\lambda}_{11}, \boldsymbol{\lambda}_{12}, \cdots, \boldsymbol{\lambda}_{1L}, \boldsymbol{\lambda}_{21}, ..., \boldsymbol{\lambda}_{nL}$ ($\boldsymbol{\lambda}_{11} = [\lambda_1, 0, ..., 0]^T$, $\boldsymbol{\lambda}_{1L} = [\lambda_L, 0, ..., 0]^T$, ..., $\boldsymbol{\lambda}_{nL} = [0, 0, ..., \lambda_L]^T$), we solve the following problem for each instance

$$\arg\min_{\theta_{nL}} \sum_{t=1}^{H} \mathsf{L}(\mathbf{s}_t^{\boldsymbol{\lambda}_{nL}}, \lambda_t, \pi_{\theta_{nL}}). \tag{I.11}$$

The controller parameter space can be approximated therefore through the convex hull defined by the obtained parameter set $\Theta_J = \{\theta_{11} \bigcup \theta_{12} \bigcup ... \bigcup \theta_{1L} \bigcup \theta_{21} \bigcup ... \bigcup \theta_{nL}\}$. We therefore reconstruct the edges of the cloud of points in Θ_J to select the closed boundary δ_{Θ_J}. Many algorithms, such as Quickhull [176], for calculating convex hulls exist and the problem has been well studied. This approach is computationally expensive since it involves the solution of nL optimization problems. Based on the use of particle filtering optimization method for stochastic variables, we present a computational approach more tractable to solve this problem in the next section.

3.4.2 Low-level controller set optimization: a worst-case scenario approach

Given the set of operating modes $\boldsymbol{\Lambda} \in \Re^n$, this approach works under the assumption that the instance of each parameter causing the worst performance is know, that is

$$\boldsymbol{\lambda}_{i,worst} = \arg \max_{\boldsymbol{\lambda}_{i,worst} \in \boldsymbol{\lambda}_i} \sum_{t=1}^{H} \mathsf{L}(\mathbf{s}_t^{\boldsymbol{\lambda}_{i,worst}}, \lambda_t, \pi_\theta), \forall \theta \in \Theta. \tag{I.12}$$

For example, $\boldsymbol{\lambda}_i = [\lambda_i, 0, ..., 0]$ for $i = 1$. This means that problem I.12 does not have to be solved because each worst parameter configuration is known in advance. Under this assumption, we solve the following problem for each $\boldsymbol{\lambda}_{i,worst}$ with $i = \{1, 2, ..., n\}$

$$\arg\min_{\theta_{i,worst}} \sum_{t=1}^{H} \mathsf{L}(\mathbf{s}_t^{\boldsymbol{\lambda}_{i,worst}}, \lambda_t, \pi_{\theta_{i,worst}}). \tag{I.13}$$

Obtaining therefore the set of controller parameters that define $\delta_{\Theta_J} = \{\theta_{1,worst}, \theta_{2,worst}, ..., \theta_{n,worst}\}$. This approach is computationally more tractable than the parameter coverage approach. However, if sub-optimal solutions for each optimization problem are found then the boundary defining Θ_j will define a sub-optimal controller parameter space.

4 Approach

In this section, we present two particle-based optimization algorithms for learning a low-level controller sets. Particle-based algorithms [177] have been used in many robotics applications including localization [178], PID controller tuning [179, 180], and many more. For controller tuning, each particle represents a possible controller parameter configuration θ and over time the algorithm learns an approximation of the optimal controller parameters θ^* on a given task. These algorithms typically vary in the way the particles traverse/represent the parameter space. A major benefit is the small number of hyper-parameters learning-based algorithms require compared to model-based controller optimization and these parameters are generally well understood [42]. We evaluate the loss L for a particle configuration θ on a task $\mathcal{T}$ characterized by operating condition λ using the notation $\mathsf{L} = eval(\theta, \mathcal{T}^{\lambda})$. The optimization algorithms learn to minimize the loss obtained on a task by searching the controller parameter space.

4.1 Particle Swarm Optimization (PSO)

Particle Swarm Optimization is a nature inspired population-based optimization algorithm used to approximate the global optimum in a search space. Each particle in the Z-sized population is initialized with a random position (controller parameters) and velocity. The particles traverse the search space based on a global and local estimate of the optimal parameter configuration at each epoch until eventually converging on a single optimal solution [179]. At each iteration the velocity of each particle is updated based on the previous velocity (v^z_{t-1}), a cognitive and social component (cog and soc), indicating the current local and global estimate of the best performing controller parameters. Both cognitive and social components are scaled by tune-able parameters ($c1$ and $c2$ respectively) indicating their influence in the velocity updates as well as a random component ($r1$ and $r2$). We use θ^z_{best} to indicate the local optimum for each particle in the population and θ^g_{best} as the global optimum of all particles. The parameter tuning for PSO is discussed in detail in [181]. The full PSO algorithm can be seen in Algorithm 1 and returns a single approximation of the optimal controller parameters for task $\mathcal{T}^{\lambda}$ characterized by operating condition λ. One drawback of this algorithm that it cannot find optimal solutions for non-stationary tasks. The algorithm is repeated for each worst-case operating condition parameter to construct a set of low-level controllers.

Result: Approx. of optimal solution θ^*_λ for Task $\mathcal{T}^\lambda$
Input: $c1, c2, Z, maxIter, \mathcal{T}^\lambda$;
initialize θ^Z_0 and v^Z_0 randomly for Z particles;
for $i = 1$ *to* $maxIter$ **do**
  for $z = 1$ *to* Z **do**
    $cog = c1r1(\theta^z_{best} - \theta^z_{i-1})$;
    $soc = c2r2(\theta^g_{best} - \theta^z_{i-1})$;
    // Velocity update ;
    $V^z_i = V^z_{i-1} + cog + soc$;
    // Position update ;
    $\theta^z_i = \theta^z_{i-1} + V^z_i$;
    // Evaluate Particle ;
    $\mathsf{L} = eval(\theta^z_i, \mathcal{T}^\lambda)$;
    // Local Update ;
    if $\theta^z_{best} > \mathsf{L}$ **then** $\theta^z_{best} = \mathsf{L}$;
  end
  // Global Update ;
  update θ^g_{best};
end
return θ^g_{best};

Algorithm 1: Particle Swarm Optimization

4.2 Particle Filtering (PF)

Particle Filtering is a Monte Carlo-based method using the recursive computation of relevant probability distributions using the concepts of sequential importance sampling [42]. Instead of a velocity, each particle is assigned a weight w^z_i which is updated based on the performance achieved by the particle configuration on a task. A Z-sized set of particles is used to represent the probability density function of the optimal solutions in the parameter space but the position (controller parameters) of the particles remains fixed throughout the learning cycles. During each iteration, importance sampling is used to sample a K-sized subset of particles, θ^k_i, based on their associated weights. The cumulative performance L_T of all particles on a task $\mathcal{T}$ is used to update the particle weights after each iteration. This results in better-performing particles obtaining a larger weight over time and after a fixed number of epochs, the best performing controller parameters are the ones with the largest weights. A core difference between the PSO and PF algorithms is that PSO returns a single controller parameter while PF can have multiple equally weighted particles which represent a set of possible controller parameters.

Result: Extremes of Polytope Θ^*_Λ
Input: θmin, θ_{max}, σ, K, Z, $maxIter$, $\mathcal{T}$, Λ ;
initialize Z particles, Θ^Z, with uniformly distributed position over interval [θ_{min}, θ_{max}] and $w_0^Z = 1/Z$;
foreach $\lambda_i \in \Lambda$ **do**
  // –Step 1: Particle Filtering– ;
  for $i = 1$ *to* $maxIter$ **do**
    // Importance Sampling ;
    $\theta_i^K \sim P(\theta^z | w_{i-1}^z)$;
    foreach $\theta^k \in \theta_i^K$ **do**
      // Evaluate Particle ;
      $\mathsf{L} = eval(\theta^k, \mathcal{T}^{\lambda_i})$;
      // Weight Update ;
      $w_i^k = \mathsf{L}/\mathsf{L}_T$;
    end
    update L_T;
  end
  $\Theta_{\lambda_i} = \{\theta^z \in \theta^Z | w_i^z \geq \sigma\}$;
  // –Step 2: Clustering– ;
  $\Theta^C_{\lambda_i} = CLUSTER(\Theta_{\lambda_i})$
  // –Step 3: Convex Hull– ;
  $\Theta^*_{\lambda_i} = HULL(\Theta^C_{\lambda_i})$
end
//–Step 4: Combined Convex Hull– ;
$\Theta^*_\Lambda = HULL(\{\Theta^*_{\lambda_1} \bigcup \cdots \bigcup \Theta^*_{\lambda_J}\})$
return Extremes of Θ^*_Λ

Algorithm 2: Clustered Particle Filtering

4.3 Clustered Particle Filtering (CPF)

In this section, we present our algorithm (Contribution 2), referred to as Clustered Particle Filtering (CPF), used to find a sufficient number of low-level controllers for all known operating conditions. The algorithm combines three well-known algorithms, namely Particle Filtering [42], DBSCAN [182] (clustering), and Quickhull [176] (convex hull), in a novel way.

The algorithm is divided into three main steps, which are repeated for every operating condition $\lambda_i \in \Lambda$ followed by a final step. First, we run PF to get an $\mathcal{M}$-sized set of N-dimensional controller parameters that perform above a threshold σ, denoted $\Theta_{\lambda_i} = \{\theta_1, \cdots, \theta_\mathcal{M}\}$. Here, σ is a tunable hyper-parameter indicating the maximum acceptable loss on a task. Secondly, we remove outlier configurations using a clustering algorithm such as DBSCAN, resulting in a dense cluster of controller

parameters $\Theta^{C}_{\lambda_i} = CLUSTER(\Theta_{\lambda_i})$. Thirdly, we construct the convex hull around the cluster using the Quickhull algorithm denoted $\Theta^{*}_{\lambda_i} = HULL(\Theta^{C}_{\lambda_i})$, which represents an N-dimensional polytope of controller parameters that perform above threshold σ on a task characterized by operating condition λ_i.

After repeating this process for every operating condition, the final step is combining the N-dimensional polytopes found for J operating conditions resulting in a new polytope that contains controller parameters for all known operating conditions, $\Theta^{*}_{\Lambda} = HULL(\{\Theta^{*}_{\lambda_1} \bigcup \cdots \bigcup \Theta^{*}_{\lambda_J})\}$. When the extremes that define Θ^{*}_{Λ} are used as low-level controllers for a blended control architecture, the possible low-level controller combinations contain parameter configuration for all known conditions and hence represent a sufficient controller set. For now, we manually select the extreme parameters and will integrate automatic extremum-generation methods in future work. The full algorithm can be seen in Algorithm 2.

The filtering, clustering, and convex hull algorithms used in our approach can be replaced with an equivalent algorithm. The novelty of our approach lies in how these methods are combined to find the controller set. The hyper-parameter tuning for each of these algorithms is also well understood. Our approach only requires the tuning of one additional parameter, σ, which indicates the performance threshold on a given task and is dependent on the task itself.

5 Application Domain: Quadcopter

This section describes the application domain that we use to illustrate our approach, the quadcopter UAV. Quadcopters and other unmanned aerial vehicles have become increasingly popular due to their immense application potential in fields like transportation, reconnaissance [183], search and rescue [184], drone racing [185], precision agriculture [186] , forestry [187] and many more. The physics of these vehicles are well understood with a wide variety of low-level control architecture and models are openly available [188, 189]. Failures in UAVs can have catastrophic consequences which makes fault tolerance for these vehicles of paramount importance [190].

5.1 Dynamics

The quadcopter airframe dynamics are modeled based on the UAV model presented in [191]. The quadcopter is composed of a central hub with four arms extending radially

and a motor at the end. The derived body forces considered are:

$$F_b = F_M + F_D + mgR_{IB}\mathbf{e}_z, \tag{I.14}$$

where $F_b \in \Re^3$ is the resulting force acting on the body frame, $F_M \in \Re^3$is the resultant force generated by the motors, $F_D \in \Re^3$ is the drag force resulting due to the movement of a UAV through the air, m is the mass of the UAV, g is the gravity acceleration, $R_{IB} \in \Re^3$ is the rotation matrix from the inertial frame to the body frame, and $e_z = [0\ 0\ 1]^T$.

The rotation matrix is calculated based on the Euler angles $[\phi, \eta, \psi]$ [191]. The order of rotations from the inertial frame to the body frame considered in this work is yaw (Z)-pitch (Y)-roll (X) assuming the UAV is moving forward in the positive X direction. The equations of motion derived from equation (I.14) are

$$\dot{\mathbf{v}}_b = \frac{1}{m}(F_M + F_D) + gR_{IB}\mathbf{e}_z - \mathbf{w}_b \times \mathbf{v}_b \tag{I.15}$$

$$\dot{\mathbf{w}}_b = I_b^{-1}(M_M - \mathbf{w}_b \times I_b\mathbf{w}_b), \tag{I.16}$$

where $\mathbf{v}_b \in \Re^3$ and $\mathbf{w}_b \in \Re^3$ represent the linear and angular velocity vectors of the UAV in the body frame, M_M is the torque generated by the motors, and $I_b \in \Re^{3\times3}$ is the inertial matrix of the UAV. Next, we describe how each force and torque is calculated.

The four rotating motors including propellers are used to generate motor forces F_M and torques M_M. For each motor i, the force and torque generated are given by

$$F_{M_i} = c_T\omega_{M_i}^2(1 - \gamma_i) \tag{I.17}$$

$$M_{M_i} = c_Q\omega_{M_i}^2(1 - \gamma_i) \tag{I.18}$$

where c_T is the coefficient of thrust and c_Q is the torque constant. The net force applied to the airframe is the summation of the forces generated by the motors. Rotor faults represent a partial or full loss of effectiveness (LOE) denoted by $\gamma_i \in [0, 1]$, where 1 indicates full failure and 0 normal operation. The net torque acting on the quadcopter arises from the aerodynamics (the combination of the produced rotor forces and air resistances) applied to the vehicle.

Form drag is the most common and easily modeled aerodynamic effect. Form drag arises due to the movement of a reference area through a fluid. The general expression of the form drag force is

$$F_{Drag} = -\frac{1}{2}\rho C_D A\mathbf{v}_b^2 \tag{I.19}$$

where ρ is the air density, C_D is the drag coefficient, A is the reference area that is perpendicular to the velocity of the object v_b. Drag force generated due to translation of the quadcopter is given by

$$F_D = -\frac{1}{2}\rho C_D \begin{pmatrix} A_{yz} v_{b,x} |v_{b,x}| \\ A_{xz} v_{b,y} |v_{b,y}| \\ A_{xy} v_{b,z} |v_{b,z}| \end{pmatrix} \tag{I.20}$$

where $|.|$ is the absolute value (necessary since the aerodynamic drag always act in opposite direction of the velocity vector), A represents the cross-sectional area of the UAV in each plane, and $v_{b,i}$ represents the velocity of the i axis in the body frame. The torque generated due to drag is considered negligible in this work.

Position and attitude of the quadcopter are estimated using on-board sensors subject to noise. We model noise using a zero-mean gaussian distribution of varying magnitudes. Position noise is added to x, y and z in meters and attitude noise to ψ, θ and ψ in radians.

5.2 Control Task

The task of the quadcopter in this work is focused on trajectory tracking. A trajectory is a temporally-indexed set of coordinates in 2D or 3D, denoted $\boldsymbol{\zeta}_t$ over T time points. We denote the reference (desired) trajectory as $\boldsymbol{\zeta}_t^R$, and the executed trajectory as $\tilde{\boldsymbol{\zeta}}_t$ at time t. The goal of a trajectory tracking task can be defined as minimizing the total trajectory deviation error.

For the large-scale application of autonomous flying vehicles, there must be rules to avoid catastrophic collisions. Federal agencies aiming to introduce regulations for UAV traffic management are focusing on allocating 4D operational volumes that are temporally and spatially separated for each vehicle to minimize the possibility of collisions [192]. In the event of faults, the primary focus of the control system should be to stay within the allocated operational volume as this minimizes the impact on the surrounding vehicles. In this work, we consider the operational volume as a sphere with radius $\mathcal{R}$ around each coordinate in the reference trajectory $\boldsymbol{\zeta}_t^R$. This creates an operational tube surrounding a trajectory, which can be seen in Fig. I.2. We can define the number of steps outside the allocated operational volume as:

Definition 31 (Operational Volume Loss)**.** *Given a reference trajectory $\boldsymbol{\zeta}_t^R$, the executed trajectory $\tilde{\boldsymbol{\zeta}}_t$ and the radius of the operational volume $\mathcal{R}$, we define the total*

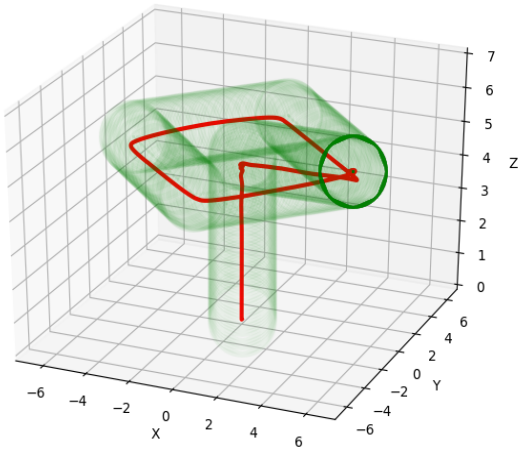


Figure I.2: Example of Operational Volume. The red line shows a trajectory with the green surrounding area representing the operational volume associated with it.

operation volume loss $\mathsf{L}_{0:T}$ *as :*

$$\mathsf{L}_{0:T} = \sum_{t=0}^{T} \mathsf{L}_t \tag{I.21}$$

$$\mathsf{L}_t = \begin{cases} 1, & \textit{if } |\zeta_t^R - \tilde{\zeta}_t| \geq \mathcal{R} \\ 0, & \textit{otherwise} \end{cases} \tag{I.22}$$

for a trajectory over time $t = 0, \cdots, T$.

The objective for fault tolerant control of a quadcopter can be defined as minimizing $\mathsf{L}_{0:T}$.

5.3 Control Architecture

Cascading control architectures have become the industry standard for commercial quadcopters due to their well-understood nature [193]. Position control and attitude control are separated into subsystems: the outer (position) control loop generates the required attitude reference and the inner (attitude) control loop maps these to the required torques and forces. A control allocation algorithm determines the reference rotor commands based on the torques and forces. The state vector of the quadcopter is defined as $\boldsymbol{s_t} = [x\ \dot{x}\ y\ \dot{y}\ z\ \dot{z}\ \phi\ \dot{\phi}\ \eta\ \dot{\eta}\ \psi\ \dot{\psi}\ ]$, where x, y, z indicate the position in space and ϕ, η, ψ the roll, pitch and yaw angles. The actions consist of the four motor throttle commands $\boldsymbol{a_t} = [U_1, U_2, U_3, U_4]$. The system evolves according to:

$$\boldsymbol{s_{t+1}} = T(\boldsymbol{s_t}, \lambda, \boldsymbol{a_t^b}) \tag{I.23}$$

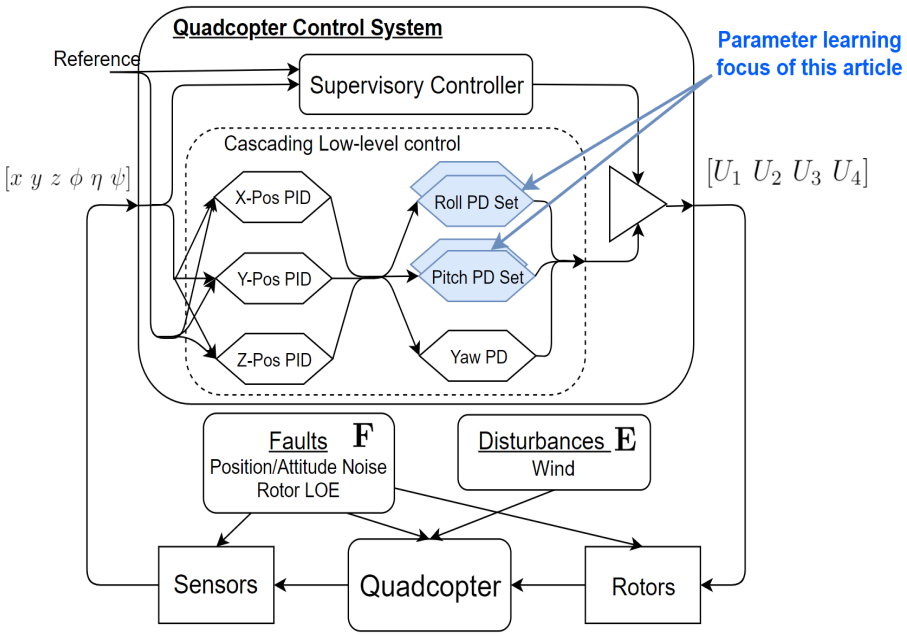


Figure I.3: Quadcopter cascading control architecture. The learning focus of this article are the roll and pitch PD controller sets.

$$\boldsymbol{a_t^b} = \sum_{j=1}^{J} \varphi_j \pi_{\theta_j}(\boldsymbol{s_t}), \tag{I.24}$$

where a_t^b is the blended control action of J controllers at time t, such that $\sum_{j=1}^{J} \varphi_j = 1$ and $\varphi_j \in [0, 1]$. We use PID and PD controllers for the outer and inner control loop respectively which is a well known quadcopter control configuration [194]. The full modeling details and dynamics of cascading PID control of a quadcopter can be found in [194] and are omitted for brevity. The important factor is the dimensionality of the attitude controller parameters as these parameters are the learning focus of this article. In this control configuration, the roll and pitch controllers are 2-dimensional, i.e. $\theta = [K_p, K_d]$. Since quadcopters are symmetric, both roll and pitch controller uses the same PD gains. The hierarchical control architecture can be seen in Fig I.3. CPF learns a polygon, while PSO learns a point, for each operating condition.

5.3.1 Learning Attitude Controller Sets

The list of operating conditions and corresponding parameters can be found in Table I.1. For the tuning process, PSO uses the worst-case parameter (λ_i^{max}) and CPF randomly samples from the continuous parameter range ($\lambda_i^{min} \leq \lambda_i \leq \lambda_i^{max}$) at each epoch. The minimum for each parameter is 0 and represents nominal conditions, $\lambda_i^{min} = \lambda_0$. The discrete partition seen in Table I.1 is used in the next section and added here for brevity.

Table I.1: Quadcopter operating conditions indicated using sub-script and fault magnitudes used in experiment section using superscript.

Operating Conditions	**Index**	λ^1	λ^2	λ^3	$\lambda^{4/max}$
Nominal	λ_0	-	-	-	-
Attitude Noise (rad)	λ_1	0.15	0.3	0.45	0.6
Position Noise (m)	λ_2	0.9	1.8	2.7	3.6
Wind (m/s)	λ_3	3	6	9	12
Rotor LOE (%)	λ_4	5	10	15	20

We are interested in five different operating conditions, namely nominal, attitude and position sensor noise, wind, and rotor loss of effectiveness. Traditional FTC approaches optimize one controller for each worst-case operating condition, resulting in the controller parameter set $\Theta_{\Lambda}^{PSO} = \{\theta^*_{\lambda_0}, \theta^*_{\lambda_1}, \theta^*_{\lambda_2}, \theta^*_{\lambda_3}, \theta^*_{\lambda_4}\}$, where $\theta^*_{\lambda_i}$ represents the parameter configuration learned using PSO for each operating condition $\lambda_i^{max} \in \Lambda$.

CPF finds a polygon for each operating condition represented by a set of controller configurations $\Theta^*_{\lambda_i} = \{\theta_1, \cdots, \theta_J\}$, where J controller parameters perform above threshold σ for operating condition $\lambda_i \in \Lambda$. The operating parameters are sampled randomly from the parameter range instead of considering only the worst-case. The sets are combined as previously described to find the sufficient controller set $\Theta_{\Lambda}^{CPF} = \{\theta_1, \cdots, \theta_M\}$, where M represents the number of extreme parameters that cover the combined sets $\{\Theta^*_{\lambda_0} \bigcup \Theta^*_{\lambda_1} \bigcup \Theta^*_{\lambda_2} \bigcup \Theta^*_{\lambda_3} \bigcup \Theta^*_{\lambda_4}\}$. In comparison to traditional approaches that find a point, our approach constructs a polygon for each condition. This allows our approach to ensure the parameter set is affinely independent. The details of running PSO and CPF to obtain the low-level PD controller sets are presented in the appendix for the interested reader. In the next section, we compare the performance of the controller sets found PSO and CPF.

5.4 Simulator

We extend a python-based quadcopter trajectory tracking simulator [148] with the operating conditions of interest i.e. rotor faults, wind disturbances, and sensor noise as well as the operational volume functionality. We then create a wrapper to implement the optimization algorithms around a single run of trajectory tracking simulation parameterized by the roll and pitch PD controller configurations. The result of each simulation is the number of steps outside the operational volume when using a given PD configuration; the optimization algorithms aim is to minimize this. The full codebase[1] to run the experiments presented in this article is made available, and

[1] https://github.com/YvesSohege/Journal-RAS-Simulation

requires only python to execute. Additionally, we provide video demonstrations of the tuning processes used to obtain the controller sets in the repository.

6 Experiments

We demonstrate the feasibility and performance of our approach using the quadcopter trajectory-tracking simulator. We present the controller sets learned using PSO (Θ_Λ^{PSO}) and CPF approach (Θ_Λ^{CPF}) and then compare the average performance of both sets in a MMAC architecture over all operating conditions. We compare two supervisory architectures, traditional switching with optimal FDI (unrealistic baseline) and uniform randomized blended control (RBC). One drawback of our approach is that the resulting controller set is not suitable for switched control, as the controllers that are found are not fault specific. We hence omit switching using CPF controllers from the experiments.

6.1 Controller Parameter Sets

We can visualize the PD parameter space due to its low dimensionality. In Fig. I.4, we show the intermediate convex hulls ($\Theta_{\lambda_i}^* \in \Lambda$) learned using the CPF approach and the manually selected extreme parameter set (Θ_Λ^{CPF}, black diamond icon) to cover the hull. We overlay the PSO controller set (Θ_Λ^{PSO}, cross icon) and use colored polygons/crosses to distinguish between the five investigated operating conditions. Fig. I.4 (top) highlights the core concept of our approach - the construction of convex hulls rather than point-based tuning. In this example, the convex hull of the five sub-spaces ($\{\Theta_{\lambda_0}^* \bigcup \Theta_{\lambda_1}^* \bigcup \Theta_{\lambda_2}^* \bigcup \Theta_{\lambda_3}^* \bigcup \Theta_{\lambda_4}^*\}$) can be represented using only three extreme parameters compared to the traditional approach utilizing one controller per operating condition. Fig. I.4 (bottom) highlights that any controller found using PSO could be interpolated from the CPF controller set but not vice-versa. Further, the Θ_Λ^{PSO} set is *affinely dependent*, as the wind controller (blue cross) and position noise controller (green cross) could be interpolated from the remaining PSO controllers. The point parameters found using PSO are also contained within the corresponding convex hulls, found using CPF, for each operating condition. This shows that a convex hull based tuning method provides a more suitable basis to find the parameter set for a blended control architecture (Contribution 3) and that the size of the controller set can be reduced through the presented approach (Contribution 4). Our next step is to investigate if this reduction causes a significant loss in performance.

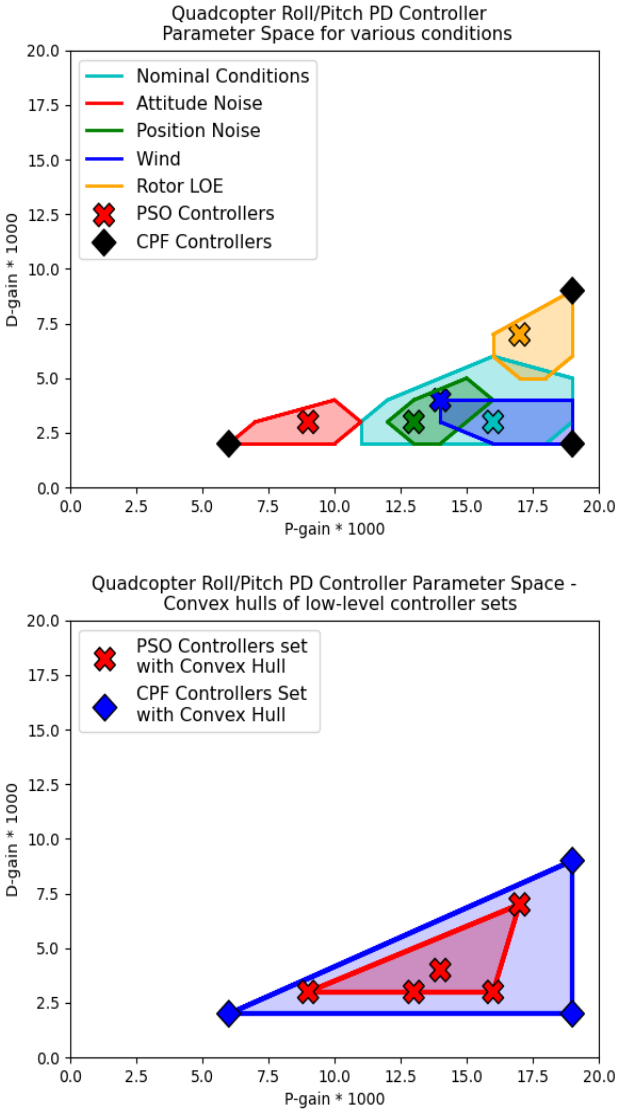


Figure I.4: Quadcopter attitude PD parameter space. Top: Coloured polygons represent the subspaces $\Theta^*_{\lambda_i}$ found through CPF for each operating condition. Diamond icons represent Θ^{CPF}_{Λ} and cross icons Θ^{PSO}_{Λ}. Bottom: Shows the convex hull coverage of the PD parameter space for each controller set.

6.2 Performance Comparison

We conduct two experiments to investigate the performance over fixed and continuous environment parameters. For the first set of experiments, the parameter range for each operating condition is divided into four fixed *levels* of increasing fault magnitude indicated by $\lambda_i^{level} \in \Lambda$, where λ_i^0 represents nominal conditions and every level a 25% increase in magnitude, which can be found in Table I.1. As a performance measure, the average operational volume loss, Eq. I.21, over 100 random trajectories for each level of each operating condition is used. We then compare the average performance

achieved across all conditions. The second set of experiments compares the same performance metric using a random environment magnitude, sampled uniformly from the continuous environment parameter range $\lambda_i^{min} \leq \lambda_i \leq \lambda_i^{max}$. A Friedman test is used to statistically compare the performance of the three architectures on 500 random trajectories with randomized magnitudes for each operating condition.

The Friedman test validated that there was no statistical difference in the performance achieved by the three architecture. This is visualized in Fig. I.5, which shows the detailed performance of the three architectures for experiment 1. The average performance over all operating conditions can be seen above each subplot and again shows that all three architectures perform comparably. The X and Y-axis of each subplot represent the level and type of the operating condition respectively. Each cell represents the average loss (number of steps outside of the operational volume) for each operating condition taken over 100 independent runs. We consider a task failed if the loss exceeds 500 ($\sigma = 500$) and on average a trajectory is completed in 2000-3000 steps.

We provide boxplots for each operating condition from experiment 2 in the appendix. Both experiments show that the controller set found using our approach can achieve comparable performance to the traditional tuning approach with a smaller controller set (Contribution 5). The CPF set performs slightly worse on average, and this is attributed to the larger convex hull coverage but the overall impact on the control task is not significant. It is important to note that this set of experiments serves as an empirical demonstration of the feasibility of our approach on a complex relevant system but we are not yet able to guarantee that the controller set found using our approach is always smaller than the considered number of operating conditions. We plan to investigate this in future work. However, we can guarantee that novel operating conditions whose optimal controller parameters fall within the convex hull of the existing controller set will not require additional controllers since a blended control architecture can theoretically interpolate the required low-level controller from within the hull. Our approach has an inherent trade-off over the size of the convex hull, the number of extremal controllers, and the coverage of the selected controllers: a large convex hull can cover more unknown operating conditions than a small convex hull, but it sacrifices the additional loss incurred in known operating conditions.

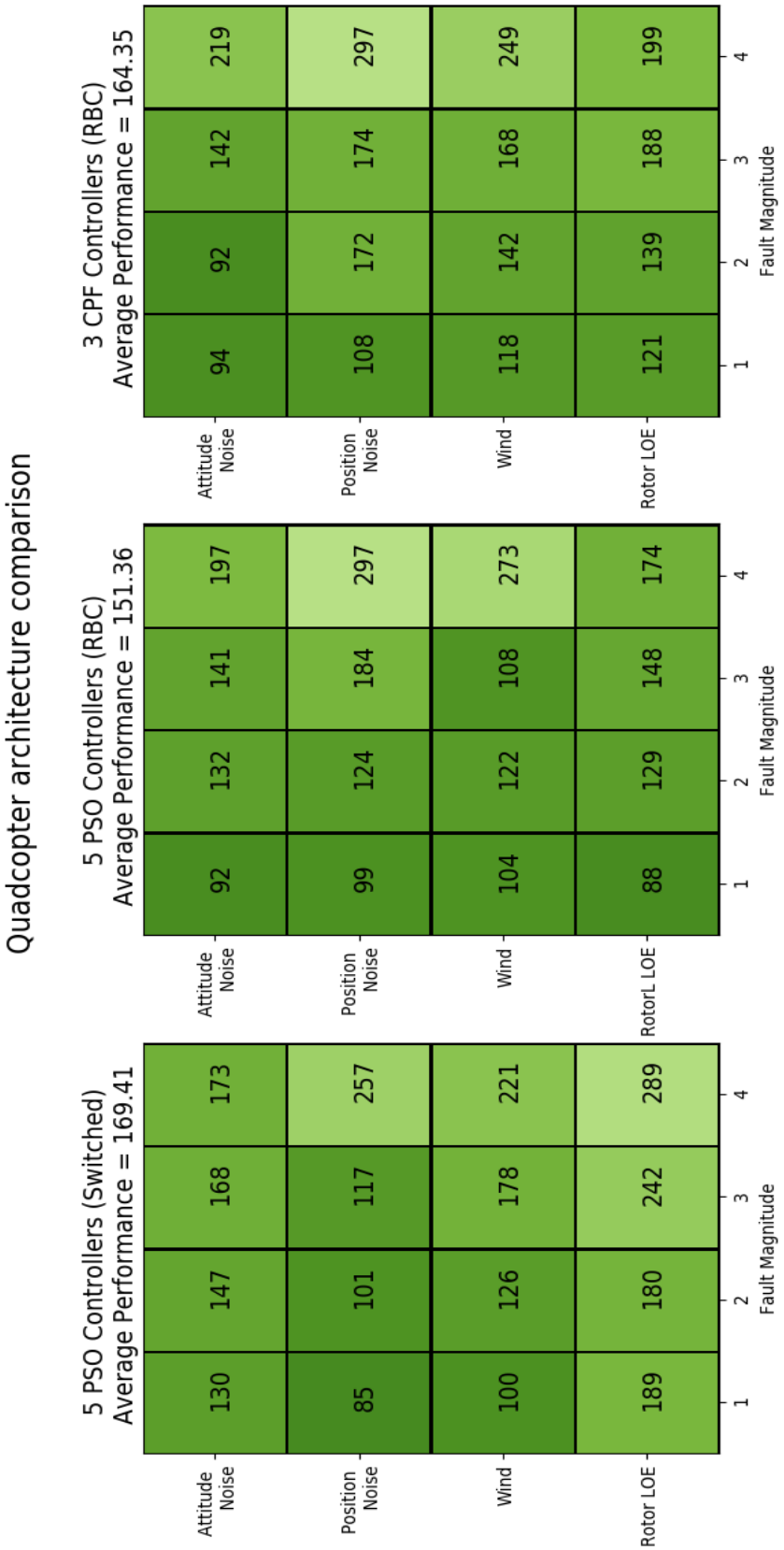


Figure I.5: Comparison average number of steps outside operational volume when using: 1) Switching using Θ_{Λ}^{PSO}, 2) Randomized Blended Control using Θ_{Λ}^{PSO} and 3) Randomized Blended Control using Θ_{Λ}^{CPF}, in various operating conditions. Each cell represents the average over 100 runs and X and Y axis define the operating condition parameters (also found in Table I.1).

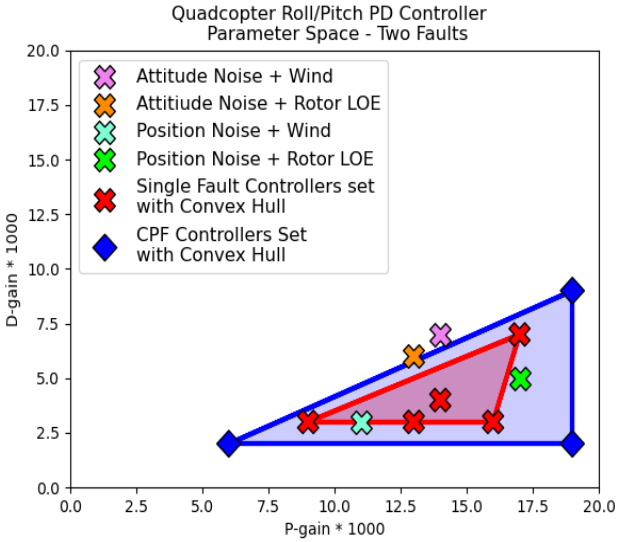


Figure I.6: Quadcopter attitude PD parameter space showing the convex hull coverage of the PD parameter space for each controller set and the controllers found using PSO on dual fault conditions.

6.3 Multiple Operating Conditions

We repeated the experiments with four novel operating conditions consisting of a combination of two of the existing operating conditions. The baseline PSO controller set is extended with four additional controllers for the new operating conditions. Fig I.6 shows the controller parameter approximations found using PSO on dual fault conditions and the convex hull coverage of the controller sets. The magnitude of the operating conditions is reduced by half during the tuning process, as larger magnitudes do not allow the learning algorithm to find a feasible controller. We compare a switching architecture with optimal FDI and 9 low-level controllers $\Theta_{\Lambda}^{PSO-ext}$ (4 single fault, 4 double fault and 1 nominal controller), randomized blending using the single fault PSO controller set (4 single fault and 1 nominal controller) and randomized blending using the CPF controller set consisting of 3 controllers. The Friedman test comparing the performance of the three architectures fails, highlighting a statistical difference in the achieved performance. The detailed performance on each level of the dual fault conditions, shown in Fig. I.7, demonstrates that the controller set found using both approaches can provide fault tolerance to novel operating conditions as long as the convex hull contains the required controller parameters. The wind and attitude noise conditions together seem to be particularly challenging as all architectures fail on the highest magnitude. These results demonstrate that adding more controllers does not necessarily improve the overall performance, and the controller set obtained through the CPF approach provides comparable performance with a significantly

smaller number of controllers. We plan on investigating novel operating conditions in detail in future work.

7 Conclusion

Traditional multiple model FTC approaches revolve around designing one controller for every fault or disturbance considered at design time. While this approach has been shown to work in practice for a limited set of known operating conditions, it is impossible to consider every possible fault *a priori*. Convex hull-based tuning methods are more suitable to sufficiently cover the controller parameter space, as point-based estimation methods do not ensure that the parameter set is affinely independent. This leads to significant growth in the size of the low-level controllers set when using traditional point-based tuning approaches. In this article, we presented the theory which allows a blended multiple model framework to complete a task in non-stationary operating conditions. A novel learning-based framework is presented that automatically finds a sufficient number of low-level controller parameters to solve a non-stationary control task. The novelty of our approach is the construction of a convex hull in the controller parameter space instead of using point-based estimation methods to obtain the low-level controller set. By using the extreme parameters of the convex hull as a controller set, our approach can reduce the number of low-level controller parameters required to provide fault tolerance compared to point-based estimation. The presented algorithm requires minimal additional hyper-parameter configuration, as it consists of a combination of existing algorithms, including particle filtering, clustering, and convex hulls, for which parameter tuning is well studied. We conducted a thorough empirical analysis on a quadcopter trajectory-tracking task subject to faults and disturbances. We showed that our proposed method can automatically learn a controller set whose size is smaller than that defined through traditional point-based tuning strategies, without incurring a significant performance loss. In future work, we plan to integrate a learning-based supervisory controller that can learn how to exploit the convex hull of the low-level controller parameter set for unknown operating conditions and extend the presented algorithm with automatic extremum-generation methods.

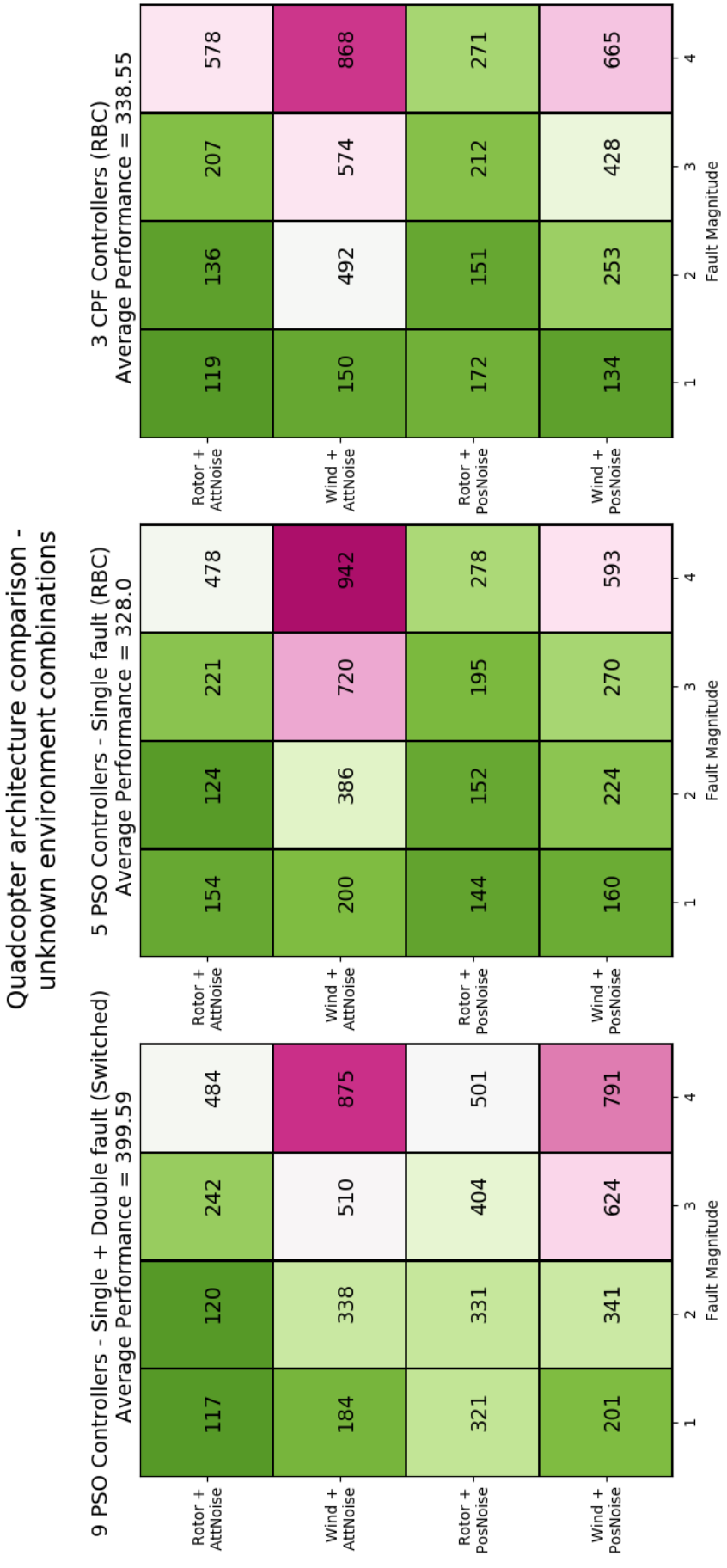


Figure I.7: Comparison average number of steps outside operational volume when using: 1) Switching using the extended controller set $\Theta_{\Lambda}^{PSO-ext}$ (9 controllers) , 2) Randomized Blended Control using Θ_{Λ}^{PSO} (5 controllers) and 3) Randomized Blended Control using Θ_{Λ}^{CPF} (3 controllers), in various combinations of conditions. Each cell represents the average over 100 runs and X and Y axis define the operating condition parameters.

8 Additional Experimental Details

Here we provide details on the parameters and configurations used for the experiments. We refer the interested reader to the github repository : https://github.com/YvesSohege/Journal-RAS-Simulation to find the required files to replicate the experiments. Parameter details are summerized in Table I.3.

8.1 Trajectory details

We run both PSO and CPF using our quadcopter simulator and overlay the resulting controller sets Θ^{PSO} with the regions Θ^{CPF} in Figure I.4. We use a randomized reference trajectory $\boldsymbol{\zeta}^R(t)$ over time points $t = 0, \cdots, T$, consisting of one fixed and two randomly sampled way-points in a 10m bounding box. The fixed way-point represents a simple takeoff maneuver and the two randomly selected way-points ensure low-level controllers are able to perform over a large variety of complex trajectories. The operational volume radius $\mathcal{R}$ around a given trajectory is set to 1 meter. The maximum number of steps is set to 3000 and a task is considered failed if the number of steps outside the operational volume exceeds 1000. Details on the quadcopters model parameters such as size and weight can be found in Table I.3. Video clips showing the optimization process running for PSO and CPF on each environment is available in the repository.

8.1.1 PSO Attitude Controllers

Particle-Swarm Optimization has been used to learn quadcopter PID parameters before [179]. Each particle θ_i consists of a tuple representing a possible PD controller configuration $\theta_t^i = (K_p, K_d)$, where K_p and K_d are proportional and derivative gain parameters respectively. The performance of a particle is set to the loss defined in Eq. I.21 when using a PD attitude controller defined by (K_p, K_d) on a random trajectory tracking task under the worst-case fault scenario, λ^{max}. The task is randomized to generalize the performance over all types of trajectories, resulting in a better overall controller configurations. A swarm of Z particles is initialized with randomized PD configurations and velocities. At each iteration, all particle configurations get evaluated on the task and each particle progresses towards the estimate of the optimal PD controller configuration. This results in the particles converging on a single optimal PD controller $\theta^*_{\lambda_i}$, where λ_i represents the environment that characterizes the task. We define the PD controller set obtained by using PSO as $\Theta^{PSO} = \{\theta^*_{\lambda_0}, \theta^*_{\lambda_1}, \theta^*_{\lambda_2}, \theta^*_{\lambda_3}, \theta^*_{\lambda_4}\}$, where the environment represented by λ_i can be found in Table I.1.

8.1.2 PSO Parameter Details

The swarm size Z is experimentally tuned to 16, we found larger swarms did not improve the quality of the result. The maximum number of iterations is set to 100 and experimentally the particles converged after around 40-60 iterations. This is related to the maximum allowed velocity which was set to 3 in our experiments and set according to the size of the search space. A lower value results in a slower search as particles need longer to traverse the space. The cognitive and social parameters are set to 2 and 1 respectively, which fall within the standard parameter range and are experimentally tuned according to [195]. We run PSO 10 times for each environment and take the average controller configurations found, which can be seen in Fig I.4. In total the PSO experiments encompass around 25,000 individual quadcopter simulations. The PSO process after 3 iterations and when converged can be seen for each environment in appendix figure I.8.

8.1.3 CPF Attitude Controllers

Each particle consists of a possible PD controller configuration $\theta^i = (K_p, K_d)$ but instead of a velocity we associate a weight w_t^i with each particle. Particle filtering aims to cover a PD parameter subspace with Z particles and randomly samples a K-sized subset of the particles according to their current weights. Particles are evaluated on a random trajectory tracking task and their weights updated based on the loss obtained according to Eq. I.21. Importance resampling ensures that bad performing particles get a progressively lower probability of being selected. One key difference to PSO is that the fault magnitude λ^t used for each task is also randomized in the interval $\lambda^0 \leq \lambda^t \leq \lambda^4$. Instead of converging on a single optimal solution, particle filtering can have many equally optimal particles - resulting in a subset of optimal PD controllers.

8.1.4 CPF Parameter Details

We set the bounds of the PD parameter space θ_min and θ_max as 0 and 20. We use 400 uniformly distributed particles with equal starting weights. We set the sample size to $K = 20$ which was determined experimentally to work best. Each CPF run takes 100 iterations totalling 2000 quadcopter simulations per environment. CPF is run 10 times on each environment for a total of 100,000 quadcopter simulations. After the filtering process completes, DBSCAN is used with the previously defined parameters on all particles whose average performance is below $\sigma = 500$. The resulting optimal PD controller regions, indicated by colour for each environment, can be seen in Fig I.4. The final clusters found and weights of each particle can be found for each environment

Table I.2: Parameters used for experiment 1.

Parameter	**Label**	**Value**
PSO Z	Swarm Size	16
PSO $maxIter$	Stop criteria	100
PSO $c1$	Cognitive Term	2
PSO $c2$	Social Term	1
PSO	Max velocity	3
CPF Z	Population Size	400
CPF $maxIter$	Stop criteria	100
CPF θ_{min}	Lower Bound	0
CPF θ_{max}	Upper Bound	20
CPF K	Re-sample size	20
CPF σ	threshold	500
CPF c_{size}	Min cluster size	4
CPF c_ϵ	Cluster max. distance	1.2
Quadcopter	Mass	1.2 kg
Quadcopter	Propeller diameter	10 cm
Quadcopter	Propeller pitch	4.5 deg
Quadcopter	Arm Length	30 cm
Quadcopter	Body	10 cm

in Figure I.9. We use DBSCAN [182], a well known and effective clustering algorithm that requires two parameters. The minimum number of samples to form a cluster c_{size} is typically set to $2 * D$, where D is the dimensionality of the samples; hence, for PD tuning we use $c_{size} = 4$. The maximum distance between samples to form a cluster, c_ϵ, is empirically determined to be $c_\epsilon = 1.2$.

9 Detailed Controller tuning results

9.1 Experiment 2: Additional Results

We compare the three architectures on each single fault operating condition over 500 random trajectories. The fault magnitudes are uniformly randomized over the entire fault space. The results can be seen in boxplot form in Fig. I.10. We also conducted a Friendman statistical comparison on the performance achieved by each architecture and found no significant difference in the distributions.

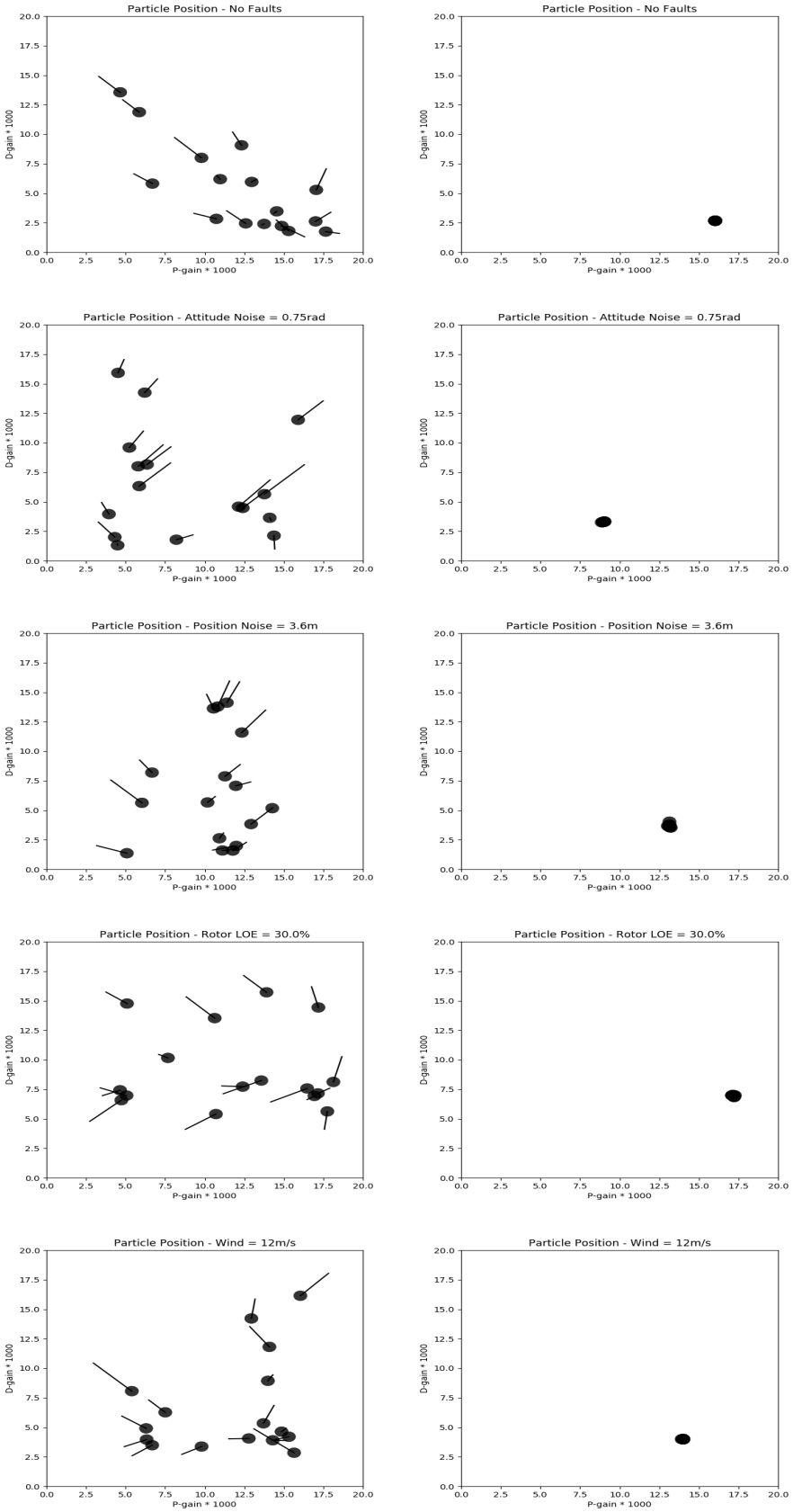


Figure I.8: Particle Swarm Optimization for quadcopter attitude PD controller gain parameters under various fault conditions. 1) Attitude Noise : 0.6rad | 2) Position Noise : 3.6m | 3) Rotor Loss of effectiveness : 20% | 4) Wind : 12m/s.

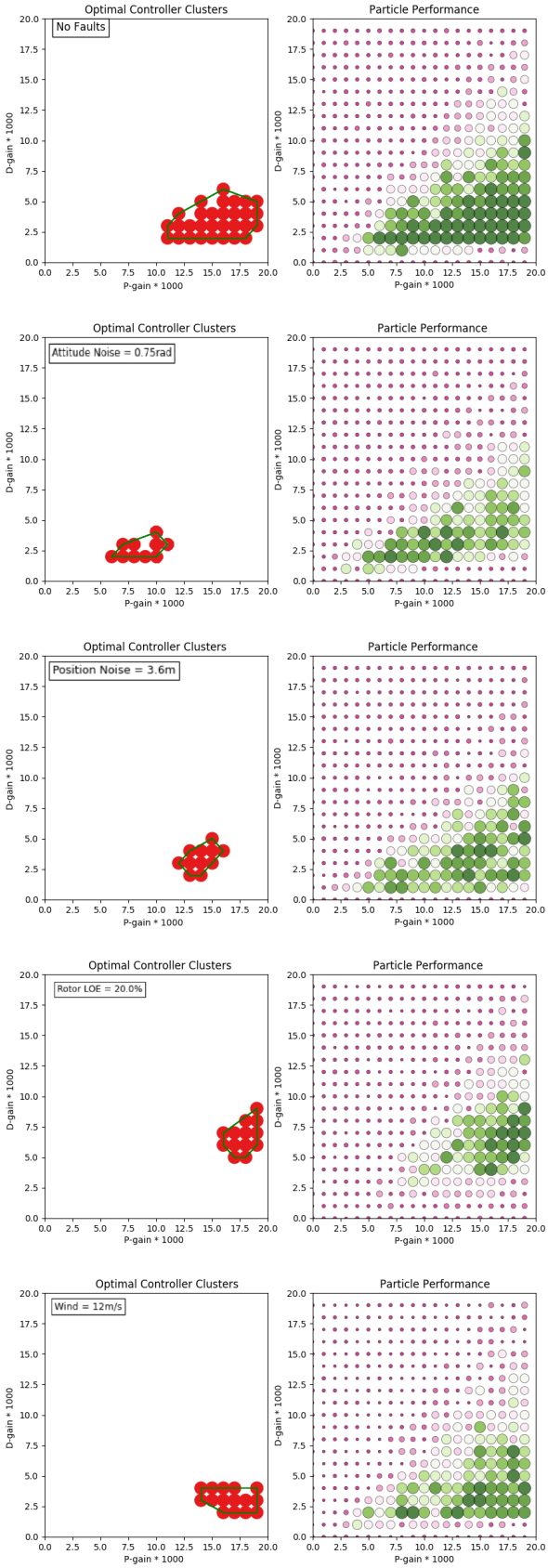


Figure I.9: Clustered Particle Filtering for quadcopter attitude PD controller gain parameters under various fault conditions. 1) Nominal Conditions | 2) Attitude Noise | 3) Position Noise | 4) Rotor Loss of effectiveness | 5) Wind

Table I.3: Quadcopter Roll and Pitch PD controller sets found using PSO and CPF approaches. Full set of environments can be found in Table I.1.

Controllers	**P-gain**	**D-gain**
PSO - Nominal ($\theta^*_{\lambda_0}$)	16000	3000
PSO - Attitude Noise ($\theta^*_{\lambda_1}$)	9000	3000
PSO - Position Noise ($\theta^*_{\lambda_2}$)	13000	3000
PSO - Wind ($\theta^*_{\lambda_3}$)	14000	4000
PSO - Rotor LOE ($\theta^*_{\lambda_4}$)	17000	7000
PSO - Attitude Noise + Wind ($\theta^*_{\lambda_1,\lambda_3}$)	14000	7000
PSO - Attitude Noise + Rotor LOE ($\theta^*_{\lambda_1,\lambda_4}$)	13000	6000
PSO - Position Noise + Wind ($\theta^*_{\lambda_2,\lambda_3}$)	11000	3000
PSO - Position Noise + Rotor LOE ($\theta^*_{\lambda_2,\lambda_4}$)	17000	5000
CPF - C1 (θ_1)	6000	2000
CPF - C2 (θ_2)	19000	2000
CPF - C3 (θ_3)	19000	9000

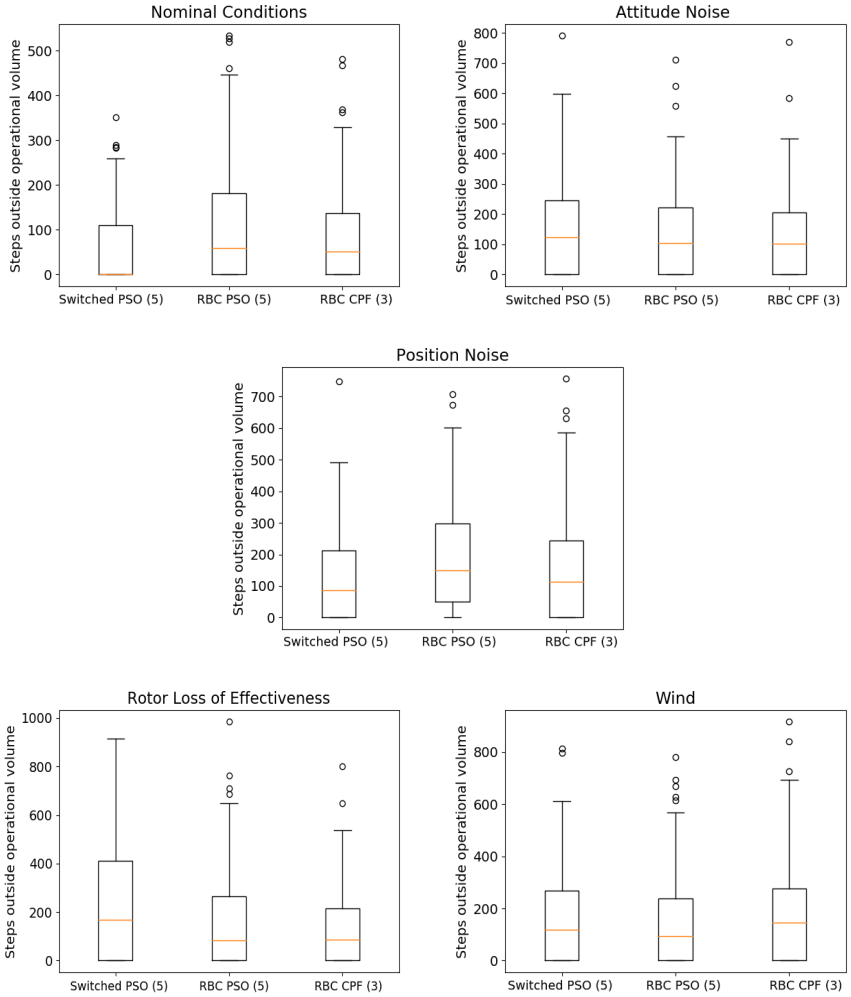


Figure I.10: Boxplots showing performance of each high-level architecture under random magnitude operating parameters over 500 simulations.

References

[1] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, pp. 1–18, 2021.

[2] K. Büyükkabasakal, B. Fi˙dan, and A. Savran, "Mixing adaptive fault tolerant control of quadrotor UAV," *Asian Journal of Control*, vol. 19, no. 4, pp. 1441–1454, 2017.

[3] F. Kong, Y. Guo, and W. Lyu, "Dynamics modeling and motion control of an new unmanned underwater vehicle," *IEEE Access*, vol. 8, pp. 30 119–30 126, 2020.

[4] J. García and D. Shafie, "Teaching a humanoid robot to walk faster through safe reinforcement learning," *Engineering Applications of Artificial Intelligence*, vol. 88, p. 103360, 2020.

[5] D. Wang, F. Fu, W. Li, Y. Tu, C. Liu, and W. Liu, "A review of the diagnosability of control systems with applications to spacecraft," *Annual Reviews in Control*, vol. 49, pp. 212 – 229, 2020.

[6] L. Jenny, C. Diaz, C. Ocampo-Martinez, and S. Olaru, "Dual mode control strategy for the energy efficiency of complex and flexible manufacturing systems," *Journal of Manufacturing Systems*, vol. 56, pp. 104 – 116, 2020.

[7] K. J. Arrow, D. Blackwell, and M. A. Girshick, "Bayes and minimax solutions of sequential decision problems," *Econometrica*, vol. 17, no. 3/4, pp. 213–244, 1949. [Online]. Available: http://www.jstor.org/stable/1905525

[8] S. Zhang and M. Sridharan, "A survey of knowledge-based sequential decision making under uncertainty," 2020, last visited 2021-08-01. [Online]. Available: https://arxiv.org/abs/2008.08548

[9] W. B. Powell, "From reinforcement learning to optimal control: A unified framework for sequential decisions," *Handbook of Reinforcement Learning and Control*, vol. 325, 2021.

[10] H. Sussmann, *Nonlinear controllability and optimal control*. CRC Press, Jan. 2017.

[11] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*. The MIT Press, 2018.

[12] R. P. Borase, D. K. Maghade, S. Y. Sondkar, and S. N. Pawar, "A review of PID control, tuning methods and applications," *International Journal of Dynamics and Control*, vol. 9, p. 818–827, 2020.

[13] A. Mesbah, "Stochastic model predictive control with active uncertainty learning: A survey on dual control," *Annual Reviews in Control*, vol. 45, pp. 107 – 117, 2018.

[14] D. Shevitz and B. Paden, "Lyapunov stability theory of nonsmooth systems," *IEEE Transactions on Automatic Control*, vol. 39, no. 9, pp. 1910–1914, 1994.

[15] Y. Rizk, M. Awad, and E. W. Tunstel, "Decision making in multiagent systems: A survey," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 10, no. 3, pp. 514–529, 2018.

[16] Z. Hu, K. Wan, X. Gao, Y. Zhai, and Q. Wang, "Deep reinforcement learning approach with multiple experience pools for uav's autonomous motion planning in complex unknown environments," *Sensors*, vol. 20, no. 7, p. 1890, 2020.

[17] D. Ernst, M. Glavic, F. Capitanescu, and L. Wehenkel, "Reinforcement learning versus model predictive control: A comparison on a power system problem," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 2, pp. 517–529, 2009.

[18] M. Kaspar, J. D. M. Osorio, and J. Bock, "Sim2real transfer for reinforcement learning without dynamics randomization," 2020, last accessed on 01-08-31. [Online]. Available: https://arxiv.org/abs/2002.11635

[19] A. A. Amin and K. M. Hasan, "A review of fault tolerant control systems: Advancements and applications," *Measurement*, vol. 143, pp. 58–68, 2019.

[20] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, "Meta-reinforcement learning of structured exploration strategies," *Proceedings of the*

32nd International Conference on Neural Information Processing Systems, p. 5307–5316, 2018.

[21] D. Wang, H. He, and D. Liu, "Adaptive critic nonlinear robust control: A survey," *IEEE Transactions on Cybernetics*, vol. 47, no. 10, pp. 3429–3451, 2017.

[22] N. T. Nguyen, "Model-reference adaptive control," in *Model-Reference Adaptive Control: A Primer*. Springer International Publishing, 2018, pp. 83–123. [Online]. Available: https://doi.org/10.1007/978-3-319-56393-0_5

[23] B. J. Gravell, P. M. Esfahani, and T. H. Summers, "Robust control design for linear systems via multiplicative noise," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 7392–7399, 2020, 21th IFAC World Congress.

[24] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in nonstationary environments: A survey," *IEEE Computational Intelligence Magazine*, vol. 10, no. 4, pp. 12–25, 2015.

[25] M. Blanke, M. Kinnaert, J. Lunze, M. Staroswiecki, and J. Schröder, *Diagnosis and fault-tolerant control*. Springer, 2006, vol. 3.

[26] Z.-S. Hou and Z. Wang, "From model-based control to data-driven control: Survey, classification and perspective," *Information Sciences*, vol. 235, pp. 3 – 35, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025512004781

[27] M. Kuipers and P. Ioannou, "Multiple model adaptive control with mixing," *IEEE Transactions on Automatic Control*, vol. 55, no. 8, pp. 1822–1836, 2010.

[28] C. Patel and I. Kroo, "Control law design for improving uav performance using wind turbulence," in *44th AIAA Aerospace Sciences Meeting and Exhibit*, 2006, p. 231.

[29] S. Aradi, "Survey of deep reinforcement learning for motion planning of autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–20, 2020.

[30] N. Casas, "Deep deterministic policy gradient for urban traffic light control," 2017, last accessed 01-08-21. [Online]. Available: https://arxiv.org/abs/1703.09035

[31] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on*

Machine Learning, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37, 2015, pp. 1889–1897.

[32] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, last accessed 01-08-21. [Online]. Available: https://arxiv.org/abs/1707.06347

[33] L. Buşoniu, T. d. Bruin, D. Tolić, J. Kober, and I. Palunko, "Reinforcement learning for control: Performance, stability, and deep approximators," *Annual Reviews in Control*, vol. 46, pp. 8 – 28, 2018.

[34] M. Athans, D. Castanon, K. Dunn, C. Greene, Wing Lee, N. Sandell, and A. Willsky, "The stochastic control of the f-8c aircraft using a multiple model adaptive control (mmac) method–part i: Equilibrium flight," *IEEE Transactions on Automatic Control*, vol. 22, no. 5, pp. 768–780, 1977.

[35] J. Xie, S. Li, H. Yan, and D. Yang, "Model reference adaptive control for switched linear systems using switched multiple models control strategy," *Journal of the Franklin Institute*, vol. 356, no. 5, pp. 2645 – 2667, 2019.

[36] Z. Liu, D. Theilliol, L. Yang, Y. He, and J. Han, "Transition control of tilt rotor unmanned aerial vehicle based on multi-model adaptive method," in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2017, pp. 560–566.

[37] W. Zhang, Q. Li, Y. Zhang, Z. Lu, and C. Nian, "Weighted multiple model adaptive boundary control for a flexible manipulator," *Science Progress*, vol. 103, pp. 1–20, 2019.

[38] P. Goel, G. Dedeoglu, S. I. Roumeliotis, and G. S. Sukhatme, "Fault detection and identification in a mobile robot using multiple model estimation and neural network," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 3, 2000, pp. 2302–2309 vol.3.

[39] K. Rajagopal and S. Balakrishnan, "Intelligent switching between multiple model-based adaptive controllers using data-driven control theory," in *2016 American Control Conference (ACC)*. IEEE, 2016, pp. 2506–2511.

[40] S. C. Hoi, D. Sahoo, J. Lu, and P. Zhao, "Online learning: A comprehensive survey," *Neurocomputing*, vol. 459, pp. 249–289, 2021.

[41] A. Garlapati, A. Raghunathan, V. Nagarajan, and B. Ravindran, "A reinforcement learning approach to online learning of decision trees," 2015, last accessed 01-08-21. [Online]. Available: https://arxiv.org/abs/1507.06923

[42] P. M. Djuric, J. H. Kotecha, Jianqui Zhang, Yufei Huang, T. Ghirmai, M. F. Bugallo, and J. Miguez, "Particle filtering," *IEEE Signal Processing Magazine*, vol. 20, no. 5, pp. 19–38, 2003.

[43] A. Abbaspour, S. Mokhtari, A. Sargolzaei, and K. K. Yen, "A survey on active fault-tolerant control systems," *Electronics*, vol. 9, no. 9, 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/9/1513

[44] P. Goupil, "AIRBUS state of the art and practices on FDI and FTC in flight control system," *Control Engineering Practice*, vol. 19, pp. 524–539, 2011.

[45] S. X. D. Shen Yin, Bing Xiao and D. Zhou, ""A review on recent development of spacecraft attitude fault tolerant control system"," *Transactions on Industrial Electronis*, vol. 63, no. 5, pp. 3311–3320, 2016.

[46] J. Jiang and X. Yu, ""Fault-tolerant control systems: A comparative study between active and passive approaches"," *Annual Reviews in Control*, vol. 36, pp. 60–72, 2012.

[47] J. Lunze, "From fault diagnosis to reconfigurable control: A unified concept," in *Control and Fault-Tolerant Systems (SysTol), 2016 3rd Conference on*. IEEE, 2016, pp. 413–421.

[48] C. d. V. D. Molenkamp, E. van Kampen and Q. Chu, ""Intelligent controller selection for aggressive quadrotor manoeuvring"," *IAA Information Systems-AIAA Infotech @ Aerospace, AIAA SciTech Forum*, 2017.

[49] R. J. Patton, "Fault-tolerant control," *Encyclopedia of systems and control*, pp. 422–428, 2015.

[50] X. Yu and J. Jiang, "A survey of fault-tolerant controllers based on safety-related issues," *Annual Reviews in Control*, vol. 39, pp. 46–57, 2015.

[51] M. Kuipers and P. Ioannou, "Multiple model adaptive control with mixing," *IEEE Transactions on Automatic Control*, vol. 55, no. 8, pp. 1822–1836, 2010.

[52] Y. Zhang and J. Jiang, ""Integrated active fault-tolerant control using IMM approach"," *Transactions on Aerospace and Electronic Systems*, vol. 37, no. 4, pp. 1221–1235, 2001.

[53] D. Magill, "Optimal adaptive estimation of sampled stochastic processes," *IEEE Transactions on Automatic Control*, vol. 10, no. 4, pp. 434–439, 1965.

[54] M. Athans, K.-P. Dunn, C. Greene, W. Lee, N. Sandell, I. Segall, and A. Willsky, "The stochastic control of the f-8c aircraft using the multiple model adaptive control (mmac) method," in *Decision and Control including the 14th Symposium on Adaptive Processes, 1975 IEEE Conference on*, vol. 14. IEEE, 1975, pp. 217–228.

[55] C. Yu, R. J. Roy, H. Kaufman, and B. W. Bequette, "Multiple-model adaptive predictive control of mean arterial pressure and cardiac output," *IEEE Transactions on Biomedical Engineering*, vol. 39, no. 8, pp. 765–778, 1992.

[56] D. W. Lane and P. S. Maybeck, "Multiple model adaptive estimation applied to the lambda urv for failure detection and identification," in *Decision and Control, 1994., Proceedings of the 33rd IEEE Conference on*, vol. 1. IEEE, 1994, pp. 678–683.

[57] S. Fekri, M. Athans, and A. Pascoal, "Robust multiple model adaptive control (rmmac): A case study," *International Journal of Adaptive Control and Signal Processing*, vol. 21, no. 1, pp. 1–30, 2007.

[58] V. Hassani, J. P. Hespanha, M. Athans, and A. M. Pascoal, "Stability analysis of robust multiple model adaptive control," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 350–355, 2011.

[59] M. Kuure-Kinsey and B. W. Bequette, "Multiple model predictive control of nonlinear systems," in *Nonlinear Model Predictive Control*. Springer, 2009, pp. 153–165.

[60] J. Sun, "Small-signal methods for AC distributed power systems–a review," *Power Electronics, IEEE Transactions on*, vol. 24, no. 11, pp. 2545–2554, 2009.

[61] J. H. Taylor and A. J. Antoniotti, "Linearization algorithms for computer-aided control engineering," *Control Systems, IEEE*, vol. 13, no. 2, pp. 58–64, 1993.

[62] K. S. Narendra and Z. Han, "The changing face of adaptive control: the use of multiple models," *Annual Reviews in Control*, vol. 35, no. 1, pp. 1–12, 2011.

[63] P. D. Hanlon and P. S. Maybeck, "Multiple-model adaptive estimation using a residual correlation Kalman filter bank," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 36, no. 2, pp. 393–406, 2000.

[64] W. Zhang, "Stable weighted multiple model adaptive control: discrete-time stochastic plant," *International Journal of Adaptive Control and Signal Processing*, vol. 27, no. 7, pp. 562–581, 2013.

[65] P. A. Ioannou, "Cdc semi-plenary:"robust adaptive control: The search for the holy grail"," in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*. IEEE, 2008, pp. 12–13.

[66] L. Magni, R. Scattolini, and M. Tanelli, "Switched model predictive control for performance enhancement," *International Journal of Control*, vol. 81, no. 12, pp. 1859–1869, 2008.

[67] D. Q. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014.

[68] H. Alwi, C. Edwards, and C. Tan, *Fault Tolerant Control and Fault Detection and Isolation*. Springer, 2011.

[69] Z. Gao, C. Cecati, and S. X. Ding, "A survey of fault diagnosis and fault-tolerant techniques—part i: Fault diagnosis with model-based and signal-based approaches," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 6, pp. 3757–3767, 2015.

[70] P. Bolzern, P. Colaneri, and G. D. Nicolao, "Design of stabilizing strategies for discrete-time dual switching linear systems," *Automatica*, vol. 69, pp. 93–100, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0005109816300644

[71] M. Fiacchini, M. Jungers, and A. Girard, "Stabilization and control lyapunov functions for language constrained discrete-time switched linear systems," *Automatica*, vol. 93, pp. 64–74, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0005109818301262

[72] M. Witczak, *Fault Diagnosis and Fault-Tolerant Control Strategies for Non-Linear Systems*. Springer, 2014.

[73] H. Noura, D. Theilliol, and Jean-Christophe, *Fault-tolerant Control Systems: Design and Practical Applications*. Springer, 2009.

[74] L. Etienne, A. Girard, and L. Greco, "Stability and stabilizability of discrete-time dual switching systems with application to sampled-data systems," *Automatica*, vol. 100, pp. 388–395, 2019.

[75] K. Lee and R. Bhattacharya, "Stability analysis of large-scale distributed networked control systems with random communication delays: A switched system approach," *Systems & Control Letters*, vol. 85, pp. 77–83, 2015.

[76] H. Lin and P. J. Antsaklis, "Stability and stabilizability of switched linear systems: A survey of recent results," *IEEE Transactions on Automatic Control*, vol. 54, no. 2, pp. 308–322, 2009.

[77] M. Philippe, R. Essick, G. E. Dullerud, and R. M. Jungers, "Stability of discrete-time switching systems with constrained switching sequences," *Automatica*, vol. 72, pp. 242–250, 2016.

[78] O. Costa, M. Fragoso, and R. Marques, *Discrete-Time Markov Jump Linear Systems*. Springer, 2006.

[79] S. Baldi, P. A. Ioannou, and E. B. Kosmatopoulos, "Adaptive mixing control with multiple estimators," *Journal of Adaptive Control and Signal Processing*, vol. 26, no. 8, pp. 800–820, 2012.

[80] R. S. Sutton, A. G. Barto, and R. J. Williams, "Reinforcement learning is direct adaptive optimal control," *IEEE Control Systems*, vol. 12, no. 2, pp. 19–22, 1992.

[81] A. P. Schoellig, F. L. Mueller, and R. D'Andrea, "Optimization-based iterative learning for precise quadrocopter trajectory tracking," *Autonomous Robots*, vol. 33, no. 1-2, pp. 103–127, 2012.

[82] F. L. Mueller, A. P. Schoellig, and R. D'Andrea, "Iterative learning of feed-forward corrections for high-performance tracking," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 3276–3281.

[83] M. Hehn and R. D'Andrea, "A frequency domain iterative feed-forward learning scheme for high performance periodic quadrocopter maneuvers," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 2445–2451.

[84] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[85] M. C. Koval, C. R. Mansley, and M. L. Littman, “"Autonomous quadrotor control with reinforcement learning",” last accessed 06-2018. [Online]. Available: https://www.mkoval.org/projects/quadrotor/files/quadrotor-rl.pdf

[86] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[87] D. Hartman, K. Landis, M. Mehrer, S. Moreno, and J. Kim, *Quadcopter Simulation*. [Online]. Available: https://github.com/dch33/Quad-Sim

[88] R. S. Sutton, “Temporal credit assignment in reinforcement learning,” Ph.D. dissertation, University of Massachusetts Amherst, 1984.

[89] M. Blanke, M. Kinnaert, J. Lunze, and M. Staroswiecki, *Diagnosis and Fault-Tolerant Control*. Springer, 2016.

[90] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” 2015, last accessed 01-08-21. [Online]. Available: https://arxiv.org/abs/1509.02971

[91] N. S. Özbek, M. Önkol, and M. Ö. Efe, “Feedback control strategies for quadrotor-type aerial robots: a survey,” *Transactions of the Institute of Measurement and Control*, vol. 38, no. 5, pp. 529–554, 2016.

[92] M. W. Mueller and R. D’Andrea, “Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 45–52.

[93] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, “Control of a quadrotor with reinforcement learning,” *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[94] C. Greatwood and A. G. Richards, “Reinforcement learning and model predictive control for robust embedded quadrotor guidance and control,” *Autonomous Robots*, vol. 43, p. 1681–1693, 2019.

[95] W. Koch, R. Mancuso, R. West, and A. Bestavros, “Reinforcement learning for UAV attitude control,” *ACM Transactions on Cyber-Physical Systems*, vol. 3, no. 2, p. 22, 2019.

[96] F. Fei, Z. Tu, Y. Yang, X. Zhang+, D. Xu+, and X. Deng, “Learn to Recover: Reinforcement Learning-Assisted Fault Tolerant Control for

Quadrotor UAVs," 2019, last accessed 01-08-21. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9196611

[97] J. Lunze, "From fault diagnosis to reconfigurable control: A unified concept," in *2016 3rd Conference on Control and Fault-Tolerant Systems (SysTol)*. IEEE, 2016, pp. 413–421.

[98] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[99] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[100] H. Mo and G. Farid, "Nonlinear and adaptive intelligent control techniques for quadrotor UAV – a survey," *Asian Journal of Control*, vol. 21, no. 2, pp. 989–1008, 2019.

[101] L. Hou, A. N. Michel, and H. Ye, "Stability analysis of switched systems," in *Decision and Control, 1996., Proceedings of the 35th IEEE Conference on*, vol. 2, 1996, pp. 1208–1212.

[102] C. Edwards and S. Spurgeon, *Sliding mode control: theory and applications*. CRC Press, 1998.

[103] G. Provan and Y. Sohége, "Fault-tolerant control for unseen faults using randomized methods," in *2019 4th Conference on Control and Fault Tolerant Systems (SysTol)*. IEEE, 2019, pp. 159–164.

[104] Q. Shi, H.-K. Lam, B. Xiao, and S.-H. Tsai, "Adaptive PID controller based on Q-learning algorithm," *CAAI Transactions on Intelligence Technology*, vol. 3, no. 4, pp. 235–244, 2018.

[105] W. Zhang, "Further results on stable weighted multiple model adaptive control: Discrete-time stochastic plant," *International Journal of Adaptive Control and Signal Processing*, vol. 29, no. 12, pp. 1497–1514, 2015.

[106] S. Kersting and M. Buss, "How to systematically distribute candidate models and robust controllers in multiple-model adaptive control: A coverage control approach," *IEEE Transactions on Automatic Control*, vol. 63, no. 4, pp. 1075–1089, 2018.

[107] Z. Han and K. S. Narendra, "New concepts in adaptive control using multiple models," *IEEE Transactions on Automatic Control*, vol. 57, no. 1, pp. 78–89, 2011.

[108] K. S. Narendra and K. Esfandiari, "Adaptive identification and control of linear periodic systems using second-level adaptation," *International Journal of Adaptive Control and Signal Processing*, vol. 33, no. 6, pp. 956–971, 2019.

[109] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, no. 19, pp. 70–76, 2017.

[110] T. R. Besold, A. d'Avila Garcez, S. Bader, H. Bowman, P. Domingos, P. Hitzler, K.-U. Kuehnberger, and L. C. Lamb, "Neural-symbolic learning and reasoning: A survey and interpretation," 2017, last accessed 01-08-21. [Online]. Available: https://arxiv.org/abs/1711.03902

[111] R. Rai and C. K. Sahu, "Driven by data or derived through physics? a review of hybrid physics guided machine learning techniques with cyber-physical system (cps) focus," *IEEE Access*, vol. 8, pp. 71 050–71 073, 2020.

[112] J. Wang, J. Zhuang, L. Duan, and W. Cheng, "A multi-scale convolution neural network for featureless fault diagnosis," in *2016 International Symposium on Flexible Automation (ISFA)*. IEEE, 2016, pp. 65–70.

[113] Y. Luo, Y. Xiao, L. Cheng, G. Peng, and D. D. Yao, "Deep learning-based anomaly detection in cyber-physical systems: Progress and opportunities," *ACM Computing Surveys*, vol. 54, no. 5, May 2021.

[114] N. F. Waziralilah, A. Abu, M. Lim, L. K. Quen, and A. Elfakharany, "A review on convolutional neural network in bearing fault diagnosis," in *MATEC Web of Conferences*, vol. 255. EDP Sciences, 2019, p. 06002.

[115] H. P. Nautrup, T. Metger, R. Iten, S. Jerbi, L. M. Trenkwalder, H. Wilming, H. J. Briegel, and R. Renner, "Operationally meaningful representations of physical systems in neural networks," 2020, last accessed on 01-08-21. [Online]. Available: https://arxiv.org/abs/2001.00593

[116] R. Iten, T. Metger, H. Wilming, L. del Rio, and R. Renner, "Discovering physical concepts with neural networks," *Physical Review Letters*, vol. 124, no. 1, Jan 2020. [Online]. Available: http://dx.doi.org/10.1103/PhysRevLett.124.010508

[117] A. Bunte, B. Stein, and O. Niggemann, "Model-based diagnosis for cyber-physical production systems based on machine learning and residual-based diagnosis models," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 2727–2735.

[118] Y. Luo, Y. Xiao, L. Cheng, G. Peng, and D. D. Yao, "Deep learning-based anomaly detection in cyber-physical systems: Progress and opportunities," *ACM Computing Surveys*, vol. 54, no. 5, May 2021.

[119] M. A. Chao, C. Kulkarni, K. Goebel, and O. Fink, "Fusing physics-based and deep learning models for prognostics," last accessed on 01-08-21. [Online]. Available: https://arxiv.org/abs/2003.00732

[120] D. Xia, L. Cheng, and Y. Yao, "A robust inner and outer loop control method for trajectory tracking of a quadrotor," *Sensors*, vol. 17, no. 9, p. 2147, 2017.

[121] Y. Sohège, G. Provan, M. Quiñones-Grueiro, and G. Biswas, "Deep reinforcement learning and randomized blending for control under novel disturbances," vol. 53, no. 2, 2020, pp. 8175–8180, 21th IFAC World Congress. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896320329803

[122] A. Majumdar, *Python Quadcopter Simulation*, 2018. [Online]. Available: https://github.com/abhijitmajumdar/Quadcopter_simulator

[123] P. Becker-Ehmck, M. Karl, J. Peters, and P. van der Smagt, "Learning to fly via deep model-based reinforcement learning," 2020, last accessed 01-08-21. [Online]. Available: https://arxiv.org/abs/2003.08876

[124] A. Hill, A. Raffin, M. Ernestus, and Gleave, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[125] S. Y. Choi and D. Cha, "Unmanned aerial vehicles using machine learning for autonomous flight; state-of-the-art," *Advanced Robotics*, vol. 33, no. 6, pp. 265–277, 2019.

[126] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. Pister, "Low-level control of a quadrotor with deep model-based reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4224–4230, 2019.

[127] W. Koch, R. Mancuso, R. West, and A. Bestavros, "Reinforcement learning for uav attitude control," *ACM Transactions on Cyber-Physical Systems*, vol. 3, no. 2, pp. 1–21, 2019.

[128] L. Bjarre, "Robust reinforcement learning for quadcopter control," Master's thesis, kth Royal Institute of Technology, School of Electrical Engineering and Computer Science, 2019.

[129] F. Fei, Z. Tu, D. Xu, and X. Deng, "Learn-to-recover: Retrofitting uavs with reinforcement learning-assisted flight control under cyber-physical attacks," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 7358–7364.

[130] S. S. Miladi N., Ladhari T., "Enmpc versus pid control strategies applied to a quadcopter," in *New Trends in Robot Control. Studies in Systems, Decision and Control*. Singapore: Springer, 2020, vol. 270, ch. 10, pp. 319–334.

[131] M. W. Mueller and R. D'Andrea, "Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 45–52.

[132] A. Kaba, A. Ermeydan, and E. Kiyak, "Model derivation, attitude control and kalman filter estimation of a quadcopter," in *2017 4th International Conference on Electrical and Electronic Engineering (ICEEE)*, 2017, pp. 210–214.

[133] R. C. Leishman, J. C. Macdonald, R. W. Beard, and T. W. McLain, "Quadrotors and accelerometers: State estimation with an improved dynamic model," *IEEE Control Systems Magazine*, vol. 34, no. 1, pp. 28–41, 2014.

[134] G. Cano Lopes, M. Ferreira, A. da Silva Simões, and E. Luna Colombini, "Intelligent control of a quadrotor with proximal policy optimization reinforcement learning," in *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*, 2018, pp. 503–508.

[135] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8973–8979.

[136] K. Rosser, J. Kok, J. Chahl, and J. Bongard, "Sim2real gap is non-monotonic with robot complexity for morphology-in-the-loop flapping wing design," in

2020 IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 7001–7007.

[137] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 3803–3810.

[138] M. Efe, "Neural network assisted computationally simple PID control of a quadrotor UAV," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 2, pp. 354–361, 2011.

[139] W. Wiesemann, D. Kuhn, and B. Rustem, "Robust markov decision processes," *Mathematics of Operations Research*, vol. 38, no. 1, pp. 153–183, 2013.

[140] D. J. Mankowitz, N. Levine, R. Jeong, Y. Shi, J. Kay, A. Abdolmaleki, J. T. Springenberg, T. Mann, T. Hester, and M. Riedmiller, "Robust reinforcement learning for continuous control with model misspecification," *International Conference on Learning Representations*, 2020.

[141] M. A. Abdullah, H. Ren, H. Ou-ammar, V. Milenkovi, R. Luo, M. Zhang, and J. Wang, "Wasserstein Robust Reinforcement Learning," 2019, last accessed 01-08-21. [Online]. Available: https://arxiv.org/abs/1907.13196

[142] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization," in *IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 1–8.

[143] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield, "Training deep networks with synthetic data: Bridging the reality gap by domain randomization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 969–977.

[144] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and P. Montr, "Active Domain Randomization," in *Conference on Robot Learning*, 2020, pp. 1162–1176.

[145] S. Wang, N. Anselmo, M. Garrett, R. Remias, M. Trivett, A. Christoffersen, and N. Bezzo, "Fly-crash-recover: A sensor-based reactive framework for online collision recovery of uavs," in *2020 Systems and Information Engineering Design Symposium (SIEDS)*, 2020, pp. 1–6.

[146] W. Zhang and Q. Li, "Weighted multiple model adaptive control of time-varying systems," *Journal of Robotics, Networking and Artificial Life*, vol. 1, no. 4, pp. 291–294, 2015.

[147] W. Zhang and L. Zhao, "Survey and tutorial on multiple model methodologies in modelling, identification and control," *International Journal of Modelling, Identification and Control*, vol. 32, no. 1, pp. 1–9, 2019.

[148] A. Majumdar, "Python quadcopter simulator," https://github.com/abhijitmajumdar/Quadcopter_simulator, 2018.

[149] L. Chan, F. Naghdy, and D. Stirling, "Application of adaptive controllers in teleoperation systems: A survey," *IEEE Transactions on Human-Machine Systems*, vol. 44, no. 3, pp. 337–352, 2014.

[150] A.-T. Nguyen, C. Sentouh, and J.-C. Popieul, "Driver-automation cooperative approach for shared steering control under multiple system constraints: Design and experiments," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 5, pp. 3819–3830, 2017.

[151] X. Na and D. J. Cole, "Application of open-loop stackelberg equilibrium to modeling a driver's interaction with vehicle active steering control in obstacle avoidance," *IEEE Transactions on Human-Machine Systems*, 2017.

[152] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and multidisciplinary optimization*, vol. 26, no. 6, pp. 369–395, 2004.

[153] S. Gray, R. Chevalier, B. Caimano, and J. Scatena, "Graduated automation for humanoid manipulation," in *Automation Science and Engineering (CASE), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1366–1373.

[154] L. Song, H. Guo, F. Wang, J. Liu, and H. Chen, "Model predictive control oriented shared steering control for intelligent vehicles," in *Control And Decision Conference (CCDC), 2017 29th Chinese*. IEEE, 2017, pp. 7568–7573.

[155] F. Altché, X. Qian, and A. de La Fortelle, "An algorithm for supervised driving of cooperative semi-autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 12, pp. 3527–3539, 2017.

[156] H. Kim, J. Cho, D. Kim, and K. Huh, "Intervention minimized semi-autonomous control using decoupled model predictive control," in *Intelligent Vehicles Symposium (IV), 2017 IEEE*. IEEE, 2017, pp. 618–623.

[157] D. Richards and A. Stedmon, "To delegate or not to delegate: A review of control frameworks for autonomous cars," *Applied ergonomics*, vol. 53, pp. 383–388, 2016.

[158] A. Gambier, "Mpc and pid control based on multi-objective optimization," in *American Control Conference, 2008*. IEEE, 2008, pp. 4727–4732.

[159] A. D. Dragan, "Robot planning with mathematical models of human state and action," 2017, last accessed 01-08-21. [Online]. Available: https://arxiv.org/abs/1705.04226

[160] M. Flad, L. Fröhlich, and S. Hohmann, "Cooperative shared control driver assistance systems based on motion primitives and differential games," *IEEE Transactions on Human-Machine Systems*, 2017.

[161] S. Nikolaidis, J. Forlizzi, D. Hsu, J. Shah, and S. Srinivasa, "Mathematical models of adaptation in human-robot collaboration," 2017, last accessed 01-08-21. [Online]. Available: https://arxiv.org/abs/1707.02586

[162] S. Nikolaidis, D. Hsu, and S. Srinivasa, "Human-robot mutual adaptation in collaborative tasks: Models and experiments," *The International Journal of Robotics Research*, vol. 36, pp. 618–634, 2017.

[163] R. Rajamani, *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[164] J. Jiang and X. Yu, "Fault-tolerant control systems: A comparative study between active and passive approaches," *Annual Reviews in Control*, vol. 36, no. 1, pp. 60 – 72, 2012.

[165] G. Vinnicombe, "Frequency domain uncertainty and the graph topology," *IEEE Transactions on Automatic Control*, vol. 38, no. 9, pp. 1371–1383, 1993.

[166] V. Pandey, I. Kar, and C. Mahanta, "Multiple model adaptive control using second level adaptation for a class of nonlinear systems with linear parameterizations," *International Journal of Dynamics and Control*, vol. 6, no. 3, pp. 1319–1334, 2018.

[167] B. Gao, S. Gao, Y. Zhong, G. Hu, and C. Gu, "Interacting multiple model estimation-based adaptive robust unscented kalman filter," *International*

Journal of Control, Automation and Systems, vol. 15, no. 5, pp. 2013–2025, 2017.

[168] D. N. Cardoso, S. Esteban, and G. V. Raffo, "A new robust adaptive mixing control for trajectory tracking with improved forward flight of a tilt-rotor uav," *ISA Transactions*, vol. 110, pp. 86–104, 2021.

[169] B. D. O. Anderson, T. S. Brinsmead, F. De Bruyne, J. Hespanha, D. Liberzon, and A. S. Morse, "Multiple model adaptive control. part 1: Finite controller coverings," *International Journal of Robust and Nonlinear Control*, vol. 10, no. 11, pp. 909–929, 2000-09.

[170] N. J. Killingsworth and M. Krstic, "Pid tuning using extremum seeking: online, model-free performance optimization," *IEEE control systems magazine*, vol. 26, no. 1, pp. 70–79, 2006.

[171] K. J. Åström, T. Hägglund, C. C. Hang, and W. K. Ho, "Automatic tuning and adaptation for pid controllers-a survey," *Control Engineering Practice*, vol. 1, no. 4, pp. 699–714, 1993.

[172] I. J. Gyöngy and D. W. Clarke, "On the automatic tuning and adaptation of pid controllers," *Control engineering practice*, vol. 14, no. 2, pp. 149–163, 2006.

[173] V. D. Blondel and J. N. Tsitsiklis, "A survey of computational complexity results in systems and control," *Automatica*, vol. 36, no. 9, pp. 1249–1274, 2000.

[174] Y. Sohège, M. Quinones-Grueiro, and G. Provan, "Unknown fault tolerant control using deep reinforcement learning: A blended control approach." DX'19, 2019.

[175] Y. Sohège, G. Provan, and M. Quiñones-Grueiro, "Neural-symbolic fault tolerant control for quadcopter trajectory-following tasks." DX, 2020. [Online]. Available: http://www.dx-2020.org/papers/DX-2020_paper_22.pdf

[176] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Transactions on Mathematical Software*, vol. 22, no. 4, pp. 469–483.

[177] E. Elbeltagi, T. Hegazy, and D. Grierson, "Comparison among five evolutionary-based optimization algorithms," *Advanced Engineering Informatics*, vol. 19, no. 1, pp. 43–53, 2005.

[178] W. Low, R. Nagarajan, and S. Yaacob, "Visual based slam using modified PSO," *6th International Colloquium on Signal Processing its Applications*, vol. 1, pp. 1–5, 2010.

[179] B. Verma and P. K. Padhy, "Robust fine tuning of optimal pid controller with guaranteed robustness," *IEEE Transactions on Industrial Electronics*, vol. 67, no. 6, pp. 4911–4920, 2020.

[180] A. Azar and H. E. H. A. A.S Sayed, A.S. Shahin, "Pid controller for 2-dofs twin rotor mimo system tuned with particle swarm optimization," in *Proceedings of the International Conference on Advanced Intelligent Systems and Informatics 2019*, vol. 1058. Springer, 2020.

[181] J. Bansal, P. Singh, and N. Pal, *Evolutionary and Swarm Intelligence Algorithms*. Springer, 2019, vol. 779.

[182] E. Schubert, J. Sander, M. Ester, H.-P. Kriegel, L.-M.-U. München, and X. Xu, "DBSCAN revisited: Why and how you should (still) use DBSCAN," *ACM Transactions on Database Systems*, vol. 42, no. 3, p. 21, 2017.

[183] S. Hayat, E. Yanmaz, C. Bettstetter, and T. X. Brown, "Multi-objective drone path planning for search and rescue with quality-of-service requirements," *Autonomous Robots*, vol. 44, no. 7, pp. 1183–1198, 2020.

[184] M. Boulares and A. Barnawi, "A novel UAV path planning algorithm to search for floating objects on the ocean surface based on object's trajectory prediction by regression," *Robotics and Autonomous Systems*, vol. 135, 2021.

[185] S. Li, M. M. Ozo, C. De Wagter, and G. C. de Croon, "Autonomous drone race: A computationally efficient vision-based navigation and control strategy," *Robotics and Autonomous Systems*, vol. 133, 2020.

[186] U. R. Mogili and B. B. V. L. Deepak, "Review on application of drone systems in precision agriculture," *Procedia Computer Science*, vol. 133, pp. 502–509, 2018.

[187] C. Torresan, A. Berton, F. Carotenuto, S. F. D. Gennaro, B. Gioli, A. Matese, F. Miglietta, C. Vagnoli, A. Zaldei, and L. Wallace, "Forestry applications of uavs in europe: a review," *International Journal of Remote Sensing*, vol. 38, no. 8-10, pp. 2427–2447, 2017.

[188] F. Santoso, M. A. Garratt, S. G. Anavatti, and I. Petersen, "Robust hybrid nonlinear control systems for the dynamics of a quadcopter drone," *IEEE*

Transactions on Systems, Man, and Cybernetics: Systems, vol. 50, no. 8, pp. 3059–3071, 2020.

[189] N. Osmic, M. Kuric, and I. Petrovic, "Detailed octorotor modeling and PD control," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, pp. 002 182–002 189.

[190] H. Hamadi, B. Lussier, I. Fantoni, C. Francis, and H. Shraim, "Comparative study of self tuning, adaptive and multiplexing FTC strategies for successive failures in an octorotor UAV," *Robotics and Autonomous Systems*, vol. 133, 2020.

[191] M. Quinones-Grueiro, T. Darrah, G. Biswas, and C. Kulkarni, "A decision-making framework for safe operations of unmanned aerial vehicles in urban environments," *Annual Conference of the Prognostics and Health Management Society*, p. 11, 2020.

[192] C. Hsieh, H. Sibai, H. Taylor, and S. Mitra, "Unmanned air-traffic management (UTM): Formalization, a prototype implementation, verification, and performance evaluation," *Federal Aviation Administration*, 2020. [Online]. Available: http://arxiv.org/abs/2009.04655

[193] M. F. Shehzad, A. Bilal, and H. Ahmad, "Position and attitude control of an aerial robot (quadrotor) with intelligent pid and state feedback lqr controller: A comparative approach," pp. 340–346, 2019.

[194] P. Wang, Z. Man, Z. Cao, J. Zheng, and Y. Zhao, "Dynamics modelling and linear control of quadcopter," in *2016 International Conference on Advanced Mechatronic Systems (ICAMechS)*, 2016, pp. 498–503.

[195] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948 vol.4.