Model-Predictive Fuzzy Control for Search-and-Rescue Path-Planning of Multi-Agent Systems

Craig Maxwell

Delft University of Technology

TU Delft

Model-Predictive Fuzzy Control for Search-and-Rescue Path-Planning of Multi-Agent Systems

by

Craig Maxwell

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Monday January 27, 2025 at 10:00 AM.

Student number:	4735102	
Project duration:	September 4, 2023 – January 27, 2025	
Thesis committee:	Dr. A. Jamshidnejad,	TU Delft, Supervisor
	Dr. E. van Kampen,	TU Delft, Chair
	Dr. A.A. Nunez Vicencio,	TU Delft, Examiner
	PhD candidate. M. Baglioni,	TU Delft, Additional

Cover: Camp Fire Rages in California, NASA Earth Observatory image by Joshua Stevens.

TU Delft

Preface

This thesis represents the culmination of a significant chapter in my life, and I would like to share a snapshot of that journey here. When I first embarked on this research, I was an optimistic student eager to take the novel proposal I was provided and construct a research project from the ground up. However, various factors, both within and beyond my control, culminated during the Covid-19 pandemic and had a devastating impact on my ability to complete the work, leading to a prolonged struggle with self-confidence.

Last year, I returned with the ambition to prove to myself that I could complete this endeavour. Balancing a full-time position as an aerospace engineer, I evaluated my previous work and started anew. This allowed me to redefine my thesis objectives more strategically while integrating with and complementing subsequent research in the department.

During the past year, I have dedicated everything I could to this research. This project has been the most challenging undertaking of my academic and professional career, forcing me to refine my abilities as both an engineer and a researcher. Through this journey, I have become more confident, critical, and strategic in my approach. I firmly believe that engineers bear a professional obligation to address the critical challenges faced by society. For this reason, I chose to apply my thesis to search-and-rescue operations in response to natural disasters. As climate change accelerates, so does the urgency for adaptation solutions, and I hope the skills I have developed during this process will enable me to make meaningful contributions in the future.

I would like to extend my deepest gratitude to Anahita, my daily supervisor. Your expert knowledge, incisive feedback, and profound technical understanding have been pivotal to the success of this research. You have not only guided me scientifically but also taught me the importance of precision and critical thinking. I am also immensely grateful to Mirko, for your technical support, stimulating debates, and constructive feedback throughout this journey. Your encouragement and insights have been instrumental in helping me refine both my scientific skills and my personal resilience.

I am deeply thankful to my family for their unwavering encouragement and belief in me throughout this journey. To my fiancée, whom I met at this institution: without your continued support and belief in me, I would not have found the strength to return and finish this chapter. Finally, to all the friends and colleagues at the TU Delft Faculty of Aerospace Engineering: I am continuously inspired by your achievements and dedication, which motivate me to aim higher.

Craig Maxwell
Delft, January 2025

Executive Summary

The research presented in this thesis investigates the formulation, design, application, and analysis of a novel controller termed a Model Predictive Fuzzy Control (MPFC) controller for the mission-planning of a multi-agent system in a system with dynamic and uncertain states. The proposed MPFC architecture integrates supervisory Model Predictive Control (MPC) with Fuzzy Logic Control (FLC) to balance the predictive planning of MPC and real-time adaptability and simplicity of FLC.

Multi-agent systems operating in complex environments face significant challenges, including the need for real-time coordination, adaptive control, and decision-making under uncertain conditions. Existing control strategies exhibit limitations when applied to such scenarios, for instance MPC may struggle with the size and complexity of the optimisation required while FLC cannot adapt agent behaviour to the system over time. The primary objective of this research is to explore the suitability of the MPFC method for this type of system by defining a generic mathematical model for the MPFC controller, implementing it for a specific case study, and performing performance analysis, sensitivity analysis, and design exploration of the MPFC by simulation.

The case study is selected as a multi-agent system for victim detection in a Search and Rescue (SaR) operation in response to a natural disaster. Models are developed for fire propagation, wind dynamics, and agent dynamics, which include dynamic, unknown, and uncertain states which the multi-agent system must adapt performance to.

This research delivers several contributions to the field of autonomous control of multi-agent systems. A *generic mathematical formulation of the MPFC* for the selected control application is developed, enabling the application of this method to other mission-planning case studies. The *implementation of the MPFC model to a SAR case study*, in which a specific case study is developed and demonstrated, combines models used in other fields of research and provides a platform for simulation and analysis of the controller. The *performance analysis* of the MPFC against two other controller architectures to validate the advantages of the MPFC against other control methods in the case study. The *sensitivity analysis* of the MPFC in the context of the case study to understand the influence of design parameters on the controller performance. Finally, the *design exploration* of the MPFC to understand design choices in the configuration of the controller and their impact on controller performance.

The results show that the MPFC can consistently outperform other control methods in a range of scenarios, and exhibits robust performance under varying conditions in the case study. However, numerous limitations and challenges in the design of the MPFC are also identified and these present opportunities for further research and improved applications of the technique. This is still an immature control technique and there are many opportunities for further research to make improvements and contributions to this field.

Contents

List of Figures

List of Tables

Nomenclature

Nomenclature

Acronyms

FIS	Fuzzy Inference System
FLC	Fuzzy Logic Control
MF	Membership Function
MPC	Model Predictive Control
MPFC	Model Predictive Fuzzy Control
SaR	Search and Rescue
TSK	Takagi-Sugeno-Kang
UAV	Unmanned Aerial Vehicle

Agent Model Variables

σ	Agent sensor information confidence degradation factor
θ^{travel}	Agent travel direction (rad)
$\boldsymbol{\phi}^{\text{agent}}(k)$	Agent state vector (all agents) at simulation time step k
$\boldsymbol{\phi}^{\text{search}}(\boldsymbol{s}_m, k)$	Search state vector at cell $\boldsymbol{s}_m$ and time k
$\boldsymbol{\tau}_a^{\text{task}}(k) \in \{0, 1\}$	Current task of agent a at time k (0: travel, 1: scan)
$\boldsymbol{M}^{\text{c,building}}$	Coarsened building occupancy map
$\boldsymbol{M}^{\text{c,downwind}}(k)$	Coarsened downwind map at simulation time step k
$\boldsymbol{M}^{\text{c,fire}}(k)$	Coarsened fire state map at simulation time step k
$\boldsymbol{M}^{\text{c,victim}}(k)$	Coarsened victim map at simulation time step k
$\boldsymbol{M}^{\text{scan}}(k)$	Scan certainty of each environment map cell at time k
$\boldsymbol{s}_a^{\text{location}}(k)$	Location vector of agent a at time k
$\boldsymbol{s}_a^{\text{target}}(k)$	Target cell vector of agent a at time k
$\boldsymbol{t}_a^{\text{scan}}(k)$	Remaining scan time for agent a at time k (s)
$\boldsymbol{t}_a^{\text{travel}}(k)$	Remaining travel time for agent a at time k (s)
Δx^{search}	Length of cells mapped by the agent in the x-axis (m)
Δy^{search}	Length of cells mapped by the agent in the y-axis (m)
$\eta \in [0, 1]$	Sensor accuracy of an agent
n^{queue}	Number of target cells that are queued for the agent to visit by MPC
$n^{\text{x,search}}$	Number of cells mapped by agents in the x-axis
$n^{\text{y,search}}$	Number of cells mapped by agents in the y-axis
$t^{\text{scan-cell}}$	Time to scan a cell (s)
v^{airspeed}	Agent airspeed ($\mathrm{m\,s^{-1}}$)
v^{ground}	Agent ground speed ($\mathrm{m\,s^{-1}}$)

Controller Model Parameters

$c^{\text{obj},1}$	Objective function non-fire proximity factor
$c^{\text{obj},2}$	Objective function fire proximity factor
$\boldsymbol{\theta}^{\text{output}}(k)$	Vector of FLC/controller parameters at simulation time step k
$\boldsymbol{I}$	FLC input parameters
$\boldsymbol{M}^{\text{att}}(a,k)$	Attraction map for agent a at time k
$\boldsymbol{M}^{\text{fire-risk}}(k)$	Fire risk map at simulation time step k
$\boldsymbol{M}^{\text{priority}}(k)$	Priority map at simulation time step k
$\boldsymbol{M}^{\text{response}}(a,k)$	Response time map for agent a at time k
Δk^{MPC}	Number of simulation time steps between each MPC prediction
Δk^{pred}	Number of simulation time steps in prediction horizon
$f_a^{\text{agent-action,FLC}}(\cdot)$	Agent action function from FLC output for agent a
$f_a^{\text{FLC}}(\cdot)$	FLC function for agent a
$n^{\text{MF,in}}$	Number of input membership functions to the FIS
$n^{\text{MF,out}}$	Number of output membership functions
n^{MPC}	Number of MPC steps over prediction horizon
$u_a(k)$	Control input to agent a at simulation time step k

Environment Model Variables

$\boldsymbol{\phi}^{\text{env}}(\boldsymbol{s}_m,k)$	Environment state vector at cell $\boldsymbol{s}_m$ and time step k
$\boldsymbol{F}(k)$	Fire spread probability map at simulation time step k
$\boldsymbol{F}(k)$	Fire spread probability map at simulation time step k
$\boldsymbol{M}^{\text{building}}$	Proportion of cell environment map cell occupied by buildings
$\boldsymbol{M}^{\text{burn-time}}(k)$	Burn time of each environment map cell at simulation time step k
$\boldsymbol{M}^{\text{fire}}(k)$	Fire map at simulation time step k
$\boldsymbol{M}^{\text{structure}}$	Flammability of each environment map cell
$\boldsymbol{M}^{\text{victim}}$	Number of victims in each environment map cell
$\boldsymbol{M}^{\text{wind-direction}}$	Wind direction factor in fire spread model
$\boldsymbol{M}^{\text{wind-distance}}$	Wind distance factor in fire spread model
$\boldsymbol{W}$	Wind spread map
Δx^{env}	Length of environment cells in the x-axis (m)
Δy^{env}	Length of environment cells in the y-axis (m)
θ^{wind}	Wind direction (rad)
c^{fs1}	Fire model base spread probability constant
c^{fs2}	Fire model wind sensitivity constant
c^{wm1}	Wind model wind direction influence constant
c^{wm2}	Wind model wind direction scale constant
c^{wmd}	Wind model distance constant
$l^{\text{x,env}}$	Length of the disaster environment in the x-axis (m)

$l^{\text{y,env}}$	Length of the disaster environment in the y-axis (m)
n^{c}	Total number of cells in bounded environment
$n^{\text{x,env}}$	Number of environment cells in x-axis
$n^{\text{y,env}}$	Number of environment cells in y-axis
p^{ignition}	Probability of ignition of a cell s at a time step k based on proximity to active fires
r^{wind}	Radius of cells in $x-$ and $y-axis$ that an active fire may ignite
t^{burnout}	Set time for cell to transition from 'catching fire' to 'extinguished' (s)
$t^{\text{burn}}((s)_c)$	Time since cell ignition (s)
t^{ignition}	Set time for cell to transition from 'catching fire' to 'burning' (s)
v^{wind}	wind velocity $(\mathrm{m\,s^{-1}})$

Indexes

a	Agent index
f	Active fire index
k	Simulation time step index
m	Flattened cell index, $m \in \{1, \ldots, n^c\}$ where $n^c = n^x \cdot n^y$
q	Task queue index
s	Simulation index, $s \in \{1, \ldots, n^{\text{sim}}\}$

Mathematical Symbols

$\bar{\cdot}$	Mean state value
$\hat{\cdot}$	Predicted state value
$\mathcal{F}(\cdot)$	Deterministic environment dynamics function
$\mathcal{G}(\cdot)$	Uncertain environment dynamics function
$\mathcal{H}(\cdot)$	Agent dynamics function
$\mathcal{K}(\cdot)$	Search state update function

Performance Metrics

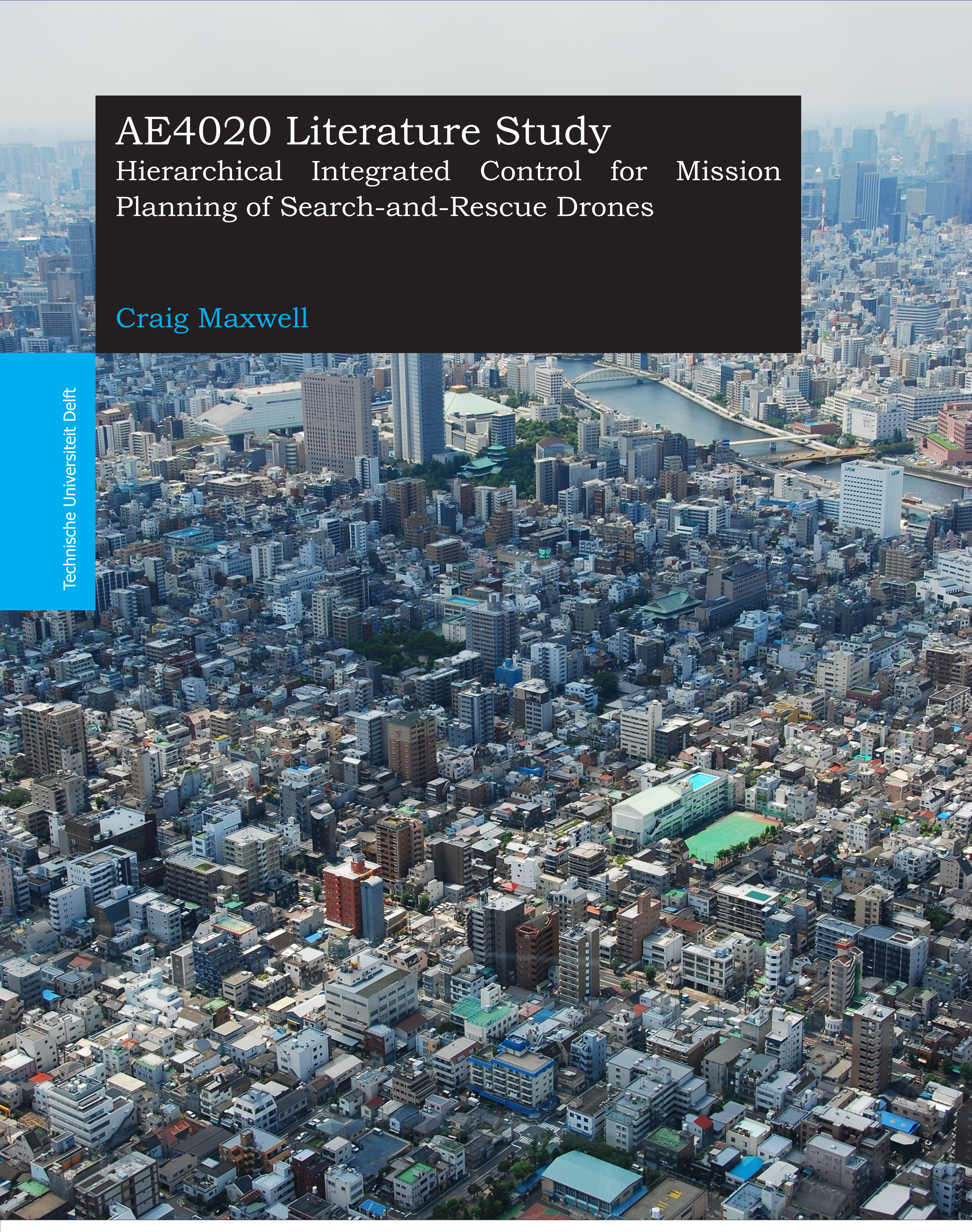
$(\overline{J}_{0.025}(k), \overline{J}_{0.975}(k))$	95% confidence interval bounds for mean instantaneous objective
$(\bar{t}^{\text{opt}}_{0.025}(k), \bar{t}^{\text{opt}}_{0.975}(k))$	95% confidence interval bounds for mean optimisation time
$\hat{f}^{\text{obj}}$	Predicted objective function at each cell/time step
$\hat{J}(k)$	Predicted instantaneous objective at simulation time step k
$\overline{J}(k)$	Mean instantaneous objective over n^{sim} simulations at simulation time step k
$\bar{t}^{\text{opt}}(k)$	Mean optimisation time of MPC or MPFC prediction step over n^{sim} simulations at simulation time step k
f^{obj}	Objective function evaluated at each cell/time step
$J_s(k)$	Instantaneous objective for simulation i at time step k
$t^{\text{opt}}_s(k)$	Optimisation time of MPC or MPFC prediction step for simulation s at simulation time step k

Simulation Parameters

$\boldsymbol{s}$	Spatial cell location
$\Delta\boldsymbol{s}_m$	Area of a single cell m (m^2)

Δt	Simulation time step duration (s)
$\eta^{\text{fire-spread}} \in [0, 1]$	Random fire spread probability threshold for ignition
c^{coarsen}	Agent spatial information map coarsening factor
k	Simulation time step
k'	Simulation time step dummy variable during prediction
N	Total number of time steps in simulation
n^{a}	Number of agents
$n^{\text{c}} = n^x \cdot n^y$	Total number of cells, for specified grid dimension $[n^x, n^y]$
$n^{\text{max-func-eval}}$	Max number of function evaluations for optimisation
$n^{\text{max-generations}}$	Max number of generations for genetic algorithm
$n^{\text{max-iterations}}$	Max number of iterations for optimisation
$n^{\text{population}}$	Population size for genetic algorithm
n^{sim}	Number of simulations used to compute mean and confidence intervals
r^{local}	Local radius defining the local prediction map size (See section 4.4)
T	Simulation end time (s)

Literature Study



AE4020 Literature Study
Hierarchical Integrated Control for Mission Planning of Search-and-Rescue Drones

Craig Maxwell

Technische Universiteit Delft

TU Delft Delft University of Technology

Challenge the future

Contents

Nomenclature

Acronyms

Symbol	Description	Dimensions	Units
DMC	Dynamical Matrix Control		
FANET	Flying Ad-hoc Network		
FMRLC	Fuzzy Model Reference Learning Controller		
GPC	Generalised Predictive Control		
LIDAR	Light Detection and Ranging		
LTI	Linear Time Invariant		
MF	Membership Function		
MIAC	Model Identification Adaptive Control		
MILP	Mixed Integer Linear Programming		
MIMO	Multiple Input Multiple Output		
MPC	Model Predictive Control		
MRAC	Model Reference Adaptive Control		
NIST	National Institute of Standards and Technology		
PID	Proportional-Integral-Derivative		
PMRAC	Predictive Model Reference Adaptive Control		
QP	Quadratic Programming		
SAR	Search and Rescue		
SfM	Structure from Motion		
SISO	Single Input Single Output		
SLAM	Simultaneous Localisation and Mapping		
UAV	Unmanned Aerial Vehicle		
UAV	Unmanned Ground Vehicle		
USAR	Urban Search and Rescue		

Greek Symbols

Symbol	Description	Dimensions	Units
λ	Scalar cost function weight		
μ	Membership function		

Σ	Model		
$\boldsymbol{\theta}$	Vector of intelligent controller parameters		

Roman Symbols

Symbol	Description	Dimensions	Units
X	Universe of discourse		
$\boldsymbol{A}$	State matrix		
$\boldsymbol{B}$	Input matrix		
$\boldsymbol{C}$	Output matrix		
$\boldsymbol{D}$	Feedthrough matrix		
$\boldsymbol{d}$	Unmeasured disturbances		
$\boldsymbol{H}$	Toeplitz matrix		
$\boldsymbol{I}$	Identity matrix		
$\boldsymbol{u}$	Control inputs		
$\boldsymbol{v}$	Measured disturbance		
$\boldsymbol{w}$	Sensor measurement noise		
$\boldsymbol{W}_u$	Cost function weighting matrix		
$\boldsymbol{W}_y$	Cost function weighting matrix		
$\boldsymbol{x}$	System states		
$\boldsymbol{x}_{\text{ref}}$	Reference system states		
$\boldsymbol{y}$	Outputs		
J	Cost function		
K_d	Derivative gain		
K_i	Integral gain		
K_p	Proportional gain		
n_u	Control horizon		
n_y	Prediction horizon		

Superscripts

Symbol	Description	Dimensions	Units
$\rightarrow$	Predicted values over prediction horizon		
k	Current sampling step		

1 Introduction

This literature review is for a thesis project in which a hierarchical predictive adaptive fuzzy control system for mission-planning of distributed multi-agent systems is proposed. The thesis project proposes the application of this system for the control of a quadrotor system for search-and-rescue (SAR) missions in an urban disaster zone. The desired property of the control system is that it is able to adjust the behaviour of the agents by using model-based predictions about the future state of the system. This should improve performance of the quadrotor system, especially in highly dynamic environments. The objective of this literature review is to review theory related to the thesis project and detail the current state-of-the-art in the relevant fields.

1.1. Disaster Environment Overview

The selected operating environment for the system is an urban disaster zone, which contains stochastic and dynamic states. The objective of the quadrotor system is to *"rescue the maximum number of victims in the minimum possible time, while minimising risk to rescue personnel"*. The quadrotor system must attempt to maximise this objective while performing the tasks it is required to do.

1.2. Controller Overview

In this review, the proposed controller is named as a *"Predictive Model Reference Adaptive Controller"*, or PMRAC. The PMRAC controller has two loops: an inner loop in which a fuzzy controller controls the quadrotor to follow the reference state trajectory, and an outer loop in which a model predictive control (MPC) controller performs online tuning of the intelligent controller parameters. MPC is a control method in which, at each sampling instance, an optimal sequence of predicted control inputs is determined that optimise a constrained cost function based on predicted system states over a receding horizon. The first control step in the sequence is implemented and the process is repeated at the next sampling instance. In the case of the PMRAC controller, this control step is the set of parameters for the intelligent controller. Two components make up the MPC controller: a *predictive model* which is used to predict future system states and an *optimiser* which is used to perform an optimisation on a constrained cost function.

1.3. Literature Review Overview

This literature review is structured as follows: Chapter 2 includes a review of disasters, disaster management, search-and-rescue, search-and-rescue robotics, requirements for an autonomous multi-agent quadrotor UAV system for search-and-rescue, and finally the identification of relevant states for the UAV system in the disaster zone. Chapter 3 reviews relevant control theory for the proposed control system, including MPC, adaptive control, fuzzy control, and network structures. Finally, in chapter 4, the problem statement is given, research questions are identified, and the conclusion to the literature review is made.

2 Quadrotor UAV Search and Rescue Review

2.1. Disasters

Disasters are events which cause major disruption to the functioning of a society, resulting in human, material, economic, and/or environmental damage [1]. They can be broadly classified into natural disasters, which occur due to natural processes and phenomena, and man-made disasters, which occur due to human activity. Natural disasters include hydro-meteorological (floods, storms, droughts, wildfires), geophysical (earthquakes, volcanic activity, tsunamis, and landslides), and biological (toxins and pathogenic microorganisms) disasters; while man-made disasters include conflicts, famine, industrial accidents, and transport accidents [2].

To understand the scale of major disasters, examples from recent history can be inspected. For instance, the 2010 Haiti earthquake struck near the Port-au-Prince metropolitan region, resulting in between 122,000 and 316,000 casualties[1]. The cumulative impact of hydro-meteorological and geophysical disasters between 1998 and 2017 is estimated by the UN office for Disaster Risk Reduction (UNDRR) as responsible for 1.3 million deaths, 4.4 billion people injured, displaced, or homeless, and US$2,908 billion of economic losses [4].

A bar plot of average annual casualties from natural disasters in each decade is shown in figure 2.1. Between 1998-2017, earthquakes accounted for 56% of total deaths despite only accounting for 7.8% of reported disasters [6], making them by far the deadliest type of natural disaster in recent decades[2].

2.2. Disaster Management

Disaster management is the preparation for and response to a disaster. The goal of disaster management is to minimise disruption to the community in the form of casualties, injuries, displacement, and economic impact when a disaster occurs. The disaster management process can be split into the following phases:

1. *Prevention and mitigation* - includes long-term strategies such as risk assessment, simulation and modelling, structural mitigation (e.g. earthquake-resistant structures), and training.
2. *Prediction and warning* - includes prediction/diagnosis of a disaster and assembling resources to mitigate the disaster, such as early warning systems and monitoring systems.
3. *Preparedness* - involves minimisation of negative impacts of the disaster through planning, such as resource inventory, logistical, evacuation, and communication planning.
4. *Response and Relief* - occurs directly after a disaster, and is highly dependent on the type of disaster. Response involves data collection tasks (situation analysis, damage assessment, crisis

[1] Sources vary, with 122,000 - 167,000 casualties estimated according to [3] or 316,000 according to government estimates.

[2] In previous decades, droughts have been the deadliest form of natural disaster

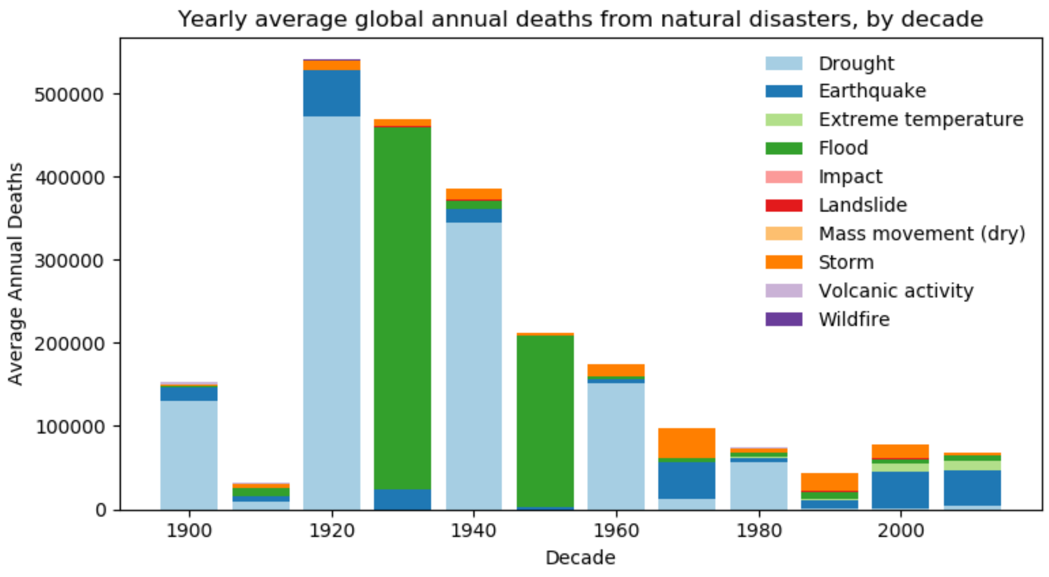


Figure 2.1: Average annual casualties caused by natural disasters per decade [5]

mapping) and logistical tasks (evacuation, dispatching of resources to the disaster zone), while relief is concerned with saving the lives of those affected by the disaster. This includes search and rescue, rubble and debris removal, logistics, and delivery of relief supplies.

5. *Recovery, reconstruction, and rehabilitation* - involves long-term activities that occur after the disaster, such as reconstruction of infrastructure, housing, communication networks, water and hygiene systems. Another critical task is evaluation, to determine whether the disaster management can be improved for future disasters.

Response and relief is the most critical phase when it comes to saving lives, as the survival rate of disaster victims is highly dependent on response time. In [7], Barbera and Cadoux claim that victim survival rates drop off dramatically 48 hours after a disaster, while in [8], Coburn, Spence, and Pomonis develop a model for victim survival rates after a disaster, and determine that the survival rate is very sensitive to improvements in rescue efficiency during the first 36 hours, after which the sensitivity decreases drastically. Methods to improve the efficiency of *response and rescue* tasks therefore have a high potential of reducing casualty rates.

2.3. Urban Search and Rescue

Search and rescue (SAR) a task within the *relief* phase of disaster management, with the objective to *"rescue the largest number of people in the shortest time, while minimizing risk to rescuers"* [9][3]. When carried out in urban environments, it is termed urban search and rescue (USAR). Due to their high population density and developed infrastructure, urban areas are especially sensitive to disasters, and have the potential for far greater numbers of victims than other environments. Most victims in disaster zones are trapped underneath collapsed infrastructure, often in voids with no easy access, making detection and extrication of victims extremely difficult and hazardous. Additionally, the disaster zone may present many other hazards to victims and rescuers, and may include unstable rubble/buildings, sharp metal and glass, fires or explosions, chemicals, ruptures water lines, exposed electrical cables, and so on.

2.4. Search and Rescue Robotics

Research into search and rescue robotics began in the 1990's, motivated by the 1995 Great Hanshin earthquake in Kobe, Japan. The scale of such disasters tends to overwhelm the resources of the USAR

[3] This is given in the context of natural disasters.

mission, and robotic systems have the potential to assist with difficult tasks or automate large tasks. Additionally, as mentioned in section 2.1, disaster zones may contain many hazards which present danger to rescuers. Robotic systems can be applied to tasks which would put rescuers in danger, placing the danger on replaceable products instead of valuable human lives. In [10], Tanzi et. al. identify two main requirements for robotic systems in search and rescue: firstly, they must have a sufficient level of autonomy as communication with the control centre is likely to be sporadic, and secondly the system must have a sufficient level of reliability with respect to accidental (safety) or malicious (security) risks.

In [11], Murphy et al. identify four categories of SAR robotic *modality*:

- Unmanned ground vehicles (UGVs)
- Unmanned aerial vehicles
- Unmanned water surface vehicles
- Unmanned underwater vehicles

and three categorised of *size*:

- *Man-packable* - robot can be completely fitted into several backpacks.
- *Man-portable* - robot can be carried short distances by several people.
- *Maxi* - robot must be transported by vehicle.

Each modality is suited to a different operating environment, and has its own set of strengths and weaknesses. UAVs have the highest manoeuvrability, highest speed, lowest cost, and lowest complexity. This is because commercial, mass-produced UAVs can operate in disaster environments, while other modalities, such as UGVs, must be highly specialised and complex to be able to navigate uneven, unstable terrain. Disadvantages of UAVs include low endurance due to battery limitations and low payload capacity. Therefore, UAVs are the most suitable choice for large-scale USAR mission planning systems, and will be the modality considered in this project.

2.4.1. Robotics Systems for Search

Search is the task of searching for and identifying victims in a disaster environment, allowing appropriate responses to be arranged in order to aid the victims. This is a major focus of SAR robotics research. In [10], Tanzi et. al. propose a multi-task UAV system with several types of UAV cooperating, in which a quadrotor system performs autonomous detection and classification of people in a disaster zone. The quadrotor UAVs are equipped with a visual sensor payload and an image recognition algorithm is applied to the signal in order to detect and classify people. People are classified into victim or rescuer, adult or child, and level of injury. Additionally, a signature is applied to each person to keep track of them. A separate algorithm is proposed to detect groups of victims, determine their direction and speed, and predict their future positions. Another algorithm is suggested for detection of buried victims by using a sensor to detect electromagnetic emissions from mobile devices. In [12], Rudol and Doherty demonstrate an autonomous UAV system which uses a combination of visual and thermal sensors for the detection and classification of human-sized heat signatures. In [13], Andriluka et al. examine the performance of several people-detection pattern recognition algorithms, concluding that part-based models result in higher performance, as they can deal with partially occluded people. In [14], Apvrille et. al. demonstrate a quadrotor system capable of autonomous navigation in indoors environments with a single 720p camera sensor. Data from the camera is used to reconstruct the environment for navigation and to identify, classify, and track people.

Aside from victim detection, work has also focused on feature detection. In [15], Kong et al. use a multi-agent quadrotor UAV system for the detection of access holes in rubble.

2.4.2. Robotics Systems for Mapping

Mapping is the task of building a digital representation of the disaster zone. Most mapping tasks are performed by UAVs, which have the advantage of high manoeuvrability. In the system proposed by Tanzi et al. [10], a group of cooperating fixed-wing UAVs equipped with LIDAR (light detection and ranging) sensors are proposed to either build 3D digital surface models of the disaster zone, or alternatively to build 2D maps using optical sensors. An alternative mapping method involves 3D reconstructions of buildings, for which quadrotor UAVs are well suited due to smaller mapping areas and higher required manoeuvrability. Simultaneous Localisation And Mapping (SLAM) is a mapping method that allows robots to perform 3D mapping and localise themselves in the environment at the same time, and is commonly used for autonomous navigation. In [16], Schmuck and Chli present a collaborative multi-agent quadrotor UAV SLAM algorithm, where UAVs navigate and build a local map of the environment, which is regularly sent to a centralised computer and fused with the local maps of other agents. The SfM (Structure from Motion) algorithm is used for 3D mapping by using features extracted from a sequence of images to build a 3D model of the environment. In [17], Verykokou et. al. compare the performance of commercial and open source image recognition software in an SfM algorithm to detect features in images of a destroyed building captured from a UAV. They then propose a fast algorithm for the determination of overlapping images.

2.4.3. Robotics Systems for Coverage

Coverage is the task of optimising sensor coverage of a given area. This can either include optimising sensor placement and numbers for continuous coverage of a given area or optimising coverage over a given area while minimising some objective, such as time. This task can be applied to *search* for victims, ensuring network coverage, and so on.

2.4.4. Robotics Systems for Logistics

Logistics are another necessary task in search and rescue operations that can be automated, in order to remove danger from rescuers or to allow workforce to be focused elsewhere. Some such tasks involve delivery of equipment and supplies to rescue teams or distribution points in the disaster zone. UAVs are suitable for the transport of high-priority lightweight packages such as small rescue equipment and medical supplies. In the package delivery industry, DHL, Amazon, FedEx, and Ali Baba are among companies with ongoing development programmes for autonomous UAV package delivery systems. Furthermore, commercial solutions for autonomous UAV logistics systems are already available in some countries. Zipline Inc. [18] is a company that operates a system of autonomous fixed-wing UAVs in Rwanda and Ghana for fast delivery of urgent medical supplies, such as blood and vaccines, to medical centres. Their system has achieved success in these countries, as their poor road infrastructure means that deliveries by land can take upwards of five hours compares to an average delivery time of 30 minutes by their system.

2.4.5. Other Applications of Robotics Systems

Structural inspection could be performed by robotic systems which use specialised sensors to scan the inside and outside of structures and determine whether they are stable and safe for rescuers to enter. In [19], Torok, Golparvar-Fard, and Kochersberger present an algorithm for automatic 3D crack detection from images of damaged buildings. In their paper, they propose the implementation of their algorithm in a small autonomous ground robot, which will navigate inside the building to perform structural inspection. In [20], Erdelj et al. mention the use of autonomous UAVs for this purpose.

Suggested autonomous robotic solutions to *medical assessment and intervention* tasks include using agile robots to open communication channels between medical personnel and victims, inspection of the victim with diagnostic or visual sensors, or provision of life support or medicine to inaccessible victims.

Ground robots could be used for *extrication and evacuation* of victims, particularly when the disaster zone poses a safety risk to rescue personnel, such as chemical, biological, and radioactive hazards.

Robotic machinery and exoskeletons could be used for *rubble removal*, allowing heavier rubble to be moved faster without requiring heavy machinery, which is difficult to manoeuvre into position when there is uneven and unstable terrain.

Finally, robotic systems can be used to establish a *temporary communications infrastructure* to be used by the search-and-rescue operation, as ground-based networks are often damaged or destroyed following a disaster. Tanzi et al. [10] propose a system of autonomous blimps operating to establish a communications backbone above the disaster zone, allowing communication between autonomous vehicles or rescuers. In [21], Bekmezci, Sahingoz, and Temel survey flying ad-hoc network (FANET) methods for multi-agent UAV systems. They identify the advantages of this type of communication infrastructure to be cost, scalability, survivability, and speed.

2.4.6. Proposed System

In this chapter, a top-level review of disasters and disaster management has been performed in sections 2.1 and 2.2, and urbanised environments have been identified as the most critical in terms of saving lives in the event of a major disaster. In section 2.3, search-and-rescue was identified as a critical stage of disaster management in which robotic systems could provide assistance. In section 2.4, a review is provided of academic research in this area, types of search-and-rescue robots, and search-and-rescue robot tasks. UAVs are identified as the most versatile type of robot in terms of performing search-and-rescue tasks, and are chosen as the focus of this project. Of the tasks identified in section 2.4.6, UAVs are a suitable choice for search, mapping, coverage, navigation, logistics, structural inspection, medical assessment and intervention, and temporary communications infrastructure. The goal of this project is therefore to develop an autonomous robotic system which can be applied to urban search-and-rescue missions to perform these tasks. The requirements for the system properties are:

- *Scalability* - ability to scale well in order to adapt to size of disaster zone. As the disaster zone can range from individual buildings to entire cities the system must have sufficient performance in all areas.
- *Fault tolerance / stability* - loss of autonomous agents must have minimal impact on system performance.
- *Responsiveness* - react to changes in the systems state by modifying behaviour. As the disaster zone is complex and dynamic, the system must be able to adapt its behaviour accordingly to remain effective.
- *Deployability* - the system must be able to be deployed quickly in rough terrain.
- *Endurance* - the system must operate over an extended period of time.

The *scalability* requirement can be met by distributed and decentralised systems, as the volume of data that must be communicated or processed for each agent reaches some limit regardless of the number of agents. The *stability* requirement can also be met by decentralised or distributed systems, as they can continue to operate in the event of failures to some nodes. The *responsiveness* requirement can be met by designing a suitable controller for the system. The *deployability* requirement can be met by using man-portable or man-packable equipment for the system. Finally, the *endurance* requirement can be directly met by blimps, or by other forms of UAV that can return to charging points. Quadrotors are selected as the type of UAV for the search-and-rescue system. This is as they are the most manoeuvrable type of UAV, are commercially available, and can perform the widest range of required tasks.

In this thesis project, a decentralised predictive model-based adaptive intelligent controller is proposed for the control of the quadrotor UAVs. No other publications were found in which this type of controller was used. A block diagram of the proposed controller is shown in figure 2.2. The controller consists of two loops: an inner loop where an intelligent controller is used to control the quadrotor inputs, and an outer loop where a model predictive controller (MPC) is used to tune the parameters of the intelligent controller. A detailed review of the proposed control system and relevant control theory is provided in chapter 3.

2.5. Simulation of USAR Missions

The control system proposed in the thesis project will be verified through simulation. Therefore, it is important to build an accurate model of the disaster zone in which the system operates. There is currently no standardised digital disaster zone simulator for USAR missions, and limited research has been published on this issue. In [22], Kitano et al. identify the requirements for an accurate SAR simulation:

- *Building and housing damage* - simulation of the damage done to buildings.
- *Fire* - simulation of fires igniting and behaviour of spreading, taking into consideration building types and weather.

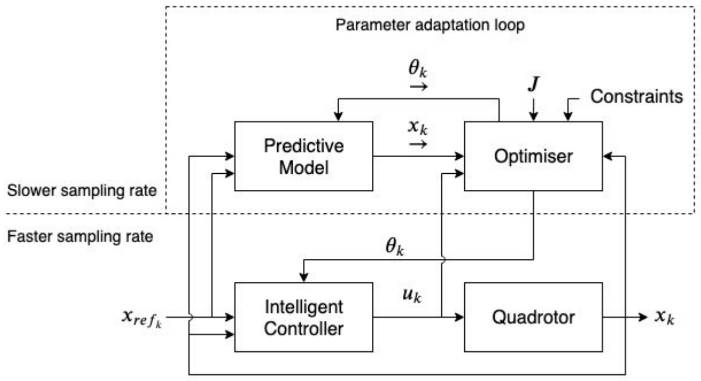


Figure 2.2: Model predictive adaptive control (MPAC) block diagram for a single agent. This block diagram does not include disturbances. u is the vector of control inputs to the quadrotor, x is the vector of system states, x_{ref} is the vector of reference system states, $\boldsymbol{\theta}$ is the vector of intelligent controller parameters, and J is the cost function. The model predictive control (MPC) block is in the outer loop, shown in the dashed box. The *"predictive model"* block makes predictions on future system states and the *"optimiser"* block calculates an optimal set of parameters for the intelligent controller which minimises a cost function. The inner loop operates at a fast sampling rate in order to control the fast dynamics of the quadrotor while the outer loop operates at a slower sampling rate due to higher computational complexity and the slower dynamics of the disaster zone.

- *Life-line damage* - simulation of damage to life-line instrastructure, including roads, electrical grid, water and gas lines, and so on.
- *Victim modelling* - simulation of victims and their state after the disaster.
- *Refugee behaviour modelling* - simulation of large numbers of victims attempting to escape from disaster site. This can impede traffic from the SAR mission which is attempting to access the disaster zone.

The National Institute of Standards and Technology (NIST) USAR test facility is a standardised environment for USAR disaster simulation, and is used to test robotic systems in real environments. In [23], Wang, Lewis, and Gennari develop a virtual simulation of the NIST USAR test facility using a game engine, which implements robot dynamics, environment dynamics, and sensor modelling along with the modelled environment. This environment is designed for ground robotic systems, with features such as uneven terrain, small spaces, and unstable objects. However, this environment is small and therefore not suitable for testing multi-agent UAV systems, but it could provide a basis for a full urban disaster zone simulation.

The simulation of the disaster environment will be built using *Gazebo* and *ROS*. *ROS* is an open-source robotics simulation framework for developing robot software. *Gazebo* is an open-source robotics simulation software, with libraries of robot models, environments, sensor models, and so on. This will be interfaced with MatLab, where the main control algorithm will be designed, to run the simulation.

2.6. Summary

In this chapter, a multi-agent multi-task quadrotor UAV system has been selected for urban search-and-rescue missions. The requirements of the system and the relevant disaster zone states were identified. The concepts of disasters and disaster management were introduced in sections 2.1 and 2.2 respectively. Search-and-rescue was identified as a critical stage of disaster management and discussed in section 2.3 and urban disaster zones were identified as the environments susceptible to the greatest number of victims. Types of robotic systems for search-and-rescue were discussed in section 2.4, and a detailed review of recent literature for different search-and-rescue robotics tasks was provided. This provided the context for section 2.4.6, where the type of UAV system for autonomous multi-task search-and-rescue was identified, and finally section 2.5, where the simulation of USAR missions is discussed.

3 Control Methods

As mentioned, a distributed PMRAC controller is proposed for the cooperative mission-planning of multi-agent systems. In the controlled system, MPC controllers will be used to perform online tuning of multiple intelligent controllers in several quadrotors based on predictions of the future system states and optimisation of an objective function. The system is decentralised, meaning that the information of system states is based on local measurements and information communicated with nearby agents. In this review, the *controlled system* is the system of quadrotors while the *control system* is the PMRAC controller which directs the actions of the entire *controlled system*. This chapter will provide a review of the theory and the state-of-the-art for the relevant control methods.

3.1. Model predictive control

Model predictive control (MPC) is a control method in which the control law is determined by computing an optimal sequence of control inputs over a receding horizon at each time step and implementing the first control input in the sequence. The two elements in a model predictive controller are a *predictive model* of the system dynamics and an *optimiser*. The predictive model predicts the future system states given the current system states and control input, while the optimiser solves an online optimisation problem to determine a sequence of future inputs which minimise a cost function.

This section will introduce the theory of MPC and different MPC methods, starting with linear MPC in section 3.1.5, then nonlinear MPC in section 3.1.6, robust MPC in section 3.1.7, explicit MPC in section 3.1.9, stochastic MPC in section 3.1.10, and a review of the current state-of-the-art and relevant aerospace applications in section 3.1.11.

3.1.1. Introduction to model predictive control

First, it is important to establish the mathematical basis of model predictive control. In this report, the nomenclature k is used for the current time step, n for the horizon size, and T for the time step duration. The subscripts $\leftarrow$ and $\rightarrow$ denote the past values and predicted future values of a variable respectively while the notation $\sim$ is used to denote sequences. A simplifying assumption made in this report is that the system is not multi-rate [1], meaning that sampling and control time steps are synchronised.

The prediction scheme used in MPC is illustrated in figure 3.1, where $\boldsymbol{u}_k$ is the control input at time step k, $\tilde{\mathbf{y}}$ is the sequence of system states, and $\tilde{\mathbf{y}}_{ref}$ is the sequence of reference system states. The system states are predicted over the output horizon, $[k + n_y]$, and the control input sequence is predicted over the control horizon, $[k + n_u]$, after which the control input is assumed to remain constant, or $\Delta\boldsymbol{u}_k = \mathbf{0}$ for $k \in n_u, \ldots, n_y$. This is an appropriate assumption if the control horizon length is greater than the settling time for the system.

Figure 3.2 shows a block diagram of an MPC controller at sampling instance k, where $\bar{\mathbf{x}}_k$ is the vector of system states, $\boldsymbol{x}_k$ is the vector of measured system states, $\boldsymbol{w}_k$ is the sensor measurement noise, $\boldsymbol{v}_k$ is the measured disturbance, $\boldsymbol{d}_k$ is the unmeasured disturbance, $\boldsymbol{u}_k$ is the control input vector, $\underset{\rightarrow}{\boldsymbol{x}}$ is the sequence of predicted system outputs, $\underset{\rightarrow}{\boldsymbol{u}}$ is the sequence of predicted control inputs, $\underset{\rightarrow}{\boldsymbol{x}}_{ref}$ is

[1] A separate field of MPC is focused on MPC for multi-rate systems, but this is not covered in this review.

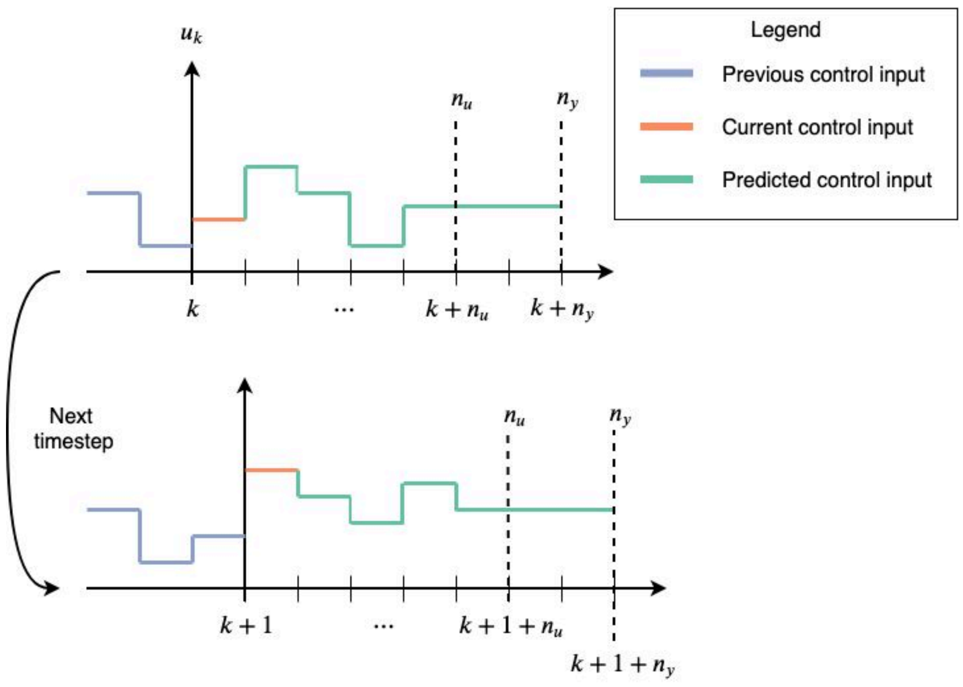


Figure 3.1: Illustration of horizons and control input sequence in MPC

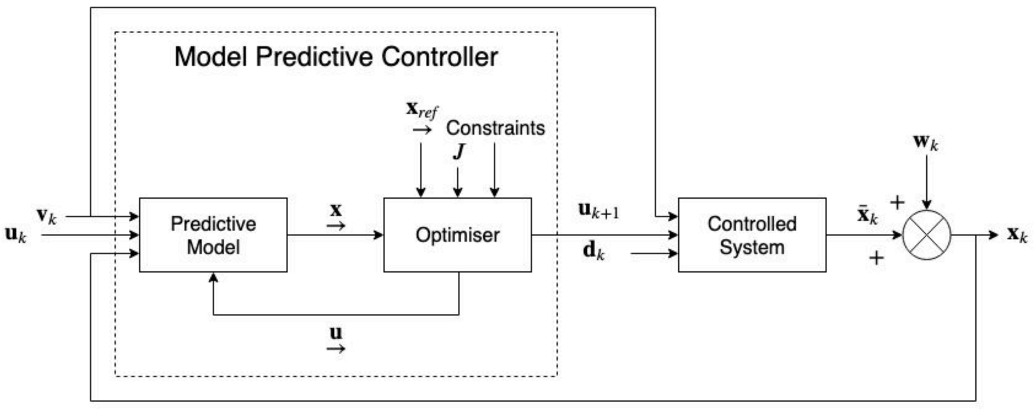


Figure 3.2: Model predictive control block diagram at sampling instance k.

the sequence of reference system states, and J is the cost function. As previously mentioned, the two main blocks in an MPC controller are the *predictive model* and the *optimiser*.

At sampling instance k, the predictive model is initialised with the measured disturbances, past control inputs, and past system states. Using this information, the predictive model predicts the system states at the next sampling instance, $k+1$. Based on the predicted system states and the reference system states, the optimiser then attempts to find an optimal control input at sampling instance $k+1$ by minimising the cost function J subject to constraints. The predicted control input is passed to the predictive model and this process is repeated over the sequence $[k, \dots, k+n_y]$. The first control input in the predicted control input sequence, u_{k+1}, is implemented and the process is repeated at the next sampling instance.

MPC has numerous advantages over other common control methods:

- *Suitability* for a wide range of systems due to several factors. Firstly, the optimisation problem can be customised towards many types of systems, such as non-linear or time-delayed systems. Additionally, the mathematics can be easily extended from SISO (Single-Input Single-Output) systems to MIMO (Multiple-Input Multiple-Output) systems. Finally, the model can be customised for the type of system to be controlled or the desired performance, for instance by including disturbance models, constraints, nonlinear dynamics, time delays, and so on.
- *Constraint handling* is integrated into the optimisation problem.
- *Stability* for linear MPC can be guaranteed under certain weak constraints.
- *Good tracking performance* is obtainable, as the reference system states over the finite horizon is considered in the optimisation.
- *Adaptation* to changing parameters. MPC can be suitable for the control of systems with time-varying parameters, as it needs only a finite number of future system parameters to compute the current control law.
- *Easier computation* compared to infinite horizon control methods.

On the other hand, the main disadvantage of MPC is:

- *High computational complexity* compared to conventional control methods such as PID. This may require the use of powerful processors or more efficient optimisation algorithms in order to implement it in practice.

3.1.2. Development of model predictive control

MPC is a mature control method which was first used in industry in the 1950's, but first entered widespread use in the 1980's. In [24], Lee reviews the progression of MPC from the 1980's to the 2010's. The first widely used MPC algorithm was Dynamical Matrix Control (DMC), which was used in industrial plants in the 1980's. This method relied on time-domain response models and did not model disturbances. Around the same time, another MPC method known as Generalised Predictive Control (GPC) was developed for systems, with the key differences being that is was developed for SISO systems, used a transfer function model, included stochastic processes, and did not model constraints. At the time, these control methods were suitable only for systems with slow dynamics, as the computational complexity and available hardware limited the controllers to slow sampling times.

In the 1990's, the state-space model became the most common model used in literature for MPC controllers. The first *nonlinear MPC* (section 3.1.6) methods were developed, using nonlinear state-space models. *Robust MPC* methods (section 3.1.7) were also developed, ensuring stability of a system subject to disturbances with known bounds.

In the 2000's, the fields of *hybrid MPC* (section 3.1.8) and *explicit MPC* (section 3.1.9) emerged. Additionally, a large volume of research was dedicated to *fast MPC*, using various tricks to design faster optimisation algorithms.

3.1.3. Fast model predictive control

For instance, in [25] Yang and Boyd present an MPC algorithm that calculates the control law online in the millisecond range, two orders of magnitude faster than in previous methods. This was done by exploiting the mathematical structure of the quadratic programming (QP) optimisation problem, causing the number of operations to become linear instead of cubic with respect to the horizon size. Additional tricks used in their algorithm were *warm starting*, where the optimisation problem is initialised using values from the previous sampling instance, and *early termination* of the optimisation problem after a several iterations. In [26], Jerez, Goulart, and Richter present several optimisation methods which allow MPC to be calculated at rates above 1 MHz, again several orders of magnitude faster than the millisecond-range rates from Yand and Boyd. Whereas the first MPC algorithms could only be used for systems with slow dynamics such as industrial plants, today the developments of fast optimisation algorithms and powerful processors have opened opportunities for the application of MPC in systems with fast dynamics, such as UAVs.

3.1.4. Stable model predictive control

Stability is the ability of a system to retain a bounded output given a bounded input [2]. In [27], Mayne et al. review stability and optimality principles for MPC, where they identify two main approaches to ensuring stability in MPC. The first is based on using a Lyapunov function, while the second requires that the state is always shrinking. The main stability methods in MPC are:

- *Terminal equality constraint method* - the constraint $x_{k+n_y} = x_f = \mathbf{0}$ is applied to the optimisation problem, where x_f is the terminal system state. This was first demonstrated for time-varying, constrained, nonlinear, discrete-time systems by Keerthi and Gilbert [28]. The drawback to this is that it may require a large control effort to satisfy the constraint, especially for a small output horizon n_y, resulting in poorer performance. Additionally, the constraint may greatly restrict the set of controllable initial states. The terminal equality constraint is applied to continuous time systems by Chen and Shaw [29] and unconstrained systems by Magni and Sepulchre [30].

- *Terminal cost function method* - a terminal cost $F(\cdot)$ is added to the cost function. Bitmead et al. [31] first applied this method to an unconstrained linear system, using a terminal cost function with the form $F(x) = 0.5x^T P_f x$, where P_f is the terminal value of the Ricatti difference equation. In an alternative method, Rawlings and Muske [32] chose $F(\cdot)$ to be the cost function for the stabilising controller $u = \kappa_f \equiv 0$.

- *Terminal constraint set method* - a set of terminal constraints, x_f, which are in the neighbourhood of the origin and a stabilising controller $\kappa_f(\cdot)$ are employed in the terminal constraint set. This was first proposed by Michalska and Mayne [33].

- *Terminal cost and constraint set method* - both a terminal cost and a terminal constraint set are used. This was first proposed by Sznaier and Damborg [34]. The advantage of this method is that it can closely approximate the infinite horizon problem for constrained nonlinear systems if the terminal cost and constraint set are chosen appropriately.

- *Contraction constraint method* - the constraint $\|x(k+1|k)\| \leq \alpha \|x(k)\|$, $\alpha < 1$ is imposed, which ensures the state is reduced at each sampling instance. This was first proposed by Polak and Yang [35].

3.1.5. Linear model predictive control

Linear MPC is a method of MPC which uses a linear predictive model. The three most common types of predictive model used in MPC are state-space, transfer function, and finite impulse response models. State-space models are chosen for the thesis project, as these are the most suitable choice for Multiple-Input Multiple-Output (MIMO) systems, and this literature review will focus on the use of state-space models accordingly.

In the design of the predictive model and optimiser, there is a trade-off between accuracy and computational efficiency. Therefore, it is important to design the model and optimiser to the required level of detail while maintaining time-steps on the desired time-scale. For the optimiser, this may involve

[2] This is one definition of stability in the context of control theory.

formulation of the cost function, design of the optimisation algorithm, design of the constraints, the prediction horizon, the control horizon, and the sampling time. For the predictive model, this can involve selection of the number of parameters, choice of linear or non-linear model, and modelling of disturbances.

Linear time invariant state-space model

The state-space model for a discrete-time linear time invariant (LTI) system is given by:

$$\begin{aligned}
\underbrace{\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ \vdots \\ x_n(k+1) \end{bmatrix}}_{\boldsymbol{x}_{k+1}} &= \underbrace{\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} x_1(k) \\ x_2(k) \\ \vdots \\ x_n(k) \end{bmatrix}}_{\boldsymbol{x}_k} + \underbrace{\begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,m} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \cdots & b_{n,m} \end{bmatrix}}_{\boldsymbol{B}} \underbrace{\begin{bmatrix} u_1(k) \\ u_2(k) \\ \vdots \\ u_m(k) \end{bmatrix}}_{\boldsymbol{u}_k} \\
\underbrace{\begin{bmatrix} y_1(k) \\ y_2(k) \\ \vdots \\ y_n(k) \end{bmatrix}}_{\boldsymbol{y}_k} &= \underbrace{\begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l,1} & c_{l,2} & \cdots & c_{l,n} \end{bmatrix}}_{\boldsymbol{C}} \underbrace{\begin{bmatrix} x_1(k) \\ x_2(k) \\ \vdots \\ x_n(k) \end{bmatrix}}_{\boldsymbol{x}_k} + \underbrace{\begin{bmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,m} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ d_{l,1} & d_{l,2} & \cdots & d_{l,m} \end{bmatrix}}_{\boldsymbol{D}} \underbrace{\begin{bmatrix} u_1(k) \\ u_2(k) \\ \vdots \\ u_m(k) \end{bmatrix}}_{\boldsymbol{u}_k}
\end{aligned} \tag{3.1}$$

or, in compact form:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k \\ \boldsymbol{y}_k &= \boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{D}\boldsymbol{u}_k \end{aligned} \tag{3.2}$$

Where k is the discrete time index (sampling instance); $\boldsymbol{x}_k \in \mathbb{R}^{n_x}$, $\boldsymbol{u}_k \in \mathbb{R}^{n_u}$, and $\boldsymbol{y}_k \in \mathbb{R}^{n_y}$ are the system state, control input, and system output vectors respectively; and $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ are the state matrix, input matrix, output matrix, and feed-forward matrices respectively.

The linear state-space model can be modified to include disturbances modelled as integrated white noise, $\boldsymbol{d}_k$, where $\boldsymbol{d}_{k+1} = \boldsymbol{d}_k + v_k$ and v_k is unknown and zero-mean. The linear state-space model with output disturbances is given by:

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k; \quad \boldsymbol{y}_k = \boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{D}\boldsymbol{u}_k + \boldsymbol{d}_k \tag{3.3}$$

with the augmented model given by:

$$z_{k+1} = \tilde{\mathbf{A}} z_k + \tilde{\mathbf{B}}\boldsymbol{u}_k; \quad \boldsymbol{y}_k = \tilde{\mathbf{C}} z_k + \boldsymbol{D}\boldsymbol{u}_k + v_k \tag{3.4}$$

where

$$z_{k+1} = \begin{bmatrix} \boldsymbol{x}_{k+1} \\ \boldsymbol{d}_{k+1} \end{bmatrix}; \quad \tilde{\mathbf{A}} = \begin{bmatrix} \boldsymbol{A} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{I} \end{bmatrix}; \quad \tilde{\mathbf{B}} = \begin{bmatrix} \boldsymbol{B} \\ \mathbf{0} \end{bmatrix}; \quad \tilde{\mathbf{C}} = \begin{bmatrix} \boldsymbol{C} & \boldsymbol{I} \end{bmatrix} \tag{3.5}$$

and an observer must provide estimates on $\boldsymbol{x}$ and $\boldsymbol{d}$.
Alternatively, the state-space model with state disturbances is given by:

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{F}\boldsymbol{d}_k; \quad \boldsymbol{y}_k = \boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{D}\boldsymbol{u}_k \tag{3.6}$$

with the augmented model given by:

$$z_{k+1} = \tilde{\mathbf{A}} z_k + \tilde{\mathbf{B}}\boldsymbol{u}_k; \quad \boldsymbol{y}_k = \tilde{\mathbf{C}} z_k + \boldsymbol{D}\boldsymbol{u}_k + v_k \tag{3.7}$$

where

$$z_{k+1} = \begin{bmatrix} \boldsymbol{x}_{k+1} \\ \boldsymbol{d}_{k+1} \end{bmatrix}; \quad \tilde{\mathbf{A}} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{F} \\ \mathbf{0} & \boldsymbol{I} \end{bmatrix}; \quad \tilde{\mathbf{B}} = \begin{bmatrix} \boldsymbol{B} \\ \mathbf{0} \end{bmatrix}; \quad \tilde{\mathbf{C}} = \begin{bmatrix} \boldsymbol{C} & \mathbf{0} \end{bmatrix} \tag{3.8}$$

Prediction for linear model predictive control

The general form prediction equation is given by:

$$\underset{\rightarrow k}{\boldsymbol{y}} = \boldsymbol{H}\Delta\underset{\rightarrow k-1}{\boldsymbol{u}} + \boldsymbol{P}\underset{\leftarrow k}{\boldsymbol{x}} \tag{3.9}$$

Where $\boldsymbol{H}$ is a Toeplitz matrix[3] of the system step response and $\boldsymbol{P}$ is a matrix dependent on the model parameters. Using the state-space model, the vector of future predictions is given by:

$$\underbrace{\begin{bmatrix} \boldsymbol{x}_{k+1} \\ \boldsymbol{x}_{k+2} \\ \boldsymbol{x}_{k+3} \\ \vdots \\ \boldsymbol{x}_{k+n_y} \end{bmatrix}}_{\underset{\rightarrow k}{\boldsymbol{x}}} = \underbrace{\begin{bmatrix} A \\ A^2 \\ A^3 \\ \vdots \\ A^{n_y} \end{bmatrix}}_{\boldsymbol{P}_{xx}} \boldsymbol{x}_k + \underbrace{\begin{bmatrix} B & 0 & 0 & \cdots \\ AB & B & 0 & \cdots \\ A^2B & AB & B & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ A^{n_y-1}B & A^{n_y-2}B & A^{n_y-3}B & \cdots \end{bmatrix}}_{\boldsymbol{H}_x} \underbrace{\begin{bmatrix} \boldsymbol{u}_k \\ \boldsymbol{u}_{k+1} \\ \boldsymbol{u}_{k+2} \\ \vdots \\ \boldsymbol{u}_{k+n_y-1} \end{bmatrix}}_{\underset{\rightarrow k-1}{\boldsymbol{u}}} \tag{3.10}$$

and

$$\underbrace{\begin{bmatrix} \boldsymbol{y}_{k+1} \\ \boldsymbol{y}_{k+2} \\ \boldsymbol{y}_{k+3} \\ \vdots \\ \boldsymbol{y}_{k+n_y} \end{bmatrix}}_{\underset{\rightarrow k}{\boldsymbol{y}}} = \underbrace{\begin{bmatrix} CA \\ CA^2 \\ CA^3 \\ \vdots \\ CA^{n_y} \end{bmatrix}}_{\boldsymbol{P}} \boldsymbol{x}_k + \underbrace{\begin{bmatrix} CB & 0 & 0 & \cdots \\ CAB & CB & 0 & \cdots \\ CA^2B & CAB & CB & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ CA^{n_y-1}B & CA^{n_y-2}B & CA^{n_y-3}B & \cdots \end{bmatrix}}_{\boldsymbol{H}} \underset{\rightarrow k-1}{\boldsymbol{u}} \tag{3.11}$$

or, in compact form:

$$\begin{aligned} \underset{\rightarrow k}{\boldsymbol{x}} &= \boldsymbol{P}_{xx}\boldsymbol{x}_k + \boldsymbol{H}_x\underset{\rightarrow k-1}{\boldsymbol{u}} \\ \underset{\rightarrow k}{\boldsymbol{y}} &= \boldsymbol{P}\boldsymbol{x}_k + \boldsymbol{H}\underset{\rightarrow k-1}{\boldsymbol{u}} \end{aligned} \tag{3.12}$$

The control law is calculated from the minimisation of a cost function. The cost function is typically formulated as either an l_1, l_2, or l_∞-norm of the tracking error and control input rate. The l_1-norm is the sum of all individual terms in a vector, the l_2-norm is the square root of the sum of all individual terms in a vector squared, and the l_∞-norm is the largest absolute value of the terms in a vector. Any of these norms can be used in the cost function, although the l_2-norm is the most common one to use.

Generalised Predictive Control Algorithm

One of the earliest popular MPC algorithms was the Generalised Predictive Control (GPC) algorithm, developed by Clarke, Mohtadi, and Tuffs [36, 37]. In the GPC algorithm, the control law is determined from the minimisation of an l^2-norm cost function over a finite horizon.

$$\begin{aligned} J &= \sum_{i=0}^{n_y} \|\boldsymbol{r}_{k+i} - \boldsymbol{y}_{k+i}\|_2^2 + \lambda \sum_{i=0}^{n_u-1} \|\Delta\boldsymbol{u}_{k+i}\|_2^2 \\ &= \sum_{i=0}^{n_y} \|\boldsymbol{e}_{k+i}\|_2^2 + \lambda \sum_{i=0}^{n_u-1} \|\Delta\boldsymbol{u}_{k+i}\|_2^2 \end{aligned} \tag{3.13}$$

where $\boldsymbol{r}$ is the reference system output, $\Delta\boldsymbol{u}$ is the change in control input, $\boldsymbol{e} = \boldsymbol{r} - \boldsymbol{y}$ is the tracking error, and the subscript 2 denotes the l_2-norm. This is applicable to SISO and MIMO systems, although an alternative formulation for MIMO systems is given by:

[3]A Toeplitz matrix is a matrix in which the descending diagonals are all constant.

$$\begin{aligned} J &= \sum_{i=0}^{n_y} \left\| \boldsymbol{W}_y (\boldsymbol{r}_{k+i} - \boldsymbol{y}_{k+i}) \right\|_2^2 + \lambda \sum_{i=0}^{n_u-1} \left\| \boldsymbol{W}_u (\Delta \boldsymbol{u}_{k+i}) \right\|_2^2 \\ &= \sum_{i=0}^{n_y} \left\| \boldsymbol{W}_y (\boldsymbol{e}_{k+i}) \right\|_2^2 + \lambda \sum_{i=0}^{n_u-1} \left\| \boldsymbol{W}_u (\Delta \boldsymbol{u}_{k+i}) \right\|_2^2 \end{aligned} \tag{3.14}$$

Where λ is a scalar weight and $\boldsymbol{W}_y$, and $\boldsymbol{W}_u$ are weighting matrices which can be tuned to get the desired response [4]. As mentioned previously, it is assumed that the control rates beyond the control horizon are zero, or $\Delta \boldsymbol{u}_{k+i|k} = 0$ for $i \leq n_u$. Another formulation which avoids the use of control input increments is:

$$J = \sum_{i=0}^{n_y} \|\boldsymbol{e}_{k+i}\|_2^2 + \lambda \sum_{i=0}^{n_u-1} \|\boldsymbol{u}_{k+i} - \boldsymbol{u}_{SS}\|_2^2 \tag{3.15}$$

where $\boldsymbol{u}_{SS}$ is the deviation from the steady-state control input. In compact form, the cost function from equation 3.13 is:

$$\begin{aligned} J &= \left\| \underset{\rightarrow}{\boldsymbol{r}} - \underset{\rightarrow}{\boldsymbol{y}} \right\|_2^2 + \lambda \left\| \Delta \underset{\rightarrow}{\boldsymbol{u}} \right\|_2^2 \\ & \left\| \underset{\rightarrow}{\boldsymbol{e}} \right\|_2^2 + \lambda \left\| \Delta \underset{\rightarrow}{\boldsymbol{u}} \right\|_2^2 \end{aligned} \tag{3.16}$$

The control law is calculated by determining an optimal future control rate input sequence $\Delta \underset{\rightarrow}{\boldsymbol{u}}$ which minimises the cost function:

$$\min_{\Delta \underset{\rightarrow}{\boldsymbol{u}}} \quad J = \left\| \underset{\rightarrow}{\boldsymbol{e}} \right\|_2^2 + \lambda \left\| \Delta \underset{\rightarrow}{\boldsymbol{u}} \right\|_2^2 \tag{3.17}$$

The control law is implemented as $\boldsymbol{u}_k = \boldsymbol{u}_{k-1} + \Delta \boldsymbol{u}_{0|k}$. A problem with the GPC algorithm is that stability is not guaranteed as the optimisation is performed over a finite horizon, and may not be stable afterwards. Therefore to ensure stability, methods such as those discussed in section 3.1.4 must be applied.

Generalised Predictive Control using State-Space Model

There are two formulations of the GPC algorithm for state-space models: with or without state augmentation. In the augmented state-space method, the augmented state-space model is defined as:

$$\begin{aligned} \begin{bmatrix} \boldsymbol{x}_{k+1} \\ \boldsymbol{u}_k \end{bmatrix} &= \underbrace{\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}}_{\hat{\mathbf{A}}} \underbrace{\begin{bmatrix} \boldsymbol{x}_k \\ \boldsymbol{u}_{k-1} \end{bmatrix}}_{\hat{\mathbf{x}}} + \underbrace{\begin{bmatrix} \boldsymbol{B} \\ \boldsymbol{I} \end{bmatrix}}_{\hat{\mathbf{B}}} \Delta \boldsymbol{u}_k \\ \boldsymbol{y}_k &= \underbrace{\begin{bmatrix} \boldsymbol{C} & \boldsymbol{D} \end{bmatrix}}_{\hat{\mathbf{C}}} \begin{bmatrix} \boldsymbol{x}_k \\ \boldsymbol{u}_{k-1} \end{bmatrix} + \boldsymbol{D} \Delta \boldsymbol{u}_k + \boldsymbol{d}_k \end{aligned} \tag{3.18}$$

Implementing the GPC algorithm following the same steps as before, the cost is given by:

[4] The weighting matrices are positive definite and diagonal.

$$\min_{\underset{\to}{u}} \quad J = \left\| \underset{\to}{r} - H\Delta \underset{\to}{u} - P\hat{\mathbf{x}}_k - Ld \right\|^2 + \lambda \left\| \Delta \underset{\to}{u} \right\|^2 \tag{3.19}$$

Where L is a vector of ones. The control law can be calculated to be:

$$\begin{aligned} \Delta u_k &= e_1^T (H^T H + \lambda I)^{-1} H^T [\underset{\to}{r} - [P, L] \begin{bmatrix} \hat{\mathbf{x}}_\mathbf{k} \\ d \end{bmatrix}] \\ &= P_r \underset{\to}{r} - \hat{\mathbf{K}} \begin{bmatrix} \hat{\mathbf{x}}_\mathbf{k} \\ d \end{bmatrix} \end{aligned} \tag{3.20}$$

Where $\hat{\mathbf{K}} = e_1^T (H^T H + \lambda I)^{-1} H^T [P, L]$, $P_r = e_1^T (H^T H + \lambda I)^{-1} H^T$, d is the disturbance estimate, $e_1^T = [I, \mathbf{0}, \ldots, \mathbf{0}]$, and $L = [I, I, \cdots]^T$.
Without state augmentation, the optimisation problem is given by:

$$\min_{\underset{\to}{u}} \quad J = [\underset{\to}{x} - x_{ss}]^T Q [\underset{\to}{x} - x_{ss}] + [\underset{\to}{u} - u_{ss}]^T R [\underset{\to}{u} - u_{ss}] \tag{3.21}$$

Where x_{ss} and u_{ss} are estimates of the steady-state values of the state and input vectors. This can be rearranged to:

$$\min_{\underset{\to}{u}} \quad J = [P_{xx} x + H_x u]^T Q [P_{xx} x + H_x \underset{\to}{u}] + \underset{\to}{u}^T R \underset{\to}{u} \tag{3.22}$$

The optimal control law is then given by:

$$\underset{\to}{u} = -[H_x^T Q H_x + R]^{-1} H_x^T Q P_{xx} x = -K x \tag{3.23}$$

The optimal control law is then given by:

$$u_k - u_{ss} = -e_1^T [H_x^T Q H_x + R]^{-1} H_x^T Q P_{xx} (x - x_{ss}) \tag{3.24}$$

Constrained Generalised Predictive Control Algorithm

Constraints may be applied to control inputs, control input rates, and outputs. In this paper, the underline notation is used to signify minimum constraints (e.g. $\underline{u}$) and the overline to signify maximum constraints (e.g. $\overline{u}$). The control input rate, control input, and output constraints at sampling instance k are given by:

$$\underline{\Delta u} \le \Delta u_k \le \overline{\Delta u}; \quad \underline{u} \le u_k \le \overline{u}; \quad \underline{y} \le y_k \le \overline{y} \tag{3.25}$$

The matrix of constraints from sampling instance k to the prediction horizon for the input rate, input, and output constraints are given by:

$$\underbrace{\begin{bmatrix} \underline{\Delta u} \\ \underline{\Delta u} \\ \vdots \\ \underline{\Delta u} \end{bmatrix}}_{\underline{\Delta U}} \le \begin{bmatrix} \Delta u_k \\ \Delta u_{k+1} \\ \vdots \\ \Delta u_{k+n_u-1} \end{bmatrix} \le \underbrace{\begin{bmatrix} \overline{\Delta u} \\ \overline{\Delta u} \\ \vdots \\ \overline{\Delta u} \end{bmatrix}}_{\overline{\Delta U}}; \quad \underbrace{\begin{bmatrix} \underline{u} \\ \underline{u} \\ \vdots \\ \underline{u} \end{bmatrix}}_{\underline{U}} \le \begin{bmatrix} u_k \\ u_{k+1} \\ \vdots \\ u_{k+n_u-1} \end{bmatrix} \le \underbrace{\begin{bmatrix} \overline{u} \\ \overline{u} \\ \vdots \\ \overline{u} \end{bmatrix}}_{\overline{U}}; \quad \underbrace{\begin{bmatrix} \underline{y} \\ \underline{y} \\ \vdots \\ \underline{y} \end{bmatrix}}_{\underline{Y}} \le \begin{bmatrix} \Delta y_k \\ \Delta y_{k+1} \\ \vdots \\ \Delta y_{k+n_y-1} \end{bmatrix} \le \underbrace{\begin{bmatrix} \overline{y} \\ \overline{y} \\ \vdots \\ \overline{y} \end{bmatrix}}_{\overline{Y}} \tag{3.26}$$

In compact form, these equations are given by:

$$\underline{\Delta U} \le \Delta \underset{\to}{u} \le \overline{\Delta U}; \quad \underline{U} \le \underset{\to}{u} \le \overline{U}; \quad \underline{Y} \le \underset{\to}{y} \le \overline{Y} \tag{3.27}$$

The constraints can be rearranged into a single set:

$$\boldsymbol{C}\Delta\underset{\rightarrow}{\boldsymbol{u}} - \boldsymbol{d}_k \le 0 \tag{3.28}$$

where

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{I} \\ -\boldsymbol{I} \\ \boldsymbol{C}_{I/\Delta} \\ -\boldsymbol{C}_{I/\Delta} \\ \boldsymbol{H} \\ -\boldsymbol{H} \end{bmatrix}; \quad \boldsymbol{d} = \begin{bmatrix} \overline{\Delta \boldsymbol{U}} \\ -\underline{\Delta \boldsymbol{U}} \\ \overline{\boldsymbol{U}} - \boldsymbol{L}\boldsymbol{u}_{k-1} \\ -\underline{\boldsymbol{U}} - \boldsymbol{L}\boldsymbol{u}_{k-1} \\ \overline{\boldsymbol{Y}} - \boldsymbol{Q}\Delta\underset{\leftarrow}{\boldsymbol{u}} - \boldsymbol{P}\underset{\leftarrow}{\boldsymbol{y}} \\ -\underline{\boldsymbol{Y}} - \boldsymbol{Q}\Delta\underset{\leftarrow}{\boldsymbol{u}} - \boldsymbol{P}\underset{\leftarrow}{\boldsymbol{y}} \end{bmatrix} \tag{3.29}$$

These can then be implemented into the optimisation problem.

In many situations, it may be infeasible to satisfy all desired constraints at once. In this case, the constraints can be reformulated into an optimisation problem, and a weighting matrix applied to the constraints to allow for soft and hard constraints:

$$\min_{\Delta\underset{\rightarrow}{\boldsymbol{u}}} \quad \boldsymbol{W}\left\|\boldsymbol{C}\Delta\underset{\rightarrow}{\boldsymbol{u}} - \boldsymbol{d}_k\right\|_\infty \tag{3.30}$$

This allows soft constraints to be relaxed in order to find a feasible solution. The weighting matrix $\boldsymbol{W}$ has values $\boldsymbol{W}_{i,j} = \infty$ for hard constraints and $\boldsymbol{W}_{i,j} = 0$ for satisfied constraints. The stability of a constrained controller cannot be guaranteed, even if the unconstrained controller is stable. If the prediction horizon and control horizon is large enough, however, the constrained controller performance should be similar to the unconstrained case.

3.1.6. Nonlinear model predictive control

Nonlinear MPC uses a nonlinear prediction model. This causes the optimisation problem to become non-convex and more challenging to solve from an optimality and stability viewpoint. Grüne and Pannek [38] give the nonlinear constant reference MPC algorithm as:

Algorithm 1: Basic nonlinear MPC algorithm for a constant reference $x^{\text{ref}} \equiv x_*$ at each sampling instance k

```
1 Measure the system state $x(k) \in \mathbb{X}$
2 Set $x_0 = x(k)$ and solve the optimal control problem:

      min                $J(x_0, u(\cdot)) = \sum_{i=0}^{n_y} l(x_u(i, x_0), u(i))$      (3.31a)
      with respect to    $u(\cdot) \in \mathbb{U}^{n_y}(x_0)$                              (3.31b)
      subject to         $x_u(0, x_0) = x_0,$                                              (3.31c)
                         $x_u(i+1, x_0) = f(x_u(i, x_0), u(i))$                            (3.31d)

  and denote the obtained optimal control sequence by $u^*(\cdot) \in \mathbb{U}^{n_y}(x_0)$
3 Implement the NMPC control law $\mu_{n_y}(x(n)) = u^*(0) \in \mathbb{U}$ at the next sampling instance.
```

where the stage cost, l, is the l_2-norm:

$$l(x, u) = \int_0^T L(\rho(t, 0, x, u), u(t))dt = \|x\|_2^2 + \lambda\|u\|_2^2 \tag{3.32}$$

An assumption made by this algorithm is that an optimal control sequence $u^*(\cdot)$ exists. This algorithm also assumes that the first term in the sequence, $u^*(0)$, is implemented while the others are discarded, though in reality the predicted control input sequence is often used to initialise the optimisation algorithm for faster convergence. Constraints can be implemented in the optimisation problem in the same way as usual.

The basic nonlinear MPC algorithm for a time varying reference is given by:

```
Algorithm 2: Basic nonlinear MPC algorithm for a time varying reference $x^{\text{ref}}$ at each sampling instance $k$.
1 Measure the system state $x(k) \in \mathbb{X}$
2 Set $x_0 = x(k)$, solve the optimal control problem:
```

$$\begin{aligned} &\min && J(k, x_0, u(\cdot)) = \sum_{i=0}^{n_y} l(k+i, x_u(i, x_0), u(i)) && (3.33a)\\ &\text{with respect to} && u(\cdot) \in \mathbb{U}^{n_y}(x_0) && (3.33b)\\ &\text{subject to} && x_u(0, x_0) = x_0, && (3.33c)\\ &&& x_u(i+1, x_0) = f(x_u(i, x_0), u(i)) && (3.33d) \end{aligned}$$

```
  and denote the obtained optimal control sequence by $u^*(\cdot) \in \mathbb{U}^{n_y}(x_0)$.
3 Implement the NMPC control law $\mu_{n_y}(n, x(n)) = u^*(0) \in \mathbb{U}$ at the next sampling instance.
```

The extended nonlinear MPC algorithm includes dependencies of f and $\mathbb{X}$ on k:

```
Algorithm 3: Extended nonlinear MPC algorithm for a constant reference $x^{\text{ref}}$ at each sampling instance $k$.
1 Measure the system state $x(k) \in \mathbb{X}$
2 Set $x_0 = x(k)$, solve the optimal control problem:
```

$$\begin{aligned} &\min && J(x_0, u(\cdot)) = \sum_{i=0}^{n_y} \omega_{n_y-i} l(x_u(i, x_0), u(i)) + F(x_u(n_y, x_0)) && (3.34a)\\ &\text{with respect to} && u(\cdot) \in \mathbb{U}^{n_y}(x_0) && (3.34b)\\ &\text{subject to} && x_u(0, x_0) = x_0, && (3.34c)\\ &&& x_u(i+1, x_0) = f(x_u(i, x_0), u(i)) && (3.34d) \end{aligned}$$

```
  and denote the obtained optimal control sequence by $u^*(\cdot) \in \mathbb{U}^{n_y}(x_0)$.
3 Implement the NMPC control law $\mu_{n_y}(n, x(n)) = u^*(0) \in \mathbb{U}$ at the next sampling instance.
```

The extended nonlinear MPC algorithm for a time varying reference is given by:

Algorithm 4: Extended nonlinear MPC algorithm for a time varying reference x^{ref} at each sampling instance k.

```
1 Measure the system state $x(k) \in \mathbb{X}$
2 Set $x_0 = x(k)$, solve the optimal control problem:
```

$$\begin{aligned}
&\min && J(x_0, u(\cdot)) = \sum_{i=0}^{n_y} \omega_{n_y - i} l(k+i, x_u(i, x_0), u(i)) + F(k+n_y, x_u(n_y, x_0)) && \text{(3.35a)}\\
&\text{with respect to} && u(\cdot) \in \mathbb{U}^{n_y}(x_0) && \text{(3.35b)}\\
&\text{subject to} && x_u(0, x_0) = x_0, && \text{(3.35c)}\\
& && x_u(i+1, x_0) = f(x_u(i, x_0), u(i)) && \text{(3.35d)}
\end{aligned}$$

```
    and denote the obtained optimal control sequence by $u^*(\cdot) \in \mathbb{U}^{n_y}(x_0)$.
3 Implement the NMPC control law $\mu_{n_y}(n, x(n)) = u^*(0) \in \mathbb{U}$ at the next sampling instance.
```

As can be seen, depending on the control task, an appropriate nonlinear MPC algorithm can be formulated.

3.1.7. Robust model predictive control

The basic MPC algorithm makes the assumption that the model Σ is exactly the same as the system Σ_0. This can result in sub-optimal performance for real systems subject to uncertainties. *Robustness* is the property of being able to maintain stability and achieve control objectives for all permutations of a bounded uncertainty [39]. Robust MPC is a method of MPC in which an optimal sequence of control inputs is calculated while ensuring that constraints will be satisfied for all future permutations of bounded uncertainties. The robust MPC state-space model includes an unmeasured noise $\boldsymbol{w}_k$ in the model:

$$\Sigma : \begin{cases} \boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{H}\boldsymbol{w}_k \\ \boldsymbol{y}_k = \boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{D}\boldsymbol{u}_k + \boldsymbol{K}\boldsymbol{w}_k \end{cases} \tag{3.36}$$

where uncertainty sets $\boldsymbol{w}_k \in \mathcal{W}$ and $\Sigma_0 \in \mathcal{S}$ are the input set and covered[5] set respectively. The cost function requires one instance of the model Σ and disturbance $\boldsymbol{w}_k$. To formulate a cost function for a model which includes disturbances, the cost function can then be formed in two ways. With the first method, a nominal model $\hat{\Sigma}$ and nominal disturbance $\hat{\mathbf{w}}_k$ are defined. The *nominal cost* is then given by:

$$\min_{\boldsymbol{U}} \quad \hat{J}(\boldsymbol{U}, \boldsymbol{x}_k, \hat{\Sigma}, \hat{\mathbf{w}}_k) \tag{3.37}$$

With the second method, the cost function is formulated as a min-max problem:

$$\min_{\boldsymbol{U}} \quad \max_{\substack{\Sigma \in \mathcal{S} \\ \{\boldsymbol{w}_{k+i}\}_{i=0}^{n_y - 1} \subseteq \mathcal{W}}} \quad J(\boldsymbol{U}, \boldsymbol{x}_k, \Sigma, \hat{\mathbf{w}}_k) \tag{3.38}$$

This optimisation attempts to minimise the maximum cost obtained from all possible permutations of the uncertainty variables. Compared to the nominal optimisation, the min-max optimisation is more computationally intensive and can be more conservative.

Robust MPC is useful for the control of safety-critical systems subject to disturbances with known bounds, as it will find a sequence of control inputs which guarantees constraints will be met if there is one. A disadvantage of robust MPC is that it can be overly conservative and may not lead to the optimal control performance for systems.

[5] Set of LTI systems.

3.1.8. Hybrid model predictive control

Hybrid MPC is focused on the design of MPC controllers for hybrid systems, which are systems that have both continuous and discrete dynamics. This can be caused by dynamics such as switching, logical conditions, and discrete variables. Some examples of hybrid systems include and systems which combine digital and analogue components, such as embedded systems and aviation systems.

Hybrid Systems Models

The general hybrid system state-space model is be given by:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= f(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k) \\ \boldsymbol{y} &= g(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k) \end{aligned} \tag{3.39}$$

Where $f(\cdot)$ and $g(\cdot)$ are nonlinear noncontinuous functions. The two most commonly used types of hybrid systems models are piecewise affine (PWA) models and mixed logical and dynamical (MLD) models [40]. The PWA model is given by:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{A}^i \boldsymbol{x}_k + \boldsymbol{B}^i \boldsymbol{u}_k + \boldsymbol{f}^i \\ \boldsymbol{y}_k &= \boldsymbol{C}^i \boldsymbol{x}_k + \boldsymbol{g}^i \end{aligned} \quad \text{for} \begin{bmatrix} \boldsymbol{x}_k \\ \boldsymbol{u}_k \end{bmatrix} \in \mathcal{X}_i \tag{3.40}$$

where $\mathcal{X}_i$ is a partition of the state space given by:

$$\mathcal{X}_i \triangleq \left\{ \begin{bmatrix} \boldsymbol{x}_k \\ \boldsymbol{u}_k \end{bmatrix} | \boldsymbol{R}^i \begin{bmatrix} \boldsymbol{x}_k \\ \boldsymbol{u}_k \end{bmatrix} \le \boldsymbol{r}^i \right\} \tag{3.41}$$

Each subsystem $\mathbb{S}^i$ is a component in the PWA system defined by $(\boldsymbol{A}^i, \boldsymbol{B}^i, \boldsymbol{C}^i, \boldsymbol{f}^i, \boldsymbol{g}^i, \boldsymbol{R}^i, \boldsymbol{r}^i)$ for $i \in \{1, 2, ..., S\}$. $\boldsymbol{C}^i \in \mathbb{R}^{r \times n}$, $\boldsymbol{R}^i \in \mathbb{R}^{p_i \times (n+m)}$, $\boldsymbol{f}^i$, $\boldsymbol{g}^i$, and $\boldsymbol{r}^i$ are constant vectors, n is the number of states, m is the number of inputs, r is the number of outputs, and p_i is the number of hyperplanes in $\mathcal{X}_i$.

The MLD model integrates logical dynamics by modelling them as constraints. This model is given by:

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}_1 \boldsymbol{u}_k + \boldsymbol{B}_2 \boldsymbol{\delta}_k + \boldsymbol{B}_3 \boldsymbol{\zeta}_k \tag{3.42a}$$
$$\boldsymbol{y}_k = \boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{D}_1 \boldsymbol{u}_k + \boldsymbol{D}_2 \boldsymbol{\delta}_k + \boldsymbol{D}_3 \boldsymbol{\zeta}_k \tag{3.42b}$$
$$\boldsymbol{E}_1 \boldsymbol{x}_k + \boldsymbol{E}_2 \boldsymbol{u}_k + \boldsymbol{E}_3 \boldsymbol{\delta}_k + \boldsymbol{E}_4 \boldsymbol{\zeta}_k \le \boldsymbol{g} \tag{3.42c}$$

where $\boldsymbol{x}_k = [\boldsymbol{x}_c^T(k)\ \boldsymbol{x}_b^T(k)] \in \mathbb{R}^{n_c} \times \{0,1\}^{n_b}$ is the state vector (with continuous part $\boldsymbol{x}_c^T(k)$ and discrete part $\boldsymbol{x}_b^T(k)$), $\boldsymbol{u}_k = [\boldsymbol{u}_c^T(k)\ \boldsymbol{u}_b^T(k)] \in \mathbb{R}^{m_c} \times \{0,1\}^{m_b}$ is the input vector, $\boldsymbol{y}_k = [\boldsymbol{y}_c^T(k)\ \boldsymbol{y}_b^T(k)] \in \mathbb{R}^{p_c} \times \{0,1\}^{p_b}$ is the output vector, and $\boldsymbol{\zeta}_k \in \mathbb{R}^{r_c}$ and $\boldsymbol{\delta}_k \in \{0,1\}^{m_b}$ are the binary and continuous auxiliary variables respectively, and are defined by equation 3.42c [6].

Hybrid model predictive control algorithm

Stability of hybrid MPC controllers can be ensured with a terminal cost and constraint set method [41]. The cost function is given as:

$$J(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v}) = \sum_{i=0}^{n_y} \|\boldsymbol{y}_i - \boldsymbol{y}_{\text{ref}}\|_Q^2 + \|\boldsymbol{u}_i - \boldsymbol{u}_{\text{ref}}\|_R^2 + \|\boldsymbol{v}_i - \boldsymbol{v}_{\text{ref}}\|_T^2 + J_f(\boldsymbol{x}_{n_i} - \boldsymbol{x}_{\text{ref}}) \tag{3.43}$$

where $J_f(\boldsymbol{x})$ is the terminal cost function, which is added on for stability. This gives the following optimisation problem:

[6] Auxiliary variables are introduced while transforming logic relations into the model.

$$\begin{aligned} \min_{\boldsymbol{u},\boldsymbol{v}} \quad & J(\boldsymbol{x},\boldsymbol{u},\boldsymbol{v}) \\ \text{Subject to} \quad & \boldsymbol{x}(i+1) = f(\boldsymbol{x}_i,\boldsymbol{u}_i,\boldsymbol{v}_i) \\ & \boldsymbol{y}_i = g(\boldsymbol{x}_i,\boldsymbol{u}_i,\boldsymbol{v}_i) \\ & h(\boldsymbol{x}_i,\boldsymbol{u}_i,\boldsymbol{v}_i) \leq 0 \\ & [\boldsymbol{x}_i^T,\boldsymbol{u}_i^T]^T \in Z \\ & \boldsymbol{x}(n_y) \in X_f \end{aligned} \tag{3.44}$$

where X_f is the terminal region. From the optimisation problem, the control law can be established as usual. The resulting optimisation problem contains a set of linear constraints and discrete integer decision variables.

3.1.9. Explicit model predictive control

Explicit MPC is focused on solving the MPC optimisation problem offline. This involves partitioning the state-space into a set of polyhedral regions by defining intervals across the range of the various system states. The optimisation problem is solved for each combination of system states using multiparametric programming and the solutions are implemented in a lookup table, reducing the control law to the piecewise linear function:

$$\boldsymbol{u}(\boldsymbol{x}) = \begin{cases} F_1\boldsymbol{x}+g_1 \text{ if } & H_1\boldsymbol{x} \leq k_1 \\ \vdots & \vdots \\ F_M\boldsymbol{x}+g_M \text{ if } & H_M\boldsymbol{x} \leq k_M \end{cases}$$

The online computation is then reduced to determining the polyhedral region associated with the current system state and applying the associated controller input.

For some systems, explicit MPC can be implemented to have much faster rates than traditional MPC methods, making it particularly useful in systems with fast dynamics. Additionally, software certification is easier for this type of controller as the behaviour of the controller is explicitly defined, which is important for safety critical systems.

The drawbacks of explicit MPC are that the number of control laws scales exponentially with the number of parameters (states and reference signals), the number of constraints (which is dependent on the control horizon n_u and output horizon n_y), and so on; meaning that for certain systems it can become too large and time-consuming to calculate and implement in a look-up table. In [42], Alessio and Bemporad state that the typical size of problem that explicit MPC is suitable for has somewhere between $1-2$ inputs, $5-10$ states, and an output horizon ≤ 4.

Several approaches have been developed to reduce the number of computations required in explicit MPC. The most popular approach is suboptimal multiparametric quadratic programming (mp-QP), where approximate solutions to the mp-QP problem can be determined, and a trade-off is made between optimality and the number of polyhedral regions. This was first proposed by Bemporad and Filippi [43], who calculate suboptimal mp-QP solutions by relaxing the Karush-Kuhn-Tucker optimality conditions. In [44], Johansen and Grancharova propose an alternative method in which an orthogonal search-tree structure is imposed on the polyhedral region. In [45], Bemporad and Filippi propose a recursive algorithm which approximates the cost function.

3.1.10. Stochastic model predictive control

As detailed in section 3.1.7, robust MPC requires that constraints are satisfied for all possible values of uncertain parameters. This results in robust MPC treating all values of an uncertain parameter with equal importance, even if they have very different likelihoods. This is not always desirable, as in reality, the system may contain stochastic uncertainties with a known or identifiable probability distribution. Stochastic MPC is an MPC method which attempts to account for uncertainty distributions in the system. The advantages of this are that it can result in more optimal control performance as it is less conservative than robust MPC. Stochastic uncertainties can be incorporated into the MPC model in two ways:

- Stochastic variables are implemented in the cost function and the optimisation problem becomes a minimisation of the *expected* cost given the probability distributions of the stochastic variables.
- Stochastic variables are implemented in the constraints, requiring that the constraints are satisfied a certain proportion of the time. These are known as *joint chance constraints*.

In [46], Kouvaritakis and Cannon propose a stochastic MPC controller for industrial wind turbines. The goal of supervisory controllers for industrial wind turbines is to maximise service life by controlling the nacelle rotation to minimise the accumulation of fatigue damage in the tower. In order to maximise electrical output, these constraints may be allowed to be violated a certain percentage of times due to changes in wind speed. By modelling the wind speed with probability distributions, probabilistic constraints can be applied on the system to get the desired control behaviour. Stochastic MPC controllers have also been applied in literature to building climate control [47], telecommunications network traffic control [48], and other uses.

The general stochastic MPC state-space model is given as:

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k) \tag{3.45}$$

$$\boldsymbol{y}_k = h(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k) \tag{3.46}$$

where $\boldsymbol{w}_k \in \mathbb{R}^{n_w}$ and $\boldsymbol{v}_k \in \mathbb{R}^{n_v}$ are the disturbance and measurement noise with probability distributions $\boldsymbol{P}_w$ and $\boldsymbol{P}_v$ respectively. The cost function is defined as:

$$J(\boldsymbol{x}_k, \boldsymbol{u}_k) = \boldsymbol{E}_{x_k}\left[\sum_{i=0}^{n_y} J_c(\hat{\mathbf{x}}_i, \boldsymbol{u}_i) + J_f(\hat{\mathbf{x}}_{n_y})\right] \tag{3.47}$$

where J_c and J_f are the cost-per-stage function and the terminal cost function respectively. $\hat{\mathbf{x}}_i$ is the predicted state vector at time step i with the initial state vector $\hat{\mathbf{x}}_0 = \boldsymbol{x}_k$. The general form of *joint chance constraints* is given by:

$$\Pr_{x_k}[g_j(\hat{\mathbf{y}}_i) \leq 0, \text{ for all } j = 1, \dots, s] \geq \beta, \text{ for all } i = 1, \dots, n_y \tag{3.48}$$

where s is the number of inequality constraints and $\beta \in (0, 1)$ is the minimum probability of constraint satisfaction. Equation 3.48 can be extended to include multiple joint chance constraints. The constrained stochastic MPC cost function is given by:

$$\begin{aligned} & J^*(\boldsymbol{x}_k) = \min_{\boldsymbol{u}} J(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ \text{Subject to} \quad & \hat{\mathbf{x}}_{i+1} = f(\hat{\mathbf{x}}_i, \boldsymbol{u}_i, \boldsymbol{w}_i) \\ & \hat{\mathbf{y}}_i = h(\hat{\mathbf{x}}_i, \boldsymbol{u}_i) \\ & \boldsymbol{u}_i(\cdot) \in \mathbb{U} \\ & Pr_{x_k}[g_j(\hat{\mathbf{y}}_i) \leq 0, \text{ for all } j = 1, \dots, s] \geq \beta \\ & \boldsymbol{w}_i \sim \boldsymbol{P}_w \\ & \hat{\mathbf{x}}_0 = \boldsymbol{x}_k \end{aligned} \tag{3.49}$$

This applies to linear and nonlinear prediction models. Alternatively, a linear prediction model can be used. For the case of independent, identically distributed stochastic variables, the linear prediction model is given by:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{D}\boldsymbol{w}_k \\ \boldsymbol{y}_k &= \boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{F}\boldsymbol{v}_k \end{aligned} \tag{3.50}$$

For the case of multiplicative stochastic variables, the linear prediction model is given by:

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k\boldsymbol{B}\boldsymbol{u}_k + \sum_{j=1}^{q}(\bar{\mathbf{A}}_j\boldsymbol{x}_k + \bar{\mathbf{B}}_j\boldsymbol{u}_k)\boldsymbol{w}_{k,j} \tag{3.51}$$

where $w_{k,j}$ is a vector of stochastic variables. Various formulations of the optimiser exist for stochastic MPC. The stochastic tube method is used for prediction models with additive and bounded disturbances using the cost function:

$$J_\infty(\boldsymbol{x}_k, \boldsymbol{u}_k) = \boldsymbol{E}_{x_k}\left[\sum_{i=0}^{\infty} \|\hat{\mathbf{x}}_i\|_Q^2 + \|\boldsymbol{u}_i\|_R^2\right] \tag{3.52}$$

The state vector is given by $\boldsymbol{x}_k = \boldsymbol{z}_k + \boldsymbol{e}_k$, where $\boldsymbol{z}_k$ is the deterministic part and $\boldsymbol{e}_k$ is the random part of the state vector.

The auto-regressive moving average state-space model for a stochastic predictive model with additive disturbances is:

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}_k\boldsymbol{x}_k + \boldsymbol{B}_k\boldsymbol{u}_k + \boldsymbol{D}\boldsymbol{w}_k \tag{3.53}$$

where $\boldsymbol{w}_k \in \mathbb{R}^{n_w}$ is the disturbance input. In [46], it is assumed that $\boldsymbol{A}_k$, $\boldsymbol{B}_k$, and $\boldsymbol{w}_k$ are given by:

$$(\boldsymbol{A}_k, \boldsymbol{B}_k, \boldsymbol{w}_k) = (\boldsymbol{A}^{(0)}, \boldsymbol{B}^{(0)}0) + \sum_{j=1}^{p} (\boldsymbol{A}^{(j)}, \boldsymbol{B}^{(j)}, \boldsymbol{w}^{(j)})q_k^{(j)} \tag{3.54}$$

where $q_k^{(j)}$ is a random scalar with a known probability distribution, with $\boldsymbol{q} = (q_k^{(1)}, \dots, q_k^{(\rho)})$ and $\mathbb{E}(\boldsymbol{q}_k) = 0$ and $\mathbb{E}(\boldsymbol{q}_k\boldsymbol{q}_k^T) = I$. The l^2-norm cost function is given by:

$$\hat{J}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{q}_k) = \sum_{i=0}^{n_y}(\|\boldsymbol{x}_i\|_{\boldsymbol{Q}}^2 + \|\boldsymbol{u}_i\|_{\boldsymbol{R}}^2) + \left\|\boldsymbol{x}_{n_y|k}\right\|_{\boldsymbol{W}_T}^2 \tag{3.55}$$

where $\boldsymbol{Q}$ and $\boldsymbol{R}$ are the cost function weighting matrices and $\boldsymbol{W}_T$ is the terminal cost weighting matrix. A deterministic cost, $J(\boldsymbol{x}_k, \boldsymbol{u}_k)$, can be determined from the stochastic cost, $\hat{J}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{q}_k)$, based on assumptions made about $\boldsymbol{q}_k$. The nominal cost is given by:

$$J(\boldsymbol{x}_k, \boldsymbol{u}_k) = \hat{J}(\boldsymbol{x}_k, \boldsymbol{u}_k, 0) \tag{3.56}$$

The worst-case cost is given by:

$$J(\boldsymbol{x}_k, \boldsymbol{u}_k) = \max_{q_k \in \mathcal{Q}\times\cdots\times\mathcal{Q}} \hat{J}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{q}_k) \tag{3.57}$$

where the stochastic uncertainty has known bounds $q_k \in \mathcal{Q}$. The expected cost is given by:

$$\begin{aligned} J(\boldsymbol{x}_k, \boldsymbol{u}_k) &= \mathbb{E}_k\left(\hat{J}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{q}_k)\right) \\ &= \sum_{i=0}^{n_y} \mathbb{E}_k\left(\left\|\boldsymbol{x}_{i|k}\right\|_{\boldsymbol{Q}}^2 + \left\|\boldsymbol{u}_{i|k}\right\|_{\boldsymbol{R}}^2\right) + \mathbb{E}_k\left(\left\|\boldsymbol{x}_{n_y}\right\|_{\boldsymbol{W}_T}^2\right) \end{aligned} \tag{3.58}$$

where $\mathbb{E}_k(\cdot)$ is the expectation at sampling instance k. The dual-mode prediction algorithm is another stochastic MPC method used to predict state and control matrices over an infinite prediction horizon. This is useful for ensuring stability.

Stochastic constraints can be formulated in several ways. In terms of expected values, constraints are given by:

$$\mathbb{E}_k(\boldsymbol{F}\boldsymbol{x}_i + \boldsymbol{G}\boldsymbol{u}_i) \le \mathbf{1}; \quad i = 1, \dots, n_y \tag{3.59}$$

In terms of probabilities, constraints are given by:

$$\text{Pr}_k(\boldsymbol{F}\boldsymbol{x}_i + \boldsymbol{G}\boldsymbol{u}_i \le \mathbf{1}, i = 1, \dots, n_y) \ge p \tag{3.60}$$

where p is the minimum probability of the constraint being fulfilled and $\text{Pr}_k(\mathcal{A})$ is the probability of event $\mathcal{A}$ at sampling instance k. This also allows integration of robust constraints by setting $p = 1$.

Depending on the required controller performance, other methods of stochastic MPC can be applied. The dual-mode method is used to ensure stability, with the cost function given by:

$$J(\boldsymbol{x}_k, \boldsymbol{c}_k) = \sum_{i=0}^{\infty} \mathbb{E}_k(\|\boldsymbol{x}_{i|k}\|_Q^2 + \|\boldsymbol{u}_{i|k}\|_R^2 - l_{ss}) \tag{3.61}$$

where $\boldsymbol{c}_k = (c_0, \dots, c_{n_y})$. The steady state cost, l_{ss}, is given by:

$$l_{ss} = \lim_{i\to\infty} \mathbb{E}_k(\|\boldsymbol{x}_{i|k}\|_Q^2 + \|\boldsymbol{u}_{i|k}\|_R^2 - l_{ss}) \tag{3.62}$$

The gain K is related to the future control input sequence by $\boldsymbol{u}_{i|k} = K\boldsymbol{x}_{i|k} + \boldsymbol{c}_{i|k}$, $i = 0, 1, \dots$, where the decision variable matrix, $\boldsymbol{c}_k \in \mathbb{R}^{N_{n_u}}$ is given by $\boldsymbol{c}_k = (c_{0|k}, \dots, c_{N-1|k})$.

Another formulation of the stochastic MPC cost function is the mean-variance predicted cost.

3.1.11. State-of-the-art

The main methods of MPC have been reviewed in the above sections. This section will review the relevant state-of-the-art research that these methods have been applied in.

In [49], Xi, Li, and Lin review the ongoing challenges of MPC, in which they identify the major ongoing focuses of research in MPC to be *development of improved optimisation algorithms*, *focus on practical applications*, *focus on nonlinear and stochastic MPC*, and *extending MPC to other fields*.

In the context of improved optimisation algorithms, research is focused on *structural* improvements, *strategic* improvements, and *algorithmic* improvements. Research in the structural level is focused on applying MPC to large-scale systems by using hierarchical and distributed control structures. The challenges include large numbers of variables and complex models, which make it infeasible to solve a global optimisation problem. Solutions to this problem have used *multi-layer hierarchical structures* with either the same or different models on each layer or *distributed structures* in which the optimisation problem is decomposed into multiple local optimisation problems. The main challenges of distributed MPC methods includes subsystem coupling, communication between subsystems, and stability. The strategic level focuses on strategic improvements to MPC controllers. One such example is the *"off-line design and online synthesis"* strategy, where some of the online computations are instead moved to off-line to reduce computational burden during operation. Various explicit MPC methods use this strategy, as discussed in section 3.1.9. The algorithmic level is focused on increasing the speed of optimisation algorithms, either though improvements or suitable simplifications to the algorithm.

Additionally, Xi, Li, and Lin identify several problems with MPC theory. Firstly, the *effectiveness* of MPC algorithms is mainly limited by the computational burden of solving the online optimisation problems. Secondly, they claim there is a gap between *academics* and practical applications, where there is a lack of academic focus on MPC design and algorithms with regard to performance and computational limitations. Thirdly, they claim there is a limitation on the *usability* of current MPC algorithms which are based on general description and general solving methods of constrained optimisation control problems.

Aerospace

A significant field of research has investigated the applications of MPC in aerospace. In [50], Eren et al. review this field of research, where they identify the technical challenges of MPC with regards to aerospace applications to be *adaptive MPC*, *hybrid MPC*, *networked MPC*, *fault-tolerant MPC*, *MPC for time-delayed systems*, *MPC for agile systems*, and *MPC subject to uncertainties*.

Adaptive MPC methods are used for the control of systems in which the model is subject to uncertainties. In [51], Chowdhary et al. develop a concurrent learning adaptive-optimal controller, in which the model is learned online and the model is automatically switched to once the algorithm determines that the model is sufficiently tuned. The MPC method is applied to the trajectory tracking of a fighter aircraft in the presence of uncertain wing rock dynamics to demonstrate the ability to control systems with fast dynamics.

The challenges presented by hybrid MPC are that the optimisation problem can quickly become too complex to solve in real-time, creating the requirement for specialised optimisation algorithms.

Networked MPC is necessary for the collaborative control of multiple aerospace systems, such as formations and cooperating UAVs. The challenges in aerospace are that the complexity of the optimisation problem increases exponentially with the number of agents and must operate with limited information of other agents. Therefore, there is a need for MPC methods using decentralised and distributed network architectures which can guarantee the required controller properties.

A number of papers have applied nonlinear MPC to the control of aerospace vehicles for various tasks, such as [52–56], which apply nonlinear MPC methods for spacecraft attitude control, formation flight, and trajectory control.

Robust MPC has been applied extensively to aerospace tasks. In [57], Alexis et al. apply a robust MPC controller for the trajectory tracking of a quadrotor UAV subject to wind disturbances and collision avoidance constraints. In [58], Shekhar, Kearney, and Shames apply a variable-horizon robust MPC method for trajectory tracking with waypoints of a fixed-wing UAV subject to wind disturbances. In [59], Alexis et al. use an MPC controller to perform a path-planning and physical interaction task, applied to a quadrotor for aerial contact-based inspection. In [60], Richards and How use robust variable-horizon MPC for a spacecraft rendezvous problem in which robustness is guaranteed through the use of constraint tightening. In [61], Kim, Shim, and Sastry use robust MPC for the control of a helicopter UAV.

Research in stochastic MPC applications in aerospace is more limited, but some studies have attempted to do this. In [62], Blackmore et al. develop a stochastic model predictive control method where the uncertain system state is approximated by a finite number of particles which are used to reformulate the control problem to a deterministic one. This method is applied to the control of an aircraft subject to turbulence, where it is found to solve in less time than other stochastic MPC methods. In [63], Lyons, Calliess, and Hanebeck use stochastic MPC for a multi-agent system with chance constraints on the probability of collisions between agents.

In [64], Hegrenæs, Gravdahl, and Tøndel use explicit MPC for spacecraft attitude control using a constrained linear model and a short prediction horizon, $n_y = 2$.

In [65], Borrelli, Keviczky, and Balas propose a decentralised MPC architecture for the formation flight of a multi-agent quadrotor system. The scheme ensures collision avoidance through calculation of a collision avoidance manoeuvre which is executed if necessary. In [66], Richards and How implement robust decentralised MPC for the control of a multi-UAV path-planning task using an MILP model [66]. In [67], Shim and Kim present a decentralised nonlinear MPC method for the trajectory tracking of multiple helicopter UAVs subject to input and state constraints.

Deori et al. used linear MPC for the control of a quadrotor UAV [68]. More recently, Wang et al. applied nonlinear MPC for the control of a quadrotor UAV using a more accurate model [69]. Baca et al. used MPC for the broader application of trajectory tracking and disturbance rejection for multiple quadrotor UAVs [70]. They demonstrated this system experimentally for the formation flight of three quadrotors tracking a sine trajectory with a receding horizon of 2.2s. Havez et al. applied MPC to control multiple UAVs for the problem of dynamic encirclement [71]. A nonlinear MPC controller is used in simulation for a nonlinear UAV system, and a combination of linear MPC and feedback linearisation is used for real-time implementation of the control method. In [72], Ji et al. demonstrate MPC for the path tracking of a collision avoidance manoeuvre for an autonomous vehicle, using a nonlinear vehicle dynamics model and Hildreth's quadratic programming procedure in the optimiser [72]. Previous path tracking methods had relied on fuzzy logic, sliding-mode, or robust control, which all neglected nonlinear characteristics of vehicles. In [73], Di Cairano, Park, and Kolmanovsky demonstrate MPC for rendezvous docking with a tumbling platform.

3.2. System Network Structures

The network structure used in a system of autonomous agents determines how information is communicated between agents. In the context of networks, *"node"* refers to an agent and a *"connection"* refers to exchange of information between the two nodes. There are three main types of network structure:

- *Centralised* - one node is connected to all other nodes.
- *Decentralised* - connections are spread evenly across all nodes, with each node typically having multiple connections.

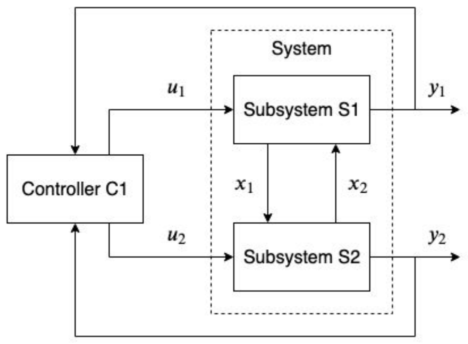


Figure 3.3: Example centralised control structure for a two-input two-output system. One controller is used for the control of both systems.

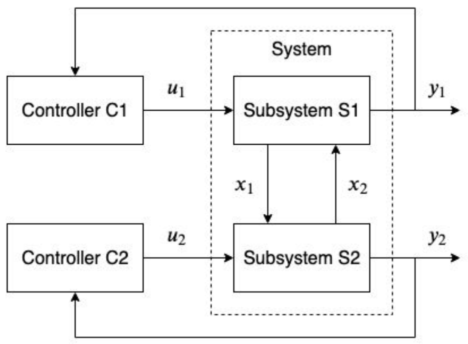


Figure 3.4: Example decentralised control structure for a two-input two-output system [74]

- *Distributed* - the network is divided into sub-groups of centralised nodes, with all sub-groups connected.

3.2.1. Centralised control systems

A centralised control structure uses one controller for the control of all parts of the system. This requires one node to process all of the information in the system, allowing the controller to act with perfect information but placing a high load on it. Additionally, connections are required between all nodes and the centralised node, whereas it is assumed that agents in the proposed system will have limited and sporadic communication with neighbours, which may only be a few at a time or even none at all.

3.2.2. Decentralised control systems

In decentralised networks, each node operates independently of the control input and outputs of other nodes, but is influenced by the system states. The controllers therefore operate independently, using only local information. This architecture is useful for simplifying the control of large-scale systems, requiring controllers to to be designed for smaller, less complex subsystems. This type of architecture assumes little or no coupling between the variables of one subsystem and another, and larger coupling can result in instability or degraded performance of the system as a whole. Figure 3.4 shows an example of a decentralised control architecture for a system which is divided into two subsystems (nodes) with one control input and output variable each.

3.2.3. Distributed Control System

In distributed control networks, there is an exchange of information between controllers. In the context of distributed MPC controllers, this may be in the form of measured states and control inputs or in the form of predicted states and control inputs. The controllers then include information received from other controllers in their predictions. Figure 3.5 shows an example of a distributed control architecture for a two-input, two-output system, where the dashed lines represent communication of information from the local controller. Distributed control architectures can be further sub-classified based on the

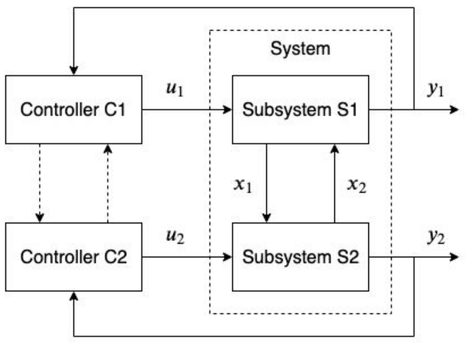


Figure 3.5: Example distributed control structure for a two-input two-output system [74]

structure:

- *Fully-connected algorithms* - nodes communicate information to all other controllers.
- *Partially-connected algorithms* - nodes communicate information to some nearby controllers.

The protocols for information communication between controllers can also be classified:

- *Non-iterative algorithms* - controllers exchange information once each sampling instance.
- *Iterative algorithms* - local controllers exchange information multiple times each sampling instance.

Iterative algorithms typically have higher performance, as controllers can converge to an agreed optimal solution while non-iterative algorithms use outdated information from nearby agents. However, iterative algorithms take longer to reach a solution as information must be communicated and the optimisation run multiple times at each sampling instance. Algorithms can also be classified based on the cost function:

- *Independent algorithms* - controllers optimise a local cost function.
- *Cooperating algorithms* - controllers optimise a global cost function.

The controllers may aim to optimise either a local or global cost function. This local cost function is based purely on the local agent and is easier to calculate, while the global cost function accounts for interaction between agents and considers the performance of the entire system.

3.2.4. Coordination of autonomous agents

Some MPC methods have been developed for systems with multiple autonomous agents which are entirely independent of each other. This can be achieved by attempting to minimise a global cost function or dividing the network into multiple sub-networks and minimising a local cost function for each sub-network, with suitable exchange of information between sub-networks.

In [75], Dunbar and Murray develop a distributed MPC method for the control of multiple agents with decoupled dynamics and constraints, where agents have nonlinear dynamics. A cost function is designed specifically for multiple vehicle formation stabilisation, which couples the state vectors of agents. This cost function is decomposed and implemented in each subsystem, which make predictions about the states of neighbouring agents over the prediction horizon to use in the local cost function. Each subsystem optimisation problem has a *compatibility constraint*, which helps to keep the difference between the predicted and actual state vector trajectories of other agents below a defined threshold. This constraint is key to ensuring stability of the system. Dunbar and Murray note that this system is not technically decentralised, since the optimisation is synchronised across the entire system, requiring a centralised timing mechanism in the system. In [76], Franco et al. consider the same control problem, but propose a solution using delayed information exchange between agents. The future developments which Franco et al. highlight as necessary for this research are for developing robust methods and developing methods which can handle disturbances in the communication between agents.

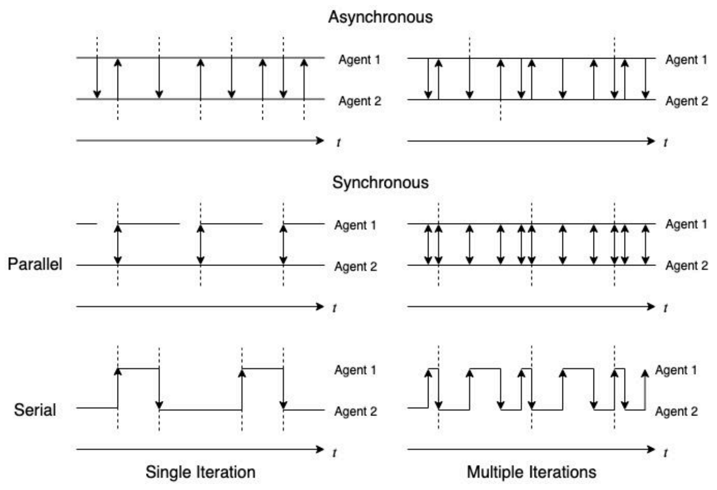


Figure 3.6: Communication protocols between two agents. The dotted lines represent an instance in which the control input is implemented, and arrows represent exchange of information.

An alternative task for multi-agent control systems in the *consensus problem*, where agents must negotiate the optimal control policy. This is applicable to flocking, rendezvous, and formation control problems; all of which are useful tasks for autonomous UAV systems. In [77], Johansson et al. propose an MPC method for a team of cooperating agents that guarantees consensus in a fixed amount of time. In [78–80], Ferrari-Trecate et al. propose discrete distributed MPC control schemes for the consensus problem of autonomous agents modelled by single or double integrator dynamics and prove that consensus is reached.

3.2.5. Communication Protocols

Communication protocols define the way in which information is exchanged between agents. These can be categorised as *asynchronous* or *synchronous* schemes and as *single iteration* or *multiple iteration* schemes. Furthermore, *synchronous* schemes can be categorised as *parallel* or *serial* schemes. In asynchronous schemes, agents transmit data intermittently and without considering other agents, while in synchronous schemes, agents transmit data concurrently at regular intervals. In single iteration schemes, agents transmit data once before control inputs are implemented, while in multiple iteration schemes, data is transmitted multiple times before control inputs are implemented. Finally, in parallel schemes, agents transmit data at exactly the same time and perform calculations simultaneously, while in serial schemes, agents wait after sending information so that only one agent is performing calculations at a given time. Examples of these communication protocols are illustrated in figure 3.6. Each of these schemes have their own advantages:

- Asynchronous - agents perform actions without waiting for other agents.
- Single iteration - less communication is required between agents, and control inputs are calculated faster.
- Multiple iteration - agents obtain information about the optimal predicted actions of neighbours, allowing them to update their actions accordingly to improve performance and satisfy constraints.
- Serial - the most recent information is received from neighbours.
- Parallel - the most recent information received may be outdated, but calculations are performed faster as multiple agents can make control calculations at once.

Communication protocols are very important in the mathematical formulation of the MPC controller. In [81], Camponogara and Talukdar propose methods for synchronous and asynchronous communication in distributed MPC architectures. In [82], Negeborn, De Schutter, and Hellendoorn propose a serial communication scheme for multi-agent distributed MPC architectures and compare the performance against a parallel scheme for the control of an electric power network.

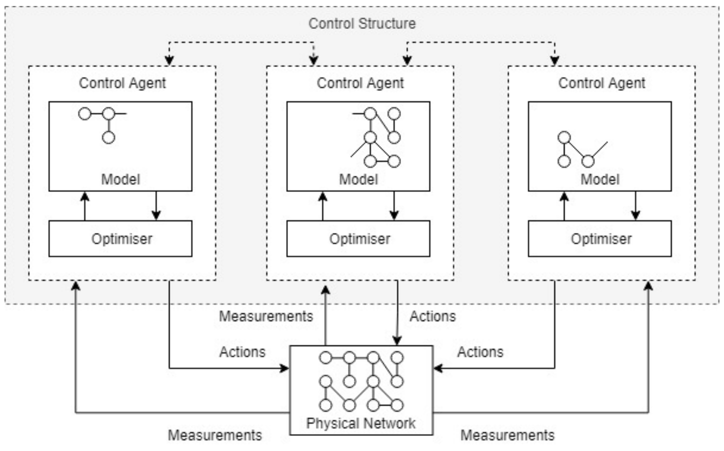


Figure 3.7: Distributed MPC architecture

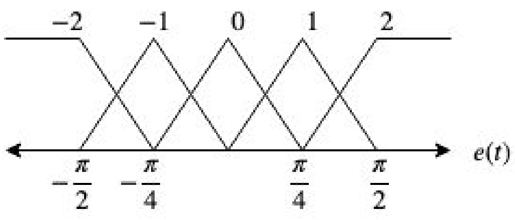


Figure 3.8: Example set of triangular membership functions for variable $e(t)$ In this case, the set of linguistic values are represented by the vector $[-2, -1, 0, 1, 2]$, which could be equivalent to *["large negative", "small negative", "negligible", "small positive", "large positive"]*. Based on diagram from Passino [84].

3.3. Fuzzy Control

As previously mentioned, a fuzzy controller was selected for the inner loop of the PMRAC controller. Fuzzy control is a branch of control theory concerned with the design of controllers which use fuzzy logic. Fuzzy logic is a form of logic created by Zadeh in his formative paper [83] as a method of implementing heuristic knowledge from natural languages into a logical system. Fuzzy controllers have been applied for complex tasks such as flight control, engine control, navigation, satellite attitude control, autonomous vehicle control, and robotics path planning. This section will introduce the fundamental theory of fuzzy logic and fuzzy controllers before reviewing the current state-of-the-art research in aerospace and adaptive fuzzy control.

3.3.1. Fuzzy Logic

Fuzzy logic is an infinite-valued logical system based on *degrees of membership*, which define to what degree an item belongs in a set. *Fuzzy sets* are a set of real numbers which have degrees of membership described by fuzzy logic, as opposed to crisp sets in which memberships are typically described by Boolean (binary) logic. The *degrees of membership* of an item in a fuzzy set are mapped by a *membership function* in the range $[0, 1]$, where 0 indicates no membership and 1 indicates full membership of the fuzzy set. Usually, fuzzy sets are named by *linguistic values* which describe a certain set within a *linguistic variable*. The set of all numbers over which a linguistic variable exists is termed the *universe of discourse*. Figure 3.8 shows a set of membership functions for the set of linguistic values for a linguistic variable.

For instance, the linguistic variable *"temperature"* could have linguistic values *"cold"*, *"warm"*, and *"hot"* described by fuzzy sets. Clearly, these linguistic values are better represented by fuzzy sets rather than crisp sets as there is no clear distinction between when a temperature should be classified as "warm" instead of "hot". The universe of discourse for this linguistic variable may be $\mathbb{R}_{>0} = \{x \in \mathbb{R} | x \geq 0\}$ Kelvin, or the full range of possible temperatures in Kelvin.

Next, the mathematical notation for fuzzy logic and fuzzy sets is introduced. The fuzzy set, M, is

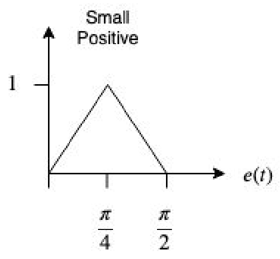


Figure 3.9: Example triangular membership function for the linguistic value *"small positive"* Based on diagram from Passino [84].

described by:

$$M = \{(x, \mu^M(x)) : x \in \mathcal{X}\} \tag{3.63}$$

where $\mathcal{X}$ is the universe of discourse and μ^M is the *membership function* which maps $\mathcal{X}$ to the degree of membership in the range $[0, 1]$. Membership functions can have any shape, but the most common types of membership function are the *triangular*, *singleton* and *Gaussian* membership functions.

The triangular membership function is defined by the piecewise equation:

$$\mu(x) = \begin{cases} \frac{x-a}{m-a} & \text{for } a < x \leq m \\ \frac{b-x}{b-m} & \text{for } m < x \leq b \\ 0 & \text{otherwise} \end{cases} \tag{3.64}$$

where a is the lower bound, b is the upper bound, and m is the mid-point of the membership function. The singleton membership function is equal to unity at one point and zero at all other points:

$$\mu(x) = \begin{cases} 1 & \text{for } x = b \\ 0 & \text{otherwise} \end{cases} \tag{3.65}$$

The Gaussian membership function is defined by:

$$\mu(x) = \exp\left(-\frac{1}{2}\left(\frac{x-c}{\omega}\right)^2\right) \tag{3.66}$$

where c is the mean and ω is the standard deviation. Figure 3.9 shows a triangular membership function μ for the input variable $e(t)$ for the linguistic value *"small positive"*.

3.3.2. Fuzzy Logic Control

Fuzzy logic controllers can generally be categorised as using *type-1* or *type-2* fuzzy logic. Type-1 fuzzy logic uses fuzzy sets which are certain while type-2 fuzzy logic uses fuzzy sets which are also described by fuzzy membership functions [85]. This review will mostly focus on type-1 fuzzy logic controllers as they are the most common.

Fuzzy controllers have several advantages over other control methods. These are:

- It is a *model-free* method, meaning that it does not rely on a mathematical model of the system or process. This is useful for complex systems where it is hard to identify an accurate model of the system. Other model-free control methods include PID control and nonlinear adaptive control.
- Fuzzy logic provides a means to translate *linguistic* control information from human operators into machine-readable control code. This is useful where information, such as a mathematical model of the system, is not available or when the human operater expertise is superior.
- Fuzzy controllers are generally *nonlinear*, meaning that they can be made to perform any nonlinear control action.

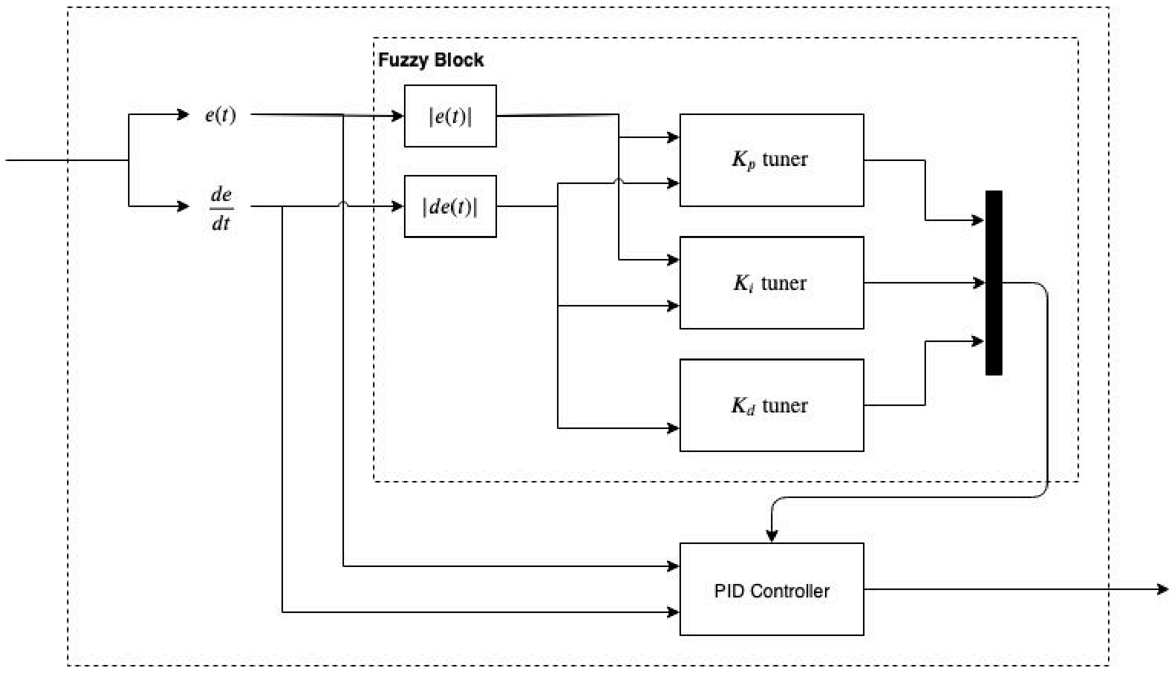


Figure 3.10: Fuzzy PID control block [86]

PID control is one of the oldest and most widespread control techniques today. PID controllers calculate an error value $e(t)$ which is the difference between the setpoint and the system output, and apply a correction based on proportional, integral, and derivative terms[7]. The time-domain control signal for a PID controller is given as:

$$u_{PID}(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \tag{3.67}$$

In which $e(t)$ is the error between the setpoint and the system output, u_{PID} is the control signal, and K_p, K_i, and K_d are the gains for the proportional, integral, and derivative parts of the controller respectively. A drawback with conventional PID controllers is that they are only suitable for a narrow range of operating conditions due to the fixed parameters. Fuzzy-PID controllers allow PID parameters to be tuned online through the addition of a fuzzy control component, in which one fuzzy tuner is used to tune each PID gain.

The fuzzy tuners follow the equation:

$$K_a = K_{a0} + U_a \Delta K_a, U_a \in [0, 1] \tag{3.68}$$

Where a denotes p, i, or d and U_a is the output from the fuzzy controller. The structure of the fuzzy-PID controller is shown in figure 3.10.

The fuzzy controllers receive inputs $|e(t)|$ and $|de(t)|$. The triangular membership functions are defined by the piecewise equation:

$$f_{ji}(x) = \begin{cases} 1 + \frac{(x-a_{ji})}{b_{ji}^-} & \text{if } (-b_{ji}^-) \leq (x - a_{ji}) \leq 0 \\ 1 - \frac{(x-a_{ji})}{b_{ji}^+} & \text{if } 0 \leq (x - a_{ji}) \leq (b_{ji}^+), j = 1, 2, \dots, N \\ 0 & \text{otherwise} \end{cases} \tag{3.69}$$

where for each input i, N membership functions are defined. b_{ji}^- forms the left-side, b_{ji}^+ the right-side, and a_{ji} the centre of the triangle.

[7]Which is where the name Proportional-Integral-Derivative (PID) controller comes from

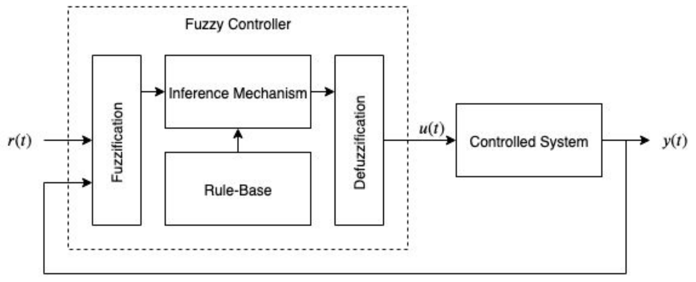


Figure 3.11: Fuzzy controller block diagram. Based on diagram from Passino [84]

Mamdani Fuzzy Inference Systems

It has been proven that any MIMO fuzzy system with n inputs and m outputs can be separated into m multiple-input-single-output (MISO) fuzzy systems [87].

Decision making is implemented in fuzzy controllers through the use of a *rule base*. This is a list of rules that defines the actions to take based on linguistic values. For example, a simple temperature regulation problem may have the rule *"if the temperature is hot and the rate of temperature change is small positive then the temperature regulation is large negative"* in the rule base. The rule base is constructed for all relevant combinations of input variables and linguistic values, and can be implemented in look-up tables or matrices. The majority of rules used in fuzzy control take the general form:

$$\text{If } \tilde{x}_1 \text{ is } \tilde{P}_1^k \text{ and } \dots \text{ and } \tilde{x}_n \text{ is } \tilde{P}_n^m\text{, then } \tilde{y} \text{ is } \tilde{Q}^j \tag{3.70}$$

where the notation, $\sim$, is used to indicate fuzzy sets representing memberships of linguistic variables. The linguistic variable $\tilde{x}_1$ and the linguistic value $\tilde{P}_1^k$ are defined on the universe of discourse $\mathcal{X}$, and the linguistic variable $\tilde{y}$ and the linguistic value $\tilde{Q}^j$ are defined on the universe of discourse $\mathcal{Y}$.

Figure 3.11 shows a block diagram of a fuzzy controller. A fuzzy controller contains four components: the *rule-base*, the *inference mechanism*, the *fuzzification* interface, and the *defuzzification* interface. The fuzzy controller takes an error, $e(t) = y(t) - r(t)$, as an input, and chooses the control input, $u(t)$, so that the control objectives are met.

The *fuzzification* stage converts the crisp inputs, which are typically system states, into fuzzy sets using the membership functions associated with each input variable. Membership functions are chosen based on heuristic knowledge of the system states. If the states are assumed to be accurate then singleton membership functions may be chosen, while if the states are assumed to have some disturbances, triangular or Gaussian membership functions may be chosen.

The *inference* block takes the fuzzy sets from the *fuzzification* block and outputs a fuzzy set of the implied controller actions. The *inference* block has two main stages. The first stage is *matching*, where the degree of firing of each rule in the rule base given the inputs is determined. In the second stage, an implied fuzzy set is created containing the controller outputs recommended by all active rules. Rules are weighted according to their priority when deciding on the final controller output. The implied fuzzy set $\hat{Q}^j$ for rule R_j with the universe of discourse $\mathcal{Y}$ has the membership function:

$$\mu^{\hat{Q}^j}(y) = \mu_j(x) * \mu^{Q^j}(y) \tag{3.71}$$

where the $*$ operation is known as the *T-norm*, which is the fuzzy intersection between $\mu_j(x)$ and $\mu^{Q^j}(y)$. $\mu_j(x)$ is the degree to which the rule is fired and μ^{Q^j} is the degree to which the recommendation of rule R_j is *attenuated*.

The final part of the fuzzy controller is the *defuzzification* block. In this step, the crisp controller output with the highest certainty is calculated from the implied fuzzy set. The most common methods for defuzzification are the *centre of gravity* and *centre average* methods. The centre of gravity defuzzification method uses the equation:

$$y_q^{\text{crisp}} = \frac{\sum_{i=1}^{R} b_i^q \int_{y_q} \mu_{\hat{B}_q^i}(y_q) dy_q}{\sum_{i=1}^{R} \int_{y_q} \mu_{\hat{B}_q^i}(y_q) dy_q} \tag{3.72}$$

where R is the number of rules, b_i^q is the centre of area under of the membership function, $\mu_{\hat{B}_q^i}$ is the membership function for the implied fuzzy set $\hat{B}_q^i$, and $\int_{y_q} \mu_{\hat{B}_q^i}(y_q)dy_q$ gives the area under the membership function. This method requires that the area under the membership functions is finite.

The centre average method is based on the centres of the membership functions and the certainty of the recommended rule:

$$y_q^{\text{crisp}} = \frac{\sum_{i=1}^R b_i^q \sup_{y_q}\{\mu_{\hat{B}_q^i}(y_q)\}}{\sum_{i=1}^R \sup_{y_q}\{\mu_{\hat{B}_q^i}(y_q)\}} \tag{3.73}$$

where "sup" denotes the "supremum", which is the least upper bound. A challenging task in the design of fuzzy controllers is the tuning of parameters, which include the number, shapes, and spacing of membership functions, and the number and type of rules.

Takagi-Sugeno Fuzzy Inference Systems

In Takagi-Sugeno fuzzy inference systems, the consequents of the rules are memoryless functions $f_i(\cdot)$. For a system with rules of the form:

$$\text{If } x_1 \text{ is } P_1^K \text{ and } \cdots \text{ and } x_n \text{ is } P_n^M, \text{ then } q^i = f_i(\cdot) \tag{3.74}$$

The output is a crisp piecewise affine function of the input variables:

$$y^{\text{crisp}} = \frac{\sum_{i=1}^R q^i \mu_i(\boldsymbol{x})}{\sum_{i=1}^R \mu_i(\boldsymbol{x})} \tag{3.75}$$

Takagi-Sugeno fuzzy inference systems have a wider range of applications than Mamdani fuzzy inference systems and can be used for applications such as fuzzy identification and adaptive fuzzy control.

3.3.3. State-of-the-art

In [88], Wang demonstrates an adaptive fuzzy controller, where a feedback control law is used to tune the fuzzy controller membership function parameters online. In [89], Amoozgar, Chamseddine, and Zhang demonstrate a fuzzy gain-scheduled PID controller. Their controller uses a fuzzy inference system to tune the gains of a PID controller online based on the tracking error and rate of tracking error. In [90], Gong and Yao develop a neural network adaptive control method for the control of systems with unknown and unrepeatable disturbances. A multi-layer neural network is tuned online using a set of adaptation laws. Constraints are added to the tuning range of neural network weights at any time in order to avoid divergence. Gong and Yao apply their control method to the motion control of a linear motor drive system, which has non-negligible disturbances. In simulations, the controller is found to have fast convergence and a good overall tracking error, with the worst tracking performance being for the noncontinuous dynamics.

3.4. Adaptive Control

Adaptive control is a control method in which parameters of a controller are tuned due to system states being initially unknown or time-varying. An adaptive controller consists of two loops: a *normal feedback loop* and a *parameter adjustment loop*. There are many different types of adaptive control, such as model identification adaptive control (in which system identification is performed online), iterative learning control, multiple models, and gain scheduling, but the focus of this review will be on model reference adaptive control (MRAC).

A block diagram of a MRAC controller is shown in figure 4.1. In a MRAC controller, the reference model has already been built (using some form of system identification), and is fed into the adjustment mechanism along with the plant inputs and outputs. There are many types of *parameter adjustment law* that may be used for the adjustment mechanism, but the two most commonly used ones are the *steepest descent method* and the *Lyapunov stability method* [91], which apply for both linear and nonlinear systems.

The steepest descent method formulates the problem as an approximate gradient descent optimisation of the error squared performance parameter. We can then use the criterion:

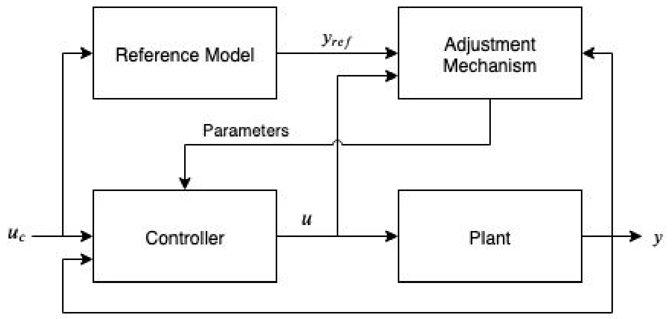


Figure 3.12: Model Reference Adaptive Control block diagram

$$I(\beta) = \frac{1}{2}e^2 \tag{3.76}$$

where β is the parameter vector and e is the error between the system and the reference outputs. From this, the adaptation law can be formulated as:

$$\frac{d\beta}{dt} = -\gamma\frac{\theta e}{\theta\beta}\text{sgn}(e) \tag{3.77}$$

Where sgn(e) is the known *signum function* and γ is the tuning parameter. The Lyapunov-based adaptation law instead assumes that the error $e(t)$ converges to zero as $t \to \infty$. The general form adaptation law derived using the Lyapunov stability criterion is:

$$\dot{\beta} = \gamma\psi e \tag{3.78}$$

where ψ is a known function depending on v and y.

3.4.1. Fuzzy Model Reference Learning Control

The fuzzy model reference learning controller (FMRLC) is a version of MRAC which uses a fuzzy controller, first proposed by Layne and Passino in [92]. The FMRLC is termed a *learning* controller as the past parameter values are used in the tuning process, allowing the controller to quickly adapt to previous operating states.

Figure 3.13 shows a block diagram of the FMRLC. The fuzzy controller receives the error $e(kT)$ and error rate $c(kT) = de(kT)/dt$ as inputs. g_c, g_e, and g_u are the scaling gains for the error, error rate, and controller output respectively. These are initially tuned heuristically and then tuned online.

The membership functions for the inputs to the fuzzy controller are pre-defined and are not tuned by the learning mechanism, whereas the membership functions for the output are tuned.

The task of the *learning mechanism* is to observe the outputs from the plant ($y_m(kT)$), reference model ($y(kT)$), and fuzzy controller and make adjustments to the rule-base accordingly. The two parts to the mechanism are the *fuzzy inverse model* and the *knowledge-base modifier*. The fuzzy inverse model performs an inverse mapping of the output error $y_e(kT)$ to the plant inputs and determines the required changes to reduce the error to zero. The knowledge-base modifier then uses this data to make the required changes to the rule-base. At each sampling instance, changes are made only to the necessary rules in order to reduce the output error to zero.

3.4.2. State-of-the-art

Adaptive control has been applied to complex tasks in aerospace. In [93] Dydek, Annaswamy, and Lavretsky applied model reference adaptive control for the control of a quadrotor UAV. The reference model is built using linearised quadrotor dynamics and a linear baseline controller. Flight test experiments showed improved performance against a linear baseline controller for actuator and loss-of-thrust failures. In [94], Sadeghzadeh et al. applied a model reference adaptive controller using the gradient descent rule for trajectory tracking of a quadrotor, and compared performance to a gain-scheduling PID controller, in which simulations showed improved performance for the former. In [95], Coza and Macnab apply an adaptive-fuzzy control method for the stabilisation of a quadrotor UAV subject to

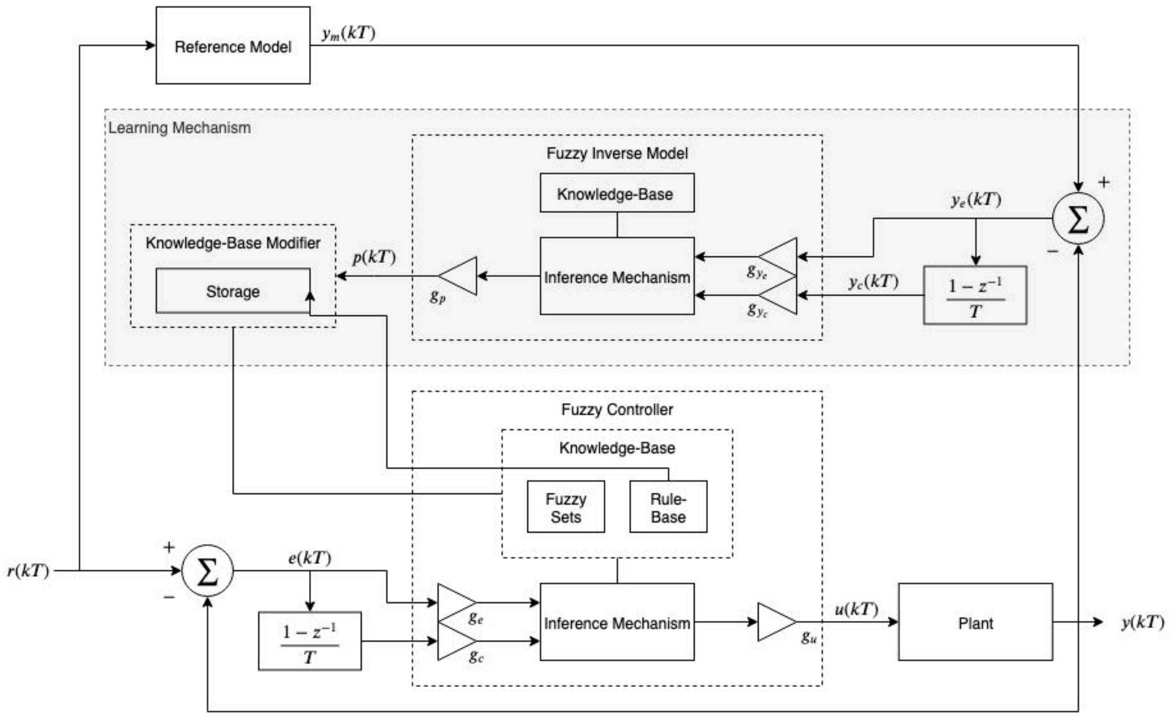


Figure 3.13: Fuzzy model reference learning controller block diagram

wind disturbances. Robustness of the controller is demonstrated by a Lyapunov stability analysis, and simulations show improved performance compared to neural network control and robust sliding-mode control.

Distributed

Limited research was found on the topic of multi-agent or distributed model reference adaptive control. In [96], Kim et al. used partial differential equation-based model reference adaptive control with a Lyapunov adaptation law for the control of heterogeneous multi-agent networks with parameter uncertainty. In a subsequent study [97], Kim et al. then extended their work to include disturbance rejection capabilities by implementation of a third loop containing a disturbance observer. In [98], Peng demonstrate a distributed model reference adaptive control method for a cooperative tracking control task for multi-agent systems subject to unknown disturbances.

4
Problem Definition

In chapter 2, the challenges of search-and-rescue missions and the applications of autonomous UAV systems were discussed. The tasks which multi-agent quadrotor UAV systems can perform were identified and the requirements for such as system were identified. In chapter 3, a review was provided of relevant control theory. This included control system architectures, model predictive control, fuzzy control, and adaptive control.

4.1. Problem Statement

The following problem statement is proposed for the thesis:

> Search-and-rescue missions are executed in complex and dynamic environments and must be completed as fast as possible. Previous research has shown multi-agent networks of cooperating quadrotor UAVs to have a lot of potential in the automation of critical tasks for search-and-rescue missions, including search, mapping, identification, and logistics. The mission environment contains many uncertain and stochastic variables, which may influence the objectives and optimal performance of an automatic search-and-rescue system. Therefore, an automated search-and-rescue system should be able to adapt its behaviour online in order to maximise performance. A distributed predictive model reference adaptive control system is proposed for the autonomous control of a swarm of quadrotors to perform search-and-rescue missions. This controller contains an outer loop with a model predictive controller which performs online tuning of the parameters of an intelligent fuzzy controller for each quadrotor.

4.2. Research Question

With the problem statement defined, the following research questions arise:

- How to design the model predictive controller? One challenge of this is identification of an appropriate model which can take the intelligent controller parameters of nearby quadrotors as inputs and output predictions on the future system states. Another challenge of this is to design an appropriate cost function which takes the predictions on future system states as an input and outputs a number representing the overall performance.
- How to implement the decentralised control architecture for the multi-agent system? The challenges of this include designing appropriate communication schemes between quadrotors, such as whether to use non-iterative or iterative MPC algorithms, or whether to use asynchronous or synchronous MPC algorithms. Further challenges include deciding how many other agents each agent should communicate with, how many quadrotor intelligent controllers are tuned by each MPC, and how to deal with communication issues which arise, such as one agent being unable to communicate with any other agents.

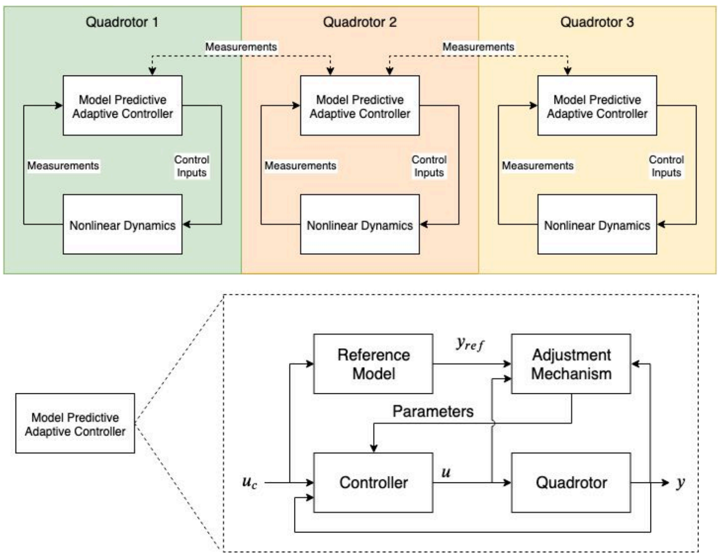


Figure 4.1: Top: Structure of proposed system. Bottom: Model reference adaptive control block diagram. The dashed lines between controllers represent exchange of local information between quadrotors.

- How to design the disaster environment model? The challenges of this task are designing an appropriate disaster environment model for simulating autonomous search-and-rescue missions. The model must include relevant disaster zone features, such as damaged buildings, victims, hazards, smoke, and so on. The model must also be an appropriate size, and allow evaluation of the scalability of the control system and of the number of agents required based on the size of the disaster area. Most importantly, the environment model must provide an accurate representation of reality so that simulations can verify the performance of the control system. Other challenges include selection of the quadrotor model, including dynamics, sensors, and so on.

- How to design the fuzzy controller? The fuzzy controller must be designed to control quadrotors to perform search tasks in the disaster environment given measurements of the local system state from nearby agents. These controllers must also have parameters which can be tuned to alter the control behaviour.

- Is the control system and simulation computationally feasible? The purpose of the control system is to perform in real-time during a disaster mission using the limited computational resources available for quadrotors.

This project has several possible contributions to the fields of control engineering, swarm intelligence, and search-and-rescue:

- A new type of controller is proposed, termed a predictive model reference adaptive controller in this review, which is theorised to be suitable for adaptive control in environments with uncertain and stochastic parameters. This project will attempt to design, implement, and evaluate the performance of this controller against alternative control methods.

- This project will also attempt to implement the controller in a distributed architecture for the control of a multi-agent robotic system. Possible contributions could be advancing understanding of how to implement distributed control for coordination of quadrotor swarms, particularly for search-and-rescue, and how to use the proposed controller for a distributed system.

- The multi-agent system and controller will be simulated for a search-and-rescue mission in an urban disaster zone. This research could advance understanding of how to simulate and assess the performance of autonomous robotic systems performing search-and-rescue missions, and provide a benchmark for future research.

Bibliography

[1] U. O. for Disaster Risk Reduction, *Terminology,* (2019).

[2] A. Khorram-Manesh, *Handbook of disaster and emergency management,* Gothenburg, İsveç: Kompendiet. Kasım **15**, 2018 (2017).

[3] E. Cavallo, A. Powell, and O. Becerra, *Estimating the direct economic damages of the earthquake in haiti,* The Economic Journal **120**, F298 (2010).

[4] P. Wallemacq and R. House, *Economic losses, poverty & disasters: 1998-2017,* (2019).

[5] H. Ritchie and M. Roser, *Ofda/cred international disaster data,* (2019).

[6] E. E. D. (EM-DAT), *Em-dat | the international disasters database,* (2019).

[7] J. A. Barbera and C. G. Cadoux, *Search, rescue, and evacuation,* Critical care clinics **7**, 321 (1991).

[8] A. W. Coburn, R. J. Spence, and A. Pomonis, *Factors determining human casualty levels in earthquakes: mortality prediction in building collapse,* in *Proceedings of the First International Forum on Earthquake related Casualties. Madrid, Spain, July 1992* (1992).

[9] U. N. H. C. for Refugees (UNHCR), *Handbook for emergencies* (United Nations High Commissioner for Refugees, 2007).

[10] T. J. Tanzi, M. Chandra, J. Isnard, D. Camara, O. Sebastien, and F. Harivelo, *Towards" drone-borne" disaster management: future application scenarios,* in *XXIII ISPRS Congress, Commission VIII (Volume III-8)*, Vol. 3 (Copernicus GmbH, 2016) pp. 181–189.

[11] R. R. Murphy, S. Tadokoro, D. Nardi, A. Jacoff, P. Fiorini, H. Choset, and A. M. Erkmen, *Search and rescue robotics,* Springer handbook of robotics , 1151 (2008).

[12] P. Rudol and P. Doherty, *Human body detection and geolocalization for uav search and rescue missions using color and thermal imagery,* in *2008 IEEE aerospace conference* (Ieee, 2008) pp. 1–8.

[13] M. Andriluka, P. Schnitzspan, J. Meyer, S. Kohlbrecher, K. Petersen, O. Von Stryk, S. Roth, and B. Schiele, *Vision based victim detection from unmanned aerial vehicles,* in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems* (IEEE, 2010) pp. 1740–1747.

[14] L. Apvrille, T. Tanzi, and J.-L. Dugelay, *Autonomous drones for assisting rescue services within the context of natural disasters,* in *2014 XXXIth URSI General Assembly and Scientific Symposium (URSI GASS)* (IEEE, 2014) pp. 1–4.

[15] A. R. Kolbe, R. A. Hutson, H. Shannon, E. Trzcinski, B. Miles, N. Levitz, M. Puccio, L. James, J. R. Noel, and R. Muggah, *Mortality, crime and access to basic needs before and after the haiti earthquake: a random survey of port-au-prince households,* Medicine, conflict and survival **26**, 281 (2010).

[16] P. Schmuck and M. Chli, *Multi-uav collaborative monocular slam,* in *2017 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2017) pp. 3863–3870.

[17] S. Verykokou, C. Ioannidis, G. Athanasiou, N. Doulamis, and A. Amditis, *3d reconstruction of disaster scenes for urban search and rescue,* Multimedia Tools and Applications **77**, 9691 (2018).

[18] Zipline, (2019).

[19] M. M. Torok, M. Golparvar-Fard, and K. B. Kochersberger, *Image-based automated 3d crack detection for post-disaster building assessment,* Journal of Computing in Civil Engineering **28**, A4014004 (2013).

[20] M. Erdelj, E. Natalizio, K. R. Chowdhury, and I. F. Akyildiz, *Help from the sky: Leveraging uavs for disaster management,* IEEE Pervasive Computing **16**, 24 (2017).

[21] I. Bekmezci, O. K. Sahingoz, and Ş. Temel, *Flying ad-hoc networks (fanets): A survey,* Ad Hoc Networks **11**, 1254 (2013).

[22] H. Kitano, S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjou, and S. Shimada, *Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research,* in *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 99CH37028)*, Vol. 6 (IEEE, 1999) pp. 739–743.

[23] J. Wang, M. Lewis, and J. Gennari, *Emerging areas: urban operations and ucavs: a game engine based simulation of the nist urban search and rescue arenas,* in *Proceedings of the 35th conference on Winter simulation: driving innovation* (Winter Simulation Conference, 2003) pp. 1039–1045.

[24] J. H. Lee, *Model predictive control: Review of the three decades of development,* International Journal of Control, Automation and Systems **9**, 415 (2011).

[25] Y. Wang and S. Boyd, *Fast model predictive control using online optimization,* IEEE Transactions on control systems technology **18**, 267 (2009).

[26] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari, *Embedded online optimization for model predictive control at megahertz rates,* IEEE Transactions on Automatic Control **59**, 3238 (2014).

[27] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, *Constrained model predictive control: Stability and optimality,* Automatica **36**, 789 (2000).

[28] S. a. Keerthi and E. G. Gilbert, *Optimal infinite-horizon feedback laws for a general class of constrained discrete-time systems: Stability and moving-horizon approximations,* Journal of optimization theory and applications **57**, 265 (1988).

[29] C. Chen and L. Shaw, *On receding horizon feedback control,* Automatica **18**, 349 (1982).

[30] L. Magni and R. Sepulchre, *Stability margins of nonlinear receding-horizon control via inverse optimality,* Systems & Control Letters **32**, 241 (1997).

[31] R. Bitmead, *Adaptive optimal control,* The thinking man's GPC (1990).

[32] J. B. Rawlings and K. R. Muske, *The stability of constrained receding horizon control,* IEEE transactions on automatic control **38**, 1512 (1993).

[33] H. Michalska and D. Q. Mayne, *Robust receding horizon control of constrained nonlinear systems,* IEEE transactions on automatic control **38**, 1623 (1993).

[34] M. Sznaier and M. J. Damborg, *Suboptimal control of linear systems with state and control inequality constraints,* in *26th IEEE Conference on Decision and Control*, Vol. 26 (IEEE, 1987) pp. 761–762.

[35] E. Polak and T. Yang, *Moving horizon control of linear systems with input saturation and plant uncertainty part 1. robustness,* International Journal of Control **58**, 613 (1993).

[36] D. W. Clarke, C. Mohtadi, and P. Tuffs, *Generalized predictive control—part i. the basic algorithm,* Automatica **23**, 137 (1987).

[37] D. W. Clarke, C. Mohtadi, and P. Tuffs, *Generalized predictive control—part ii extensions and interpretations,* Automatica **23**, 149 (1987).

[38] L. Grüne and J. Pannek, *Nonlinear model predictive control,* in *Nonlinear Model Predictive Control* (Springer, 2017) pp. 45–69.

[39] A. Bemporad and M. Morari, *Robust model predictive control: A survey,* in *Robustness in identification and control* (Springer, 1999) pp. 207–226.

[40] E. F. Camacho, D. R. Ramírez, D. Limón, D. M. De La Peña, and T. Alamo, *Model predictive control techniques for hybrid systems,* Annual reviews in control **34**, 21 (2010).

[41] M. Lazar, W. Heemels, S. Weiland, and A. Bemporad, *Stabilizing model predictive control of hybrid systems,* IEEE Transactions on Automatic Control **51**, 1813 (2006).

[42] A. Alessio and A. Bemporad, *A survey on explicit model predictive control,* in *Nonlinear model predictive control* (Springer, 2009) pp. 345–369.

[43] A. Bemporad and C. Filippi, *Suboptimal explicit receding horizon control via approximate multiparametric quadratic programming,* Journal of optimization theory and applications **117**, 9 (2003).

[44] T. A. Johansen and A. Grancharova, *Approximate explicit constrained linear model predictive control via orthogonal search tree,* IEEE Transactions on Automatic Control **48**, 810 (2003).

[45] A. Bemporad and C. Filippi, *An algorithm for approximate multiparametric convex programming,* Computational optimization and applications **35**, 87 (2006).

[46] B. Kouvaritakis and M. Cannon, *Model predictive control,* Switzerland: Springer International Publishing (2016).

[47] F. Oldewurtel, A. Parisio, C. N. Jones, M. Morari, D. Gyalistras, M. Gwerder, V. Stauch, B. Lehmann, and K. Wirth, *Energy efficient building climate control using stochastic model predictive control and weather predictions,* in *Proceedings of the 2010 American control conference* (IEEE, 2010) pp. 5100–5105.

[48] J. Yan and R. R. Bitmead, *Incorporating state estimation into model predictive control and its application to network traffic control,* Automatica **41**, 595 (2005).

[49] X. Yu-Geng, L. De-Wei, and L. Shu, *Model predictive control—status and challenges,* Acta Automatica Sinica **39**, 222 (2013).

[50] U. Eren, A. Prach, B. B. Koçer, S. V. Raković, E. Kayacan, and B. Açıkmeşe, *Model predictive control in aerospace systems: Current state and opportunities,* Journal of Guidance, Control, and Dynamics **40**, 1541 (2017).

[51] G. Chowdhary, M. Mühlegg, J. P. How, and F. Holzapfel, *Concurrent learning adaptive model predictive control,* in *Advances in Aerospace Guidance, Navigation and Control* (Springer, 2013) pp. 29–47.

[52] T. Templeton, D. H. Shim, C. Geyer, and S. S. Sastry, *Autonomous vision-based landing and terrain mapping using an mpc-controlled unmanned rotorcraft,* in *Proceedings 2007 IEEE International Conference on Robotics and Automation* (IEEE, 2007) pp. 1349–1356.

[53] S. Gros, R. Quirynen, and M. Diehl, *Aircraft control based on fast non-linear mpc & multiple-shooting,* in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)* (IEEE, 2012) pp. 1142–1147.

[54] W. B. Dunbar, M. B. Milam, R. Franz, and R. M. Murray, *Model predictive control of a thrust-vectored flight control experiment,* IFAC Proceedings Volumes **35**, 355 (2002).

[55] J. Shin and H. J. Kim, *Nonlinear model predictive formation flight,* IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans **39**, 1116 (2009).

[56] J. L. Crassidis, F. L. Markley, T. C. Anthony, and S. F. Andrews, *Nonlinear predictive control of spacecraft,* Journal of Guidance, Control, and Dynamics **20**, 1096 (1997).

[57] K. Alexis, C. Papachristos, R. Siegwart, and A. Tzes, *Robust model predictive flight control of unmanned rotorcrafts,* Journal of Intelligent & Robotic Systems **81**, 443 (2016).

[58] R. C. Shekhar, M. Kearney, and I. Shames, *Robust model predictive control of unmanned aerial vehicles using waysets,* Journal of Guidance, Control, and Dynamics **38**, 1898 (2015).

[59] K. Alexis, G. Darivianakis, M. Burri, and R. Siegwart, *Aerial robotic contact-based inspection: planning and control,* Autonomous Robots **40**, 631 (2016).

[60] A. Richards and J. P. How, *Robust variable horizon model predictive control for vehicle maneuvering,* International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal **16**, 333 (2006).

[61] H. J. Kim, D. H. Shim, and S. Sastry, *Nonlinear model predictive tracking control for rotorcraft-based unmanned aerial vehicles,* in *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, Vol. 5 (IEEE, 2002) pp. 3576–3581.

[62] L. Blackmore, M. Ono, A. Bektassov, and B. C. Williams, *A probabilistic particle-control approximation of chance-constrained stochastic predictive control,* IEEE transactions on Robotics **26**, 502 (2010).

[63] D. Lyons, J.-P. Calliess, and U. D. Hanebeck, *Chance constrained model predictive control for multi-agent systems with coupling constraints,* in *2012 American Control Conference (ACC)* (IEEE, 2012) pp. 1223–1230.

[64] Ø. Hegrenæs, J. T. Gravdahl, and P. Tøndel, *Spacecraft attitude control using explicit model predictive control,* Automatica **41**, 2107 (2005).

[65] F. Borrelli, T. Keviczky, and G. J. Balas, *Collision-free uav formation flight using decentralized optimization and invariant sets,* in *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)*, Vol. 1 (IEEE, 2004) pp. 1099–1104.

[66] A. Richards and J. How, *Decentralized model predictive control of cooperating uavs,* in *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)*, Vol. 4 (IEEE, 2004) pp. 4286–4291.

[67] D. H. Shim, H. J. Kim, and S. Sastry, *Decentralized nonlinear model predictive control of multiple flying robots,* in *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, Vol. 4 (IEEE, 2003) pp. 3621–3626.

[68] L. Deori, S. Garatti, and M. Prandini, *A model predictive control approach to aircraft motion control,* in *2015 American Control Conference (ACC)* (IEEE, 2015) pp. 2299–2304.

[69] Y. Wang, A. Ramirez-Jaime, F. Xu, and V. Puig, *Nonlinear model predictive control with constraint satisfactions for a quadcopter,* in *Journal of Physics: Conference Series*, Vol. 783 (IOP Publishing, 2017) p. 012025.

[70] T. Baca, G. Loianno, and M. Saska, *Embedded model predictive control of unmanned micro aerial vehicles,* in *2016 21st International Conference on Methods and Models in Automation and Robotics (MMAR)* (IEEE, 2016) pp. 992–997.

[71] A. T. Hafez, A. J. Marasco, S. N. Givigi, M. Iskandarani, S. Yousefi, and C. A. Rabbath, *Solving multi-uav dynamic encirclement via model predictive control,* IEEE Transactions on control systems technology **23**, 2251 (2015).

[72] J. Ji, A. Khajepour, W. W. Melek, and Y. Huang, *Path planning and tracking for vehicle collision avoidance based on model predictive control with multiconstraints,* IEEE Transactions on Vehicular Technology **66**, 952 (2016).

[73] S. Di Cairano, H. Park, and I. Kolmanovsky, *Model predictive control approach for guidance of spacecraft rendezvous and proximity maneuvering,* International Journal of Robust and Nonlinear Control **22**, 1398 (2012).

[74] R. Scattolini, *Architectures for distributed and hierarchical model predictive control–a review,* Journal of process control **19**, 723 (2009).

[75] W. B. Dunbar and R. M. Murray, *Distributed receding horizon control for multi-vehicle formation stabilization,* Automatica **42**, 549 (2006).

[76] E. Franco, L. Magni, T. Parisini, M. M. Polycarpou, and D. M. Raimondo, *Cooperative constrained control of distributed agents with nonlinear dynamics and delayed information exchange: A stabilizing receding-horizon approach,* IEEE Transactions on Automatic Control **53**, 324 (2008).

[77] B. Johansson, A. Speranzon, M. Johansson, and K. H. Johansson, *Distributed model predictive consensus,* in *International Symposium on Mathematical Theory of Networks and Systems, Kyoto, Japan, July 24-28 2006* (2006) pp. 2438–2444.

[78] G. Ferrari-Trecate, L. Galbusera, M. P. E. Marciandi, and R. Scattolini, *A model predictive control scheme for consensus in multi-agent systems with single-integrator dynamics and input constraints,* in *2007 46th IEEE Conference on Decision and Control* (IEEE, 2007) pp. 1492–1497.

[79] G. Ferrari-Trecate, L. Galbusera, M. P. E. Marciandi, and R. Scattolini, *Contractive distributed mpc for consensus in networks of single-and double-integrators,* IFAC Proceedings Volumes **41**, 9033 (2008).

[80] G. Ferrari-Trecate, L. Galbusera, M. P. E. Marciandi, and R. Scattolini, *Model predictive control schemes for consensus in multi-agent systems with single-and double-integrator dynamics,* IEEE Transactions on Automatic Control **54**, 2560 (2009).

[81] E. Camponogara and S. N. Talukdar, *Distributed model predictive control: Synchronous and asynchronous computation,* IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans **37**, 732 (2007).

[82] R. R. Negenborn, B. De Schutter, and J. Hellendoorn, *Multi-agent model predictive control for transportation networks: Serial versus parallel schemes,* Engineering Applications of Artificial Intelligence **21**, 353 (2008).

[83] L. A. Zadeh, *Fuzzy logic,* Computer **21**, 83 (1988).

[84] K. M. Passino, S. Yurkovich, and M. Reinfrank, *Fuzzy control*, Vol. 42 (Citeseer, 1998).

[85] P. A. Birkin and J. M. Garibaldi, *A comparison of type-1 and type-2 fuzzy controllers in a micro-robot context,* in *2009 IEEE international conference on fuzzy systems* (IEEE, 2009) pp. 1857–1862.

[86] K. K. Ahn and D. Q. Truong, *Online tuning fuzzy pid controller using robust extended kalman filter,* Journal of Process Control **19**, 1011 (2009).

[87] X.-J. Zeng and M. G. Singh, *Approximation theory of fuzzy systems-mimo case,* IEEE Transactions on Fuzzy Systems **3**, 219 (1995).

[88] L.-X. Wang, *Stable adaptive fuzzy control of nonlinear systems,* IEEE Transactions on fuzzy systems **1**, 146 (1993).

[89] M. H. Amoozgar, A. Chamseddine, and Y. Zhang, *Fault-tolerant fuzzy gain-scheduled pid for a quadrotor helicopter testbed in the presence of actuator faults,* IFAC Proceedings Volumes **45**, 282 (2012).

[90] J. Gong and B. Yao, *Neural network adaptive robust control of nonlinear systems in semi-strict feedback form,* Automatica **37**, 1149 (2001).

[91] S. G. Tzafestas, *Introduction to mobile robot control* (Elsevier, 2013).

[92] J. R. Layne and K. M. Passino, *Fuzzy model reference learning control,* Journal of Intelligent & Fuzzy Systems **4**, 33 (1996).

[93] Z. T. Dydek, A. M. Annaswamy, and E. Lavretsky, *Adaptive control of quadrotor uavs: A design trade study with flight evaluations,* IEEE Transactions on control systems technology **21**, 1400 (2012).

[94] I. Sadeghzadeh, A. Mehta, Y. Zhang, and C.-A. Rabbath, *Fault-tolerant trajectory tracking control of a quadrotor helicopter using gain-scheduled pid and model reference adaptive control,* in *Annual Conference of the Prognostics and Health Management Society*, Vol. 2 (2011).

[95] C. Coza and C. J. Macnab, *A new robust adaptive-fuzzy control method applied to quadrotor helicopter stabilization,* in *NAFIPS 2006-2006 Annual Meeting of the North American Fuzzy Information Processing Society* (IEEE, 2006) pp. 454–458.

[96] J. Kim, K.-D. Kim, V. Natarajan, S. D. Kelly, and J. Bentsman, *Pde-based model reference adaptive control of uncertain heterogeneous multiagent networks,* Nonlinear Analysis: Hybrid Systems **2**, 1152 (2008).

[97] J. Kim, V. Natarajan, S. D. Kelly, and J. Bentsman, *Disturbance rejection in robust pde-based mrac laws for uncertain heterogeneous multiagent networks under boundary reference,* Nonlinear Analysis: Hybrid Systems **4**, 484 (2010).

[98] Z. Peng, D. Wang, H. Zhang, G. Sun, and H. Wang, *Distributed model reference adaptive control for cooperative tracking of uncertain dynamical multi-agent systems,* IET Control Theory & Applications **7**, 1079 (2013).

1 Introduction

Mission planning for multi-agent systems presents a significant challenge when applied to systems operating in dynamic, complex, and uncertain environments. These systems must coordinate and adapt their actions in real time to achieve complex objectives, often in the face of uncertain environmental states and incomplete information. Effective mission planning requires balancing the need for predictive planning to anticipate future states with the ability to make rapid adjustments based on real-time feedback.

Another major challenge in real-time control problems is the limitation of computational resources. Controller architectures such as Model Predictive Control (MPC) require extensive computation to predict future states and optimise control actions; however, the computational resources available and the real-time nature of the control problem impose challenges, necessitating efficient algorithms that can operate within these limits.

Finally, traditional approaches for supervisory control also face several limitations, often relying on either purely predictive or reactive control strategies, which can result in sub-optimal system performance. Purely predictive models may struggle to adapt to uncertain environments, while reactive strategies cannot anticipate future events. Additionally, they may not be able to accurately account for coordination between agents, for which the optimal actions may depend on the actions of other agents in the system. Additionally, scaling these systems to operate in larger environments or with more agents can be challenging. Therefore, more sophisticated approaches are needed to handle the inherent complexities of multi-agent coordination and environmental variability.

Predictive control strategies, such as MPC, rely on predicting future states and optimising control actions based on these predictions. While MPC can effectively handle constraints and anticipate future events, it requires extensive computational resources. Conversely, reactive control strategies, such as Fuzzy Logic Control (FLC), adjust actions based on current state feedback without predicting future states. FLC is computationally efficient and robust but cannot plan for future actions.

1.1. Problem Statement

1.1.1. Proposed Control Architecture

This thesis proposes a novel control architecture, termed Model Predictive Fuzzy Control (MPFC), for the mission planning of multi-agent systems. MPFC addresses many of the challenges in dynamic, complex, and uncertain environments by combining the predictive capabilities of Model Predictive Control (MPC) with the flexibility of Fuzzy Logic Control (FLC).

In this architecture:

- A supervisory MPC is used to optimise the output Membership Function (MF) parameters of the local FLC mission planning controllers for each agent.

- The FLC controllers enable autonomous mission planning for individual agents, while the MPC tunes agent behaviour over time according to predicted future states.

By leveraging this hybrid approach, MPFC reduces the set of optimisation variables required by traditional MPC while maintaining effective agent coordination and mission planning autonomy.

1.1.2. Applications

MPFC has potential applications in various domains where multi-agent systems operate in dynamic and uncertain environments, including:

- Search and Rescue (SaR): Coordinating agents to locate and assist victims in disaster response scenarios.
- Surveillance Operations: Monitoring and tracking targets in real-time.
- Traffic Management: Optimising agent cooperation to improve system performance in urban traffic networks.

1.1.3. Research Context

This research complements recent advancements in mission planning and control for multi-agent systems. The studies outlined below demonstrate the state of the art of the various avenues of research touched on in this thesis.

Hierarchical frameworks have been explored by de Koning and Jamshidnejad [6], who investigate hierarchical cooperative mission planning for non-homogeneous autonomous SaR robots, focusing on optimising victim search behaviour in multi-agent systems. Similarly, Esteves et al. [3] explore the cooperation of aerial and ground robots for camera-based SaR, integrating vision-based pose estimation and motion tracking of victims.Cao et al. [2] propose a multi-agent reinforcement learning framework that improves efficiency in SAR tasks involving unknown, dynamic targets. By employing adaptive learning strategies, this approach enhances agent collaboration in uncertain and evolving conditions.

Planning in dynamic environments necessitates adaptive, robust control methods. Baglioni and Jamshidnejad [1] introduce a control technique for mission planning under dynamic conditions, incorporating victim evacuation models and positional uncertainties. Toumieh and Lambert [14] present a framework that combines MPC with time-aware safe corridors to increase multi-agent planning efficiency. Papaioannou et al. [11] detail a distributed search-planning framework for 3D environments that can handle a dynamically varying number of agents. This flexibility strengthens its capacity to manage changing mission parameters.

Robust performance in dynamic environments depends on effective disturbance management. Surma and Jamshidnejad [13] develops a state-dependent dynamic tube MPC approach that models and learns disturbance behaviours via fuzzy models. This strategy reduces the conservatism typical of traditional robust control methods, improving feasibility and stability in challenging settings. Sarbaz et al. [12] propose a hierarchical, optimisation-based MPC framework for large-scale discrete fuzzy systems, explicitly addressing time-varying delays and disturbances. By incorporating time-varying disturbance models, their method enhances overall stability and performance.

Scalability is an important topic when it comes to real-world implementation of multi-agent systems. Lu et al. [7] present DrMaMP, a distributed, real-time algorithm tailored for mission planning in cluttered environments. Their solution improves scalability and optimises agent coordination under resource constraints and dynamic conditions. Zhou et al. [15] introduce a distributed reinforcement learning strategy for multi-robot systems, demonstrating heightened adaptability and coordination in complex 3D environments. Their findings illustrate effective implementation of reinforcement learning to address scalability concerns in multi-agent operations.

On the topic of adaptive fuzzy models, Grosset et al. [5] implement a fuzzy multi-agent simulator for battery recharge management in autonomous industrial vehicles, utilising adaptive fuzzy logic to guide agent decision making among dynamic conditions.

Building on this body of work, this thesis introduces MPFC as a method for achieving robust and computationally efficient mission planning in multi-agent systems.

1.2. Project Scope

In order to limit the project scope it is necessary to specify the bounds for the definition of the control problem. These top-level assumptions form the initial constraints used to then build and implement the research.

AS1: *The system is in discrete time and in discrete 2D space in the x and y-planes.*
This assumption allows all modelling to use discrete dimensions and simplifies the planar dimension to represent a 2D physical area of land in which the system must operate.

AS2: The system is fully bounded within the environment and does not interface with any other external systems.
This assumption limits the system to be fully-contained within the environment, and does not interact with, for example, external data sources or other autonomous systems.

AS3: All agents in the system are identical.
This assumption constrains the all agents in the system to be homogeneous, and do not have differing designs, such as battery life, velocities, and sensors.

AS4: The control problem is formulated as a single-objective optimisation, with the objective function calculated using environment parameters and parameters influenced by agent actions.
With this assumption, the control problem is simplified to a single objective and requires that the environment states and actions of the agents influence the optimisation.

AS5: Information transfer is instantaneous and perfect.
Here, it is assumed that any information transfer between agents, controllers, and the environment, does not result in any latency, loss or corruption of data.

AS6: The prediction step of MPFC is instantaneous in relation to the system model and does not need to be scheduled ahead of time.
This assumption simplifies the scheduling of the MPFC controller, which is dependent on hardware and introduces complexities in the optimisation step and updating of FLC parameters.

AS7: In the MPFC architecture, the optimisation variables are the parameters of the fuzzy membership functions used to generate the output of the FIS.
This limits the scope by rejecting several other FLC parameters that may be selected as the optimisation variables of MPFC, including the input MF parameters, number and shape of input and output MFs, and the rule base itself.

AS8: MPFC has a perfect representative model of the system.
This assumption means that there is no difference between the MPFC prediction model and the system itself, even if MPFC cannot perfectly predict the values of future probabilistic states.

AS9: At any time step, MPFC has perfect knowledge of all agent states at that time step.
Finally, this assumption means that MPFC has perfect knowledge of the states of all other agents at a given time step, even though agents may not communicate information with each other directly.

1.3. Project Objectives

Based on the assumptions and problem statement, a series of objectives for this research are now formulated.

OB1: *Specify a generic mathematical definition of the MPFC controller.*
With the top-level assumptions defined, the first objective is to define a generic mathematical description of the MPFC controller independent of the case study for which it is implemented.

OB2: *Define and implement the MPFC controller model for a specific case study in simulation.*
Afterwards, the next objective is to apply this mathematical description to a specific case study and implement it in a simulation.

OB3: *Validate the performance of MPFC against alternative controller architectures via extensive computer-based simulations.*
This simulation will then be used to assess MPFC against other methods.

OB4: *Analyse the sensitivity of the MPFC controller to various design parameters.*
A linear sensitivity analysis will then be performed a for a range of MPFC controller design parameters.

OB5: *Explore design improvements to the basic MPFC controller implemented.*
The final objective is to perform a design exploration of the initial MPFC controller implementation in order to understand what improvements can be made.

1.4. Research Questions

To address the project objectives, three research questions are formulated.

RQ1: *How does MPFC perform compared to traditional MPC and FLC controllers in a dynamic environment?*
This research question forms the objective of the MPFC performance analysis, and selects two alternative controller architectures against which to measure performance.

RQ2: *Which design parameters drive the performance of MPFC and how can they be optimised?*
This research question targets the sensitivity analysis to identify the most influential design parameters of MPFC and understand how they should be defined when configuring an MPFC to optimise performance.

RQ3: *What design choices can be made then configuring an MPFC and in which cases should they be made?*
This research question targets the design exploration to identify strengths and weaknesses of certain design choices which must be made when configuring MPFC.

1.5. Thesis Structure

This thesis is structured a follows. In Chapter 2, a generic mathematical description of MPFC in the context of mission planning for a multi-agent system is created according to the assumptions made in the project scope (Section 1.2). This involves building the environment model (Section 2.1), the agent model (Section 2.2), and the controller models (Section 2.3). In Chapter 3, these mathematical models are applied to build a simulation in which we apply MPFC to a SaR mission scenario. In Chapter 4, the performance of MPFC is assessed via simulation. This includes a performance analysis in which the performance of MPFC is compared against alternative controllers (Section 4.2), a sensitivity analysis in which the sensitivity of MPFC performance to certain design parameters is analysed (Section 4.3), and a design exploration in which various design choices for MPFC are assessed (Section 4.2). In Chapter 5, the findings of the thesis are summarised and a series of recommendations for future research are identified.

2 Mathematical Model

This section introduces the general mathematical model of MPFC applied to the mission planning of a multi-agent autonomous system in a bounded geographical area. This model comprises three components: the environment model, describing the physical domain; the agent model, governing agent dynamics; and the MPFC controller model, implementing the control architecture.

To describe the spatial domain of the model, we define a spatial coordinate parameter, s, and a flattened cell index, m.

The parameter s represents the spatial coordinates (i, j) of a cell in a discrete 2D grid, where i and j denote the row and column indices, respectively. The indices satisfy $i \in \{1, 2, \ldots, n^{\text{x}}\}$ and $j \in \{1, 2, \ldots, n^{\text{y}}\}$, where n^{x} and n^{y} are the total numbers of rows and columns in the grid.

The flattened cell index, m, is given by $m \in \{1, 2, \ldots, n^{\text{c}}\}$, where $n^{\text{c}} = n^{\text{x}} \cdot n^{\text{y}}$, and is used as an index for matrix-based operations.

With this notation, the spatial coordinates (i, j) of a cell indexed by m are denoted as $\boldsymbol{s}_m$.

2.1. Environment Model Formulation

We define an environment model that describes the evolution of the environment states in discrete time over a 2D space in the x and y-dimensional planes. The state vector for the environment model is introduced as $\boldsymbol{\phi}^{\text{env}}(\boldsymbol{s}_m, k)$, where $\boldsymbol{s}_m = (i, j)$ indicates the spatial cell coordinates and k is the discrete simulation time step. Note that the terminology *"simulation time step"* and *"time step"* is used interchangeably in this thesis. They evolve according to a discrete-time model with both deterministic and uncertain components:

$$\boldsymbol{\phi}^{\text{env}}(\boldsymbol{s}_m, k+1) = \boldsymbol{\phi}^{\text{env}}(\boldsymbol{s}_m, k) + \Delta t \left(\mathcal{F}(\boldsymbol{\phi}^{\text{env}}(\boldsymbol{s}_m, k), \boldsymbol{s}_m, k) + \mathcal{G}(\boldsymbol{\phi}^{\text{env}}(\boldsymbol{s}_m, k), \boldsymbol{s}_m, k) \right) \tag{2.1}$$

where $\boldsymbol{\phi}^{\text{env}}(\boldsymbol{s}_m, k)$ is the environment state vector at cell $\boldsymbol{s}_m$ and simulation time step k. We introduce a generic functional notation to represent potentially complex models, where the function $\mathcal{F}(\cdot)$ captures deterministic environmental dynamics, the function $\mathcal{G}(\cdot)$ represents uncertain dynamics, and Δt is the simulation time-step size.

2.2. Agent Model Formulation

We define an agent model for the multi-agent system that describes the evolution of the agent states and agent spatial information (spatial data which is measured or estimated by the agents, including measurements of environment states) in discrete time over a 2D spatial domain. The state vector for the agent model (for all agents) is given by $\boldsymbol{\phi}^{\text{agent}}(k)$ and the state vector for agent spatial information is given by $\boldsymbol{\phi}^{\text{search}}(\boldsymbol{s}_m, k)$.

The agent model is given by:

$$\phi^{\text{agent}}(k+1) = \phi^{\text{agent}}(k) + \Delta t \mathcal{H}(\phi^{\text{agent}}(k), \phi^{\text{search}}(\boldsymbol{s}_m, k)) \tag{2.2}$$

where $\phi^{\text{agent}}(k)$ is the state vector of the agents at simulation time step k, $\phi^{\text{search}}(\boldsymbol{s}_m, k)$ is the agent spatial information at cell coordinates $\boldsymbol{s}_m$ and simulation time step k, and Δt is the simulation time step size.

The search states, represented by $\phi^{\text{search}}(\boldsymbol{s}_m, k)$, are computed via a deterministic function of the agent states $\phi^{\text{agent}}(k)$ and the environment states $\phi^{\text{env}}(\boldsymbol{s}_m, k)$. The search states evolve according to:

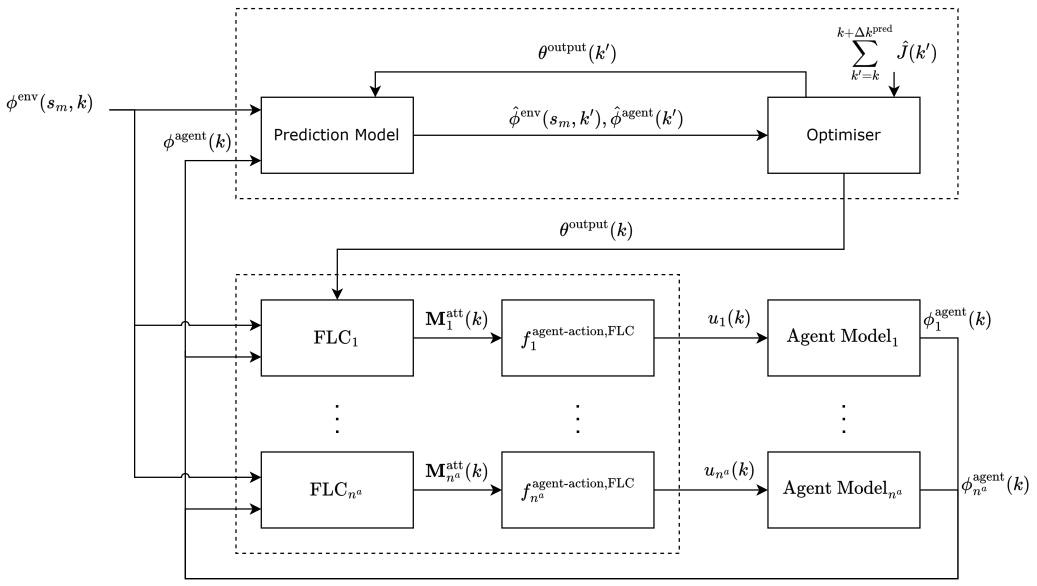
$$\phi^{\text{search}}(\boldsymbol{s}_m, k+1) = \phi^{\text{search}}(\boldsymbol{s}_m, k) + \Delta t \mathcal{K}(\phi^{\text{agent}}(k), \phi^{\text{env}}(\boldsymbol{s}_m, k)) \tag{2.3}$$

where $\phi^{\text{search}}(\boldsymbol{s}_m, k)$ is the state vector of the search map at spatial cell coordinates $\boldsymbol{s}_m$ and simulation time step k, $\phi^{\text{agent}}(k)$ is the state vector of the agents at simulation time step k, and Δt is the simulation time step size.

2.3. MPFC Formulation

In this section a generic MPFC controller model is formulated for the mission planning of a multi-agent homogeneous system.

2.3.1. Controller Architecture



$\theta^{\text{output}}(k')$
$\sum_{k'=k}^{k+\Delta k^{\text{pred}}} \hat{J}(k')$
$\phi^{\text{env}}(s_m, k)$
$\phi^{\text{agent}}(k)$
Prediction Model
$\hat{\phi}^{\text{env}}(s_m, k'), \hat{\phi}^{\text{agent}}(k')$
Optimiser
$\theta^{\text{output}}(k)$
FLC_1
$\mathbf{M}_1^{\text{att}}(k)$
$f_1^{\text{agent-action,FLC}}$
$u_1(k)$
Agent Model_1
$\phi_1^{\text{agent}}(k)$
FLC_{n^a}
$\mathbf{M}_{n^a}^{\text{att}}(k)$
$f_{n^a}^{\text{agent-action,FLC}}$
$u_{n^a}(k)$
Agent Model_{n^a}
$\phi_{n^a}^{\text{agent}}(k)$

Figure 2.1: Centralised MPFC Architecture

The generic MPFC controller architecture for a system with n^{a} agents is displayed in Figure 2.1. In the figure, the supervisory MPC controller is within the upper dashed box, and the individual FLC controllers for each agent are within the lower dashed box. The MPC controller part contains an *Optimiser* block in which the optimisation is performed, and a *Prediction Model* block, in which future system states are predicted.

2.3.2. Global Objective Function

The global objective function quantifies the performance of the system in a bounded 2D area over a discrete time step. First, we define the objective at a single time step k as the sum of the cell-level objective contributions over all cells:

$$J(k) = \sum_{m=1}^{n^{\text{c}}} f^{\text{obj}} \left(\phi^{\text{env}}(\boldsymbol{s}_m, k), \phi^{\text{search}}(\boldsymbol{s}_m, k), \phi^{\text{agent}}(k) \right) \Delta \boldsymbol{s}_m \tag{2.4}$$

where $J(k)$ is the objective function value at simulation time step k, $f^{\text{obj}}(\cdot)$ is the objective function evaluated at cell $\boldsymbol{s}_m$ and simulation time step k. The inner sum $\sum_{m=1}^{n^{\text{c}}}$ represents the sum over all spatial cells in the discretised 2D spatial domain, where n^{c} is the total number of cells and $\Delta \boldsymbol{s}_m$ is the cell area.

2.3.3. Generic MPFC Optimisation Function

The MPFC optimisation function is formulated as a prediction of the global objective function (Equation 2.4) over the prediction horizon. First, the MPFC calculates an estimated objective function

$$\hat{J}(k) = \sum_{m=1}^{n^{\text{c}}} f^{\text{obj}} \left(\hat{\boldsymbol{\phi}}^{\text{env}}(\boldsymbol{s}_m, k), \hat{\boldsymbol{\phi}}^{\text{search}}(\boldsymbol{s}_m, k), \hat{\boldsymbol{\phi}}^{\text{agent}}(k), \boldsymbol{\theta}^{\text{output}}(k) \right) \Delta \boldsymbol{s}_m \tag{2.5}$$

where $\hat{\boldsymbol{\phi}}^{\text{env}}$ is the estimated environment state vector, $\hat{\boldsymbol{\phi}}^{\text{agent}}$ is the estimated agents state vector, $\hat{\boldsymbol{\phi}}^{\text{search}}$ is the estimated agent spatial information state vector, and $\boldsymbol{\theta}^{\text{output}}(k)$ is the vector of FLC output parameters across all agents.

The optimisation function for the MPFC is then defined as:

$$\max_{\boldsymbol{\theta}^{\text{output}}(k)} \sum_{k=1}^{N} \hat{J}(k) \Delta t \tag{2.6}$$

where we set the vector of FLC output parameters, $\boldsymbol{\theta}^{\text{output}}(k)$, as the optimisation variable. The optimisation constraints are not defined for now as these are defined in the specific implementations formulated later.

2.3.4. Generic FLC Architecture

The FLC may be implemented with any choice of the type of FIS, inputs and outputs, number and shapes of input and output MFs, and the design of the rule base.

The FLC outputs an *attraction map*, $\boldsymbol{M}_a^{\text{att}}(k)$, of dimensions $(n^{\text{x,search}}, n^{\text{y,search}})$. Each cell's value indicates its *attraction* to the agent, used by the mission planning algorithm to assign appropriate cells to the agent tasks.

$$\boldsymbol{M}_a^{\text{att}}(k) = f_a^{\text{FLC}} \left(\boldsymbol{\phi}^{\text{env}}(\boldsymbol{s}_m, k), \boldsymbol{\phi}_a^{\text{agent}}(k) \right) \tag{2.7}$$

where f_a^{FLC} is a function representing the FLC for agent a, and the FLC receives as inputs the measured environment state vector $\boldsymbol{\phi}^{\text{env}}(\cdot)$ and the agent state vector $\boldsymbol{\phi}_a^{\text{agent}}(\cdot)$.

$$u_a(k) = f_a^{\text{agent-action,FLC}} \left(\boldsymbol{M}_a^{\text{att}}(k) \right) \tag{2.8}$$

where $f_a^{\text{agent-action,FLC}}$ is the mission planning function for agent a and $u_a(k)$ is the control input to the agent local controller.

3 Simulation and Prediction Model

The mathematical model defined in section 2 is non-explicit and cannot be mathematically solved for an optimal control solution. Therefore, we assess the performance of MPFC for a selected control application through simulation focused on a selected use case.

As stated in Assumption AS8, we assume the predictive controller has perfect models of the system. Consequently, these models are used directly in the prediction stage.

In this section, we take the generic mathematical model defined in section 2 and define a specific case study to implement in the simulation environment. As a top-level assumption, we reduce the n-dimensional space to a 2D space in the x and y-dimensional planes so that $\boldsymbol{s}_m = (i, j)$, where i is the x-axis coordinate and j is the y-axis coordinate.

3.1. Environment Simulation Model

The **environment** is defined with dimensions $(l^{\text{x,env}}, l^{\text{y,env}})$, which is divided into a grid of cells of dimensions $(\Delta x^{\text{env}}, \Delta y^{\text{env}})$. At each simulation time step, any environment state $\phi^{\text{env}}(\boldsymbol{s}_m, k)$, is represented by an $(n^{\text{x,env}}, n^{\text{y,env}})$ matrix, where $n^{\text{x,env}}$ and $n^{\text{y,env}}$ are the respective numbers of environment cells along the x and y axes.

The set of environment model states, $\phi^{\text{env}}(\boldsymbol{s}_m, k)$, for each cell with coordinates $\boldsymbol{s}_m$ in the disaster environment are given by:

1. The **victim map**, $\boldsymbol{M}^{\text{victim}}$, is a matrix of the number of victims in each cell, where the elements $\boldsymbol{M}^{\text{victim}}_m \in \mathbb{Z}^{+}$.
2. The **building map**, $\boldsymbol{M}^{\text{building}}$, is a matrix of the proportion of each cell occupied by buildings, where the elements $\boldsymbol{M}^{\text{building}}_m \in [0, 1]$.
3. The **fire map**, $\boldsymbol{M}^{\text{fire}}(k)$, is a matrix of the fire state in each cell at simulation time step k, where the elements $\boldsymbol{M}^{\text{fire}}_m(k) \in \{0, 1, 2, 3, 4\}$.
4. The **burn time map**, $\boldsymbol{M}^{\text{burn-time}}(k)$, is a matrix of the amount of time since a cell has caught fire at simulation time step k, where the elements $\boldsymbol{M}^{\text{burn-time}}_m(k) \in \mathbb{R}$.
5. The **structure map**, $\boldsymbol{M}^{\text{structure}}$, is a matrix of the flammability of each cell, where the elements $\boldsymbol{M}^{\text{structure}}_m \in [0, 1]$.
6. The **wind velocity**, v^{wind}, is the velocity of the wind $(\mathrm{m\,s^{-1}})$ in the disaster environment, where $v^{\text{wind}} \in \mathbb{R}$ is modelled as a constant over time and over the disaster environment for simplicity.
7. The **wind direction**, θ^{wind}, is the direction of the wind (rad) in the disaster environment, where $\theta^{\text{wind}} \in [-\pi, \pi]$ is modelled as a constant over time and over the disaster environment for simplicity.

3.1.1. Fire Model

We choose to introduce an uncertain dynamic environment state which can influence the objective function evaluated by the system. To achieve this, we define the *fire map* $\boldsymbol{M}^{\text{fire}}(k)$, and introduce a discrete cellular automata model for the fire map dynamics defined by Ohgai et. al. [10] and Freire and DaCamara [4].

The meanings of the fire map parameter values are defined in Table 3.1.

Table 3.1: Definition of fire state, $\boldsymbol{M}_m^{\text{fire}}(k)$

Value	State	Description
0	Non flammable	The cell cannot catch fire
1	Flammable	The cell is not burning but can catch fire
2	Catching fire	The cell is catching fire but cannot spread fire
3	Burning	The cell is on fire and can spread fire
4	Extinguished	The cell is extinguished and cannot spread fire

For visualisation of the fire map, the colour scheme in Figure 3.1 is established.

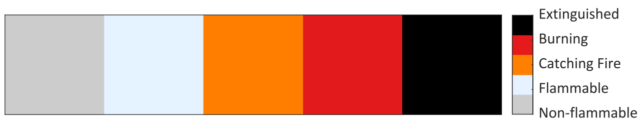


Figure 3.1: Colour scheme for fire map states

At each simulation time step, the fire map is updated according to Equation 3.1.

$$\boldsymbol{M}_m^{\text{fire}}(k) = \begin{cases} 2 & \text{if } \boldsymbol{M}_m^{\text{fire}}(k-1) = 1 \text{ and } \boldsymbol{F}_m(k) \geq \eta^{\text{fire-spread}} \\ 3 & \text{if } \boldsymbol{M}_m^{\text{fire}}(k-1) = 2 \text{ and } \boldsymbol{M}_m^{\text{burn-time}}(k) \geq t^{\text{ignition}} \\ 4 & \text{if } \boldsymbol{M}_m^{\text{fire}}(k-1) = 3 \text{ and } \boldsymbol{M}_m^{\text{burn-time}}(k) \geq t^{\text{burnout}} \\ \boldsymbol{M}_m^{\text{fire}}(k-1) & \text{otherwise} \end{cases} \tag{3.1}$$

where $t^{\text{ignition}} = 120\,\text{s}$ is the time for a cell to ignite, $t^{\text{burnout}} = 600\,\text{s}$ is the time for a cell to burn out, and $\eta^{\text{fire-spread}} \in [0, 1]$ is a randomly generated number for each instance of Equation 3.1, which is compared against the probability of fire spread to determine whether a cell is ignited or not. The **fire spread probability map**, $\boldsymbol{F}(k)$, is a matrix of the probability of fire spreading to each cell at simulation time step k, where the elements $\boldsymbol{F}_m \in [0, 1]$.

The *probability of ignition* for cells in the neighbourhood of an active fire f at cell $\boldsymbol{s}_f = (p, q)$ is given by Equation 3.2.

$$p_f^{\text{ignition}}(k) = \begin{cases} 0 & \text{if } \boldsymbol{M}_f^{\text{burn-time}}(k) > t^{\text{burnout}} \\ \frac{4}{t^{\text{burnout}} - t^{\text{ignition}}} \boldsymbol{M}_f^{\text{burn-time}}(k) + \frac{0.2t^{\text{burnout}} - 4.2t^{\text{ignition}}}{t^{\text{burnout}} - t^{\text{ignition}}} & \text{if } \boldsymbol{M}_f^{\text{burn-time}}(k) \leq \frac{t^{\text{burnout}} - t^{\text{ignition}}}{5} + t^{\text{ignition}} \\ \frac{5}{4(t^{\text{burnout}} - t^{\text{ignition}})} (-\boldsymbol{M}_f^{\text{burn-time}}(k) + t^{\text{burnout}}) & \text{otherwise} \end{cases} \tag{3.2}$$

Figure 3.2 displays the relationship between p_f^{ignition} and burn time, t^{burn}, when $t^{\text{ignition}} = 120\,\text{s}$ and $t^{\text{burnout}} = 600\,\text{s}$. After a cell begins catching fire, the probability of ignition increases rapidly from below 0 before the ignition time to a maximum value of 1 during the start of the burn phase, then slowly decreasing until the burnout time.

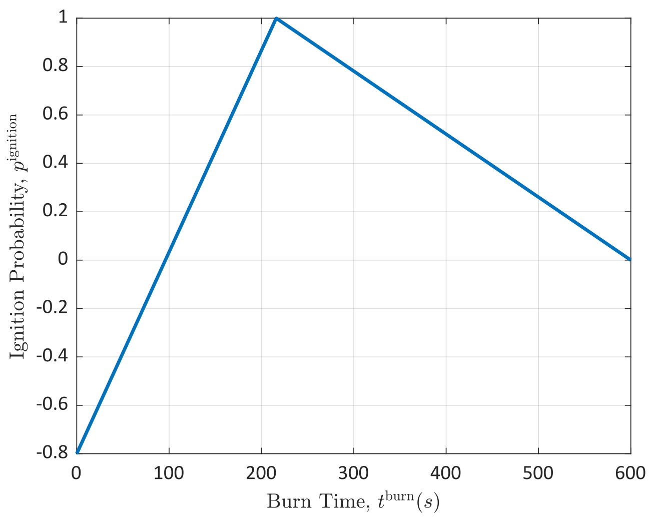


Figure 3.2: Chart of Ignition Probability with Burn Time

For each active fire, the fire spread probability is calculated based on wind influence and environmental conditions:

$$F_m(k) = c^{\text{fs1}}(\boldsymbol{M}_m^{\text{structure}} \cdot \boldsymbol{M}_m^{\text{building}}) \cdot \boldsymbol{W}_m(k)^{c^{\text{fs2}}} \cdot p_f^{\text{ignition}}(k) \tag{3.3}$$

where $M_m^{\text{structure}}$ is the flammability of each cell as defined in Table 3.2, where a value of 0 indicates a non-flammable cell and a value of 1 indicates a fully-flammable cell, identified as wooden buildings in the model. For our study, we allow any values within the range $[0, 1]$. The parameter c^{fs1} is a scaling factor that influences how much the structure and building environment contribute to the fire spread probability, and c^{fs2} is an exponent that modifies the influence of the wind direction on the fire spread probability.

Table 3.2: Building types represented by values in the structure map, $M_m^{\text{structure}}$

Value	Description
0	Fireproof buildings
0.6	Fire prevention wooden buildings
1	Wooden buildings

Figure 3.3 displays the correlation between fire spread and $M^{\text{structure}}$ for an example environment where $v^{\text{wind}} = 0\,\text{m}\,\text{s}^{-1}$, where cells with active fires are given as green and cells with have burnt out are given as yellow. Higher values of $M^{\text{structure}}$ greatly increase the speed at which the fire propagates.

The *wind direction factor*, $M_m^{\text{wind-direction}}$, defines the influence of wind direction on the spread of fire from one cell to another based on the angle between the wind direction and the direction from the fire

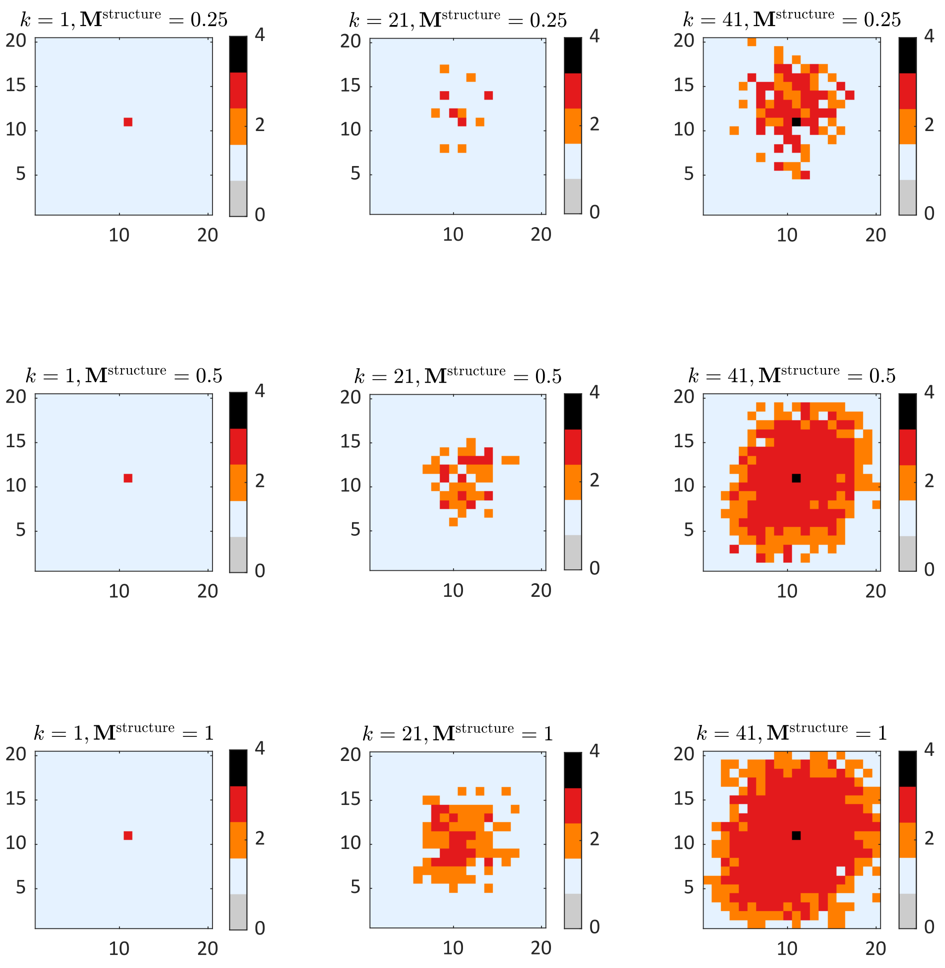


Figure 3.3: Fire Spread over time with $M^{\text{structure}}$

source to the target cell according to the following equation:

$$\boldsymbol{M}_m^{\text{wind-direction}} = \exp\left(v^{\text{wind}}\left(c^{\text{wm1}} + c^{\text{wm2}}\left(\cos(\theta^{\text{wind}} - \theta^{\text{fire}}(\boldsymbol{s}_m)) - 1\right)\right)\right) \tag{3.4}$$

where c^{wm1} and c^{wm2} are constants.

The *wind distance factor*, $\boldsymbol{M}_m^{\text{wind-distance}}$, is given by:

$$\boldsymbol{M}_m^{\text{wind-distance}} = \exp\left(c^{\text{wmd}}\sqrt{(\boldsymbol{s}_m(i) - \boldsymbol{s}_f(p))^2 + (\boldsymbol{s}_m(j) - \boldsymbol{s}_f(q))^2}\right) \tag{3.5}$$

where $\boldsymbol{s}_f = (p, q)$ is the fire cell at coordinates (p, q) and $\boldsymbol{s}_m = (i, j)$ is the environment cell at coordinates (i, j).

The *wind spread map* used in Equation 3.3 is given by the product of the wind direction and wind distance modifiers:

$$W_m = \boldsymbol{M}_m^{\text{wind-direction}} \cdot \boldsymbol{M}_m^{\text{wind-distance}} \tag{3.6}$$

for all cells within the radius r^{wind} of the active fire.

Finally, Algorithm 1 is used to calculate the fire spread probability for all active fires at each time step k and consolidate them into a single fire spread probability map parameter, $\boldsymbol{F}$.

Algorithm 1 Update Fire Spread Probability

```
1: Input: $\boldsymbol{F}$, $\boldsymbol{W}$, $\boldsymbol{P}^{\text{ignition}}$, $c^{\text{fs1}}$, $c^{\text{fs2}}$, $\boldsymbol{M}^{\text{structure}}$, $\boldsymbol{M}^{\text{building}}$, $n^{\text{x,env}}$, $n^{\text{y,env}}$, $r^{\text{wind}}$
2: Output: $\boldsymbol{F}$
3: for each active fire cell $\boldsymbol{s}_f$ in the environment do
4:     Compute row range: $R^{\text{row}} \leftarrow [\max(1, p - r^{\text{wind}}), \min(n^{\text{x,env}}, p + r^{\text{wind}})]$
5:     Compute column range: $R^{\text{col}} \leftarrow [\max(1, q - r^{\text{wind}}) : \min(n^{\text{y,env}}, q + r^{\text{wind}})]$
6:     Update fire spread probability:
7:     $\boldsymbol{F}(R^{\text{row}}, R^{\text{col}}) \leftarrow \boldsymbol{F}(R^{\text{row}}, R^{\text{col}}) + c^{\text{fs1}} \cdot (\boldsymbol{M}^{\text{structure}}(R^{\text{row}}, R^{\text{col}}) \cdot \boldsymbol{M}^{\text{building}}(R^{\text{row}}, R^{\text{col}})) \cdot \boldsymbol{W}^{c^{\text{fs2}}} \cdot p_f^{\text{ignition}}$
8: end for
```

In Figure 3.4, an example of $\boldsymbol{F}$ for a range of values of v^{wind} is shown for a disaster environment of size $n^{\text{x,search}}, n^{\text{y,search}} = 9$, where an active fire is initialised in the centre, $\boldsymbol{M}^{\text{fire}}(5, 5) = 3$. In this example, we set $r^{\text{wind}} = 3$ and $\theta^{\text{wind}} = \frac{\pi}{4}$. For this example, we can see that there is zero possibility of the fire spreading further than the wind range from the active fire.

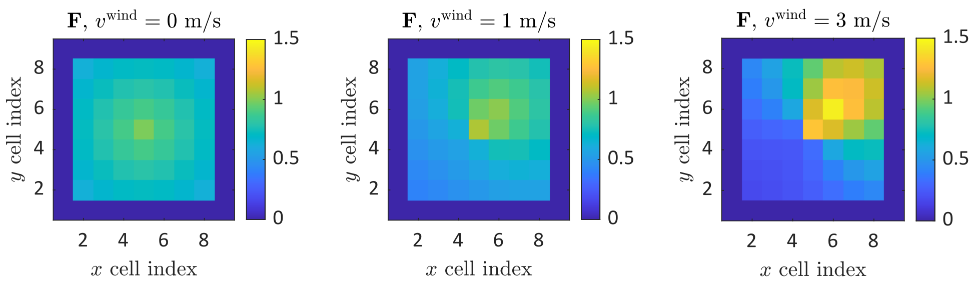


Figure 3.4: Variation of $\boldsymbol{F}$ with v^{wind}

Figure 3.5 demonstrates the relationship between fire spread and wind velocity. In this simulation, a fire is initiated in the centre of a (20×20) cell disaster environment with the wind direction set as $\theta^{\text{wind}} = \frac{\pi}{4}\text{rad}$ and $\boldsymbol{M}^{\text{structure}} = 0.2 \cdot \mathbf{1}_{20\times20}$, where $\mathbf{1}_{20\times20}$ is a matrix of ones of size 20. It can be seen that the correlation between fire spread and wind direction is greater at higher wind velocities, however, the behaviour of this model may be tuned using the constants defined in the equations above[1].

[1]In this example we set $c^{\text{fs1}} = 0.1$, $c^{\text{fs2}} = 1.2$, $c^{\text{wm1}} = 0.15$, $c^{\text{wm2}} = 1$, and $c^{\text{wmd}} = 1$.

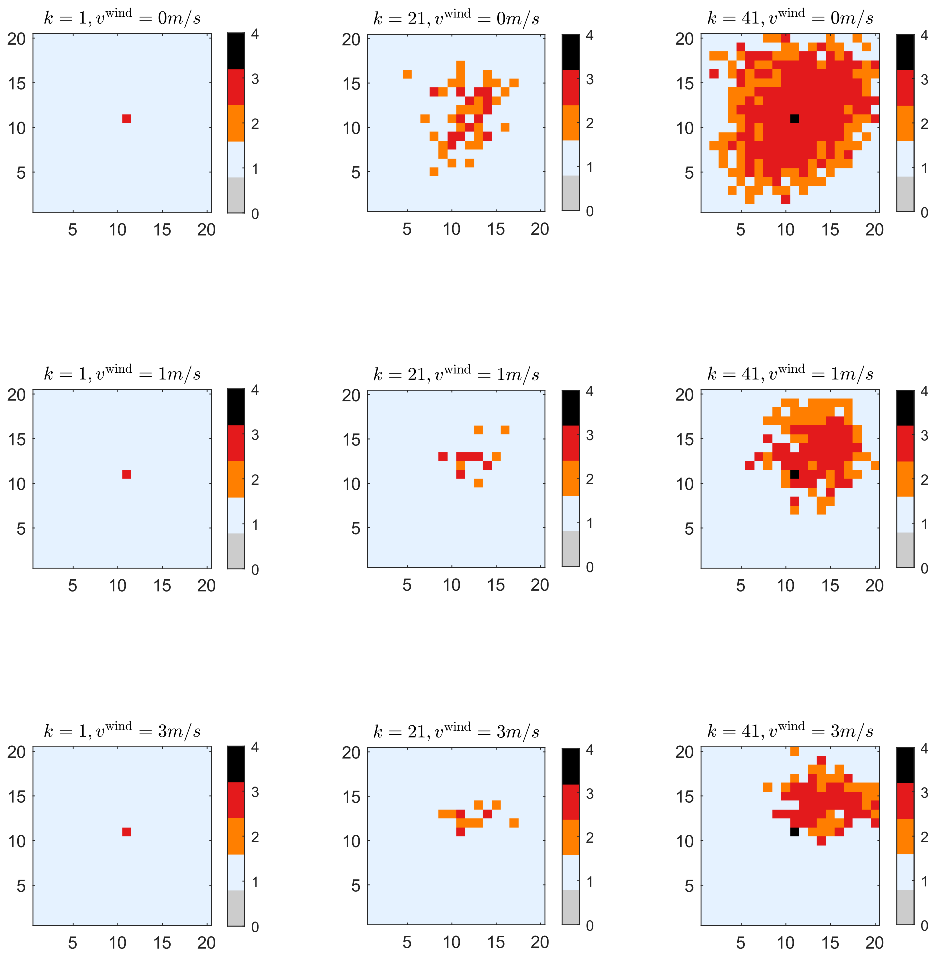


Figure 3.5: Fire Spread over time with v^{wind}

3.2. Agent Simulation Model

The agent spatial information states, $\boldsymbol{\phi}^{\text{search}}(\boldsymbol{s}_m, k)$, represents spatial information measured by the agents over the same area represented by the environment states, but divided into a smaller grid of cells of dimensions $(n^{\text{x,search}}, n^{\text{y,search}})$, where we define $n^{\text{x,search}}) = n^{\text{x,env}})/c^{\text{coarsen}}$ and $n^{\text{y,search}}) = n^{\text{y,env}})/c^{\text{coarsen}}$ for a coarsening factor, $c^{\text{coarsen}} \in \mathbb{Z}^{+}$.

To reduce the number of variables in the MPFC and MPC prediction and optimisation steps and tune the computational load, we define a *coarsening factor*, c^{coarsen}, which is a constant used to consolidate the environment parameters measured by the agents in a $c^{\text{coarsen}} \cdot c^{\text{coarsen}}$ grid of cells in the environment model to a single cell in the agent measurement model. Note that if $c^{\text{coarsen}} = 1$, then the agent measurement model will map one-to-one with the environment model.

The coarsened search map states can be calculated from the environment map states using the following function:

$$\boldsymbol{M}_m^{\text{c}} = \frac{1}{(c^{\text{coarsen}})^2} \sum_{p=1}^{c^{\text{coarsen}}} \sum_{q=1}^{c^{\text{coarsen}}} \boldsymbol{M}_{(c^{\text{coarsen}}(i-1)+p,\, c^{\text{coarsen}}(j-1)+q)} \tag{3.7}$$

where $\boldsymbol{M}$ is a generic matrix and $\boldsymbol{M}^{\text{c}}$ is the corresponding coarsened matrix.

The agent spatial information states, $\boldsymbol{\phi}^{\text{search}}(\boldsymbol{s}_m, k)$, form an $n^{\text{x,search}} \times n^{\text{y,search}}$ matrix, representing data measured or estimated by the agents on environment spatial parameters, and can be viewed as coarsened versions of the corresponding environment state matrices:

- **Coarsened scan map**, $\boldsymbol{M}^{\text{scan}}$, representing the certainty of the multi-agent system regarding environment state values in each cell, where $\boldsymbol{M}_m^{\text{scan}} \in [0, 1]$.
- **Coarsened building search map**, $\boldsymbol{M}^{\text{c,building}}$, representing the proportion of each cell occupied by buildings, where $\boldsymbol{M}_m^{\text{c,building}} \in [0, 1]$.
- **Coarsened fire search map**, $\boldsymbol{M}^{\text{c,fire}}$, representing the mean fire state in each cell, where $\boldsymbol{M}_m^{\text{c,fire}} \in [0, 4]$.
- **Coarsened victim search map**, $\boldsymbol{M}^{\text{c,victim}}$, indicating the number of victims in each cell, where $\boldsymbol{M}_m^{\text{c,victim}} \in \mathbb{Z}^{+}$.
- **Coarsened downwind map**, $\boldsymbol{M}^{\text{c,downwind}}$, indicating the downwind factor (e.g., wind influence) in each cell, where $\boldsymbol{M}_m^{\text{c,downwind}} \in [0, 1]$.

The **agent model** describes the parameters of the multi-agent system. The multi-agent system is assumed to be homogeneous, where all agents have the same parameters and initial values. Each agent, a, consists of a motor, which allows movement in a two-dimensional plane; a sensor, which allows the agent to measure all environment states in the surrounding area; and a local controller, which allows the agent to choose actions to perform.

The agent model states, $\boldsymbol{\phi}^{\text{agent}}(k)$, are defined as:

- **Sensor accuracy**, η: the accuracy of the agent sensor when measuring environment states, where $\eta \in [0, 1]$.
- **Cell scan time**, $t^{\text{scan-cell}}$: the total time for an agent to scan a cell in the search map, where $t^{\text{scan-cell}} \in \mathbb{R}$.
- **Agent airspeed**, v^{airspeed}: the airspeed of the agent when moving between locations, where $v^{\text{airspeed}} \in \mathbb{R}$.
- **Number of agents**, n^{a}: the number of agents in the system, where $n^{\text{a}} \in \mathbb{Z}^{+}$.
- **Agent queue length**, n^{queue}: the number of target cells the agent can schedule in advance, where $n^{\text{queue}} \in \mathbb{Z}^{+}$, and is used in the MPC controller architectures (Equations 3.31 and 3.32).
- **Agent travel time**, $\boldsymbol{t}_a^{\text{travel}}(k)$: the remaining time for agent a to travel to its target cell, where $\boldsymbol{t}_a^{\text{travel}}(k) \in \mathbb{R}$, $\forall a \in \{1, \ldots, n^{\text{a}}\}$.
- **Agent scan time**, $\boldsymbol{t}^{\text{scan}}(k)$: the vector of remaining scan times for each agent, where $\boldsymbol{t}_a^{\text{scan}}(k) \in [0, t^{\text{scan-cell}}]$, $\forall a \in \{1, \ldots, n^{\text{a}}\}$.

- **Agent task**, $\boldsymbol{\tau}^{\text{task}}(k)$: the vector of current agent tasks, where $\tau_a^{\text{task}}(k) \in \{0, 1\}$, $\forall a \in \{1, \dots, n^{\text{a}}\}$.
- **Agent location**, $\boldsymbol{s}^{\text{location}}(k) = [\boldsymbol{s}_1^{\text{location}}(k), \dots, \boldsymbol{s}_{n^{\text{a}}}^{\text{location}}(k)]$: the vector of agent locations, where $\boldsymbol{s}_a^{\text{location}}(x) \in \{0, 1, \cdots, n^{\text{x,search}}\}$, $\boldsymbol{s}_a^{\text{location}}(y) \in \{0, 1, \cdots, n^{\text{y,search}}\}$, $\forall a \in \{1, \dots, n^{\text{a}}\}$.
- **Agent target**, $\boldsymbol{s}^{\text{target}}(k) = [\boldsymbol{s}_1^{\text{target}}(k), \dots, \boldsymbol{s}_{n^{\text{a}}}^{\text{target}}(k)]$: the vector of agent target cells, where $\boldsymbol{s}_a^{\text{target}}(x) \in \{0, 1, \cdots, n^{\text{x,search}}\}$ and $\boldsymbol{s}_a^{\text{target}}(y) \in \{0, 1, \cdots, n^{\text{y,search}}\}$, $\forall a \in \{1, \dots, n^{\text{a}}\}$.

3.2.1. Agent Task Model

The agent task parameter $\tau_a^{\text{task}}(k)$ is defined as follows:

$$\tau_a^{\text{task}}(k) = \begin{cases} 0 & \text{if the current task for the agent is to travel to a new cell} \\ 1 & \text{if the current task for the agent is to scan the cell it is in} \end{cases} \tag{3.8}$$

This parameter is used in the MPFC/FLC agent action algorithm (Algorithm 3) and the MPC agent action algorithm (Algorithm 4) to track the current task of each agent.

3.2.2. Agent Scan Model

For the *scan map*, we define a model in which the certainty of the multi-agent system of the environment states increases when a cell is scanned and degrades linearly over time due to dynamics or uncertainties in the environment states. This parameter is used as a component in the global and predicted objective functions (Equations 3.19 and 3.23).

The scan map is updated using the following model:

$$\boldsymbol{M}_m^{\text{scan}}(k) = \begin{cases} \max\{\boldsymbol{M}_m^{\text{scan}}(k-1) - \sigma, 0\} & \text{if cell } c \text{ is not scanned at time step } k \\ \max\{\boldsymbol{M}_m^{\text{scan}}(k-1) - \sigma, \eta\} & \text{if cell } c \text{ is scanned at time step } k \text{ by agent } a \end{cases} \tag{3.9}$$

where η is the agent sensor accuracy and σ is the degradation in confidence at each time step.

3.2.3. Agent Dynamics Model

A simple dynamics model is adapted to model agent movement. When moving, it is assumed that the agent moves in a straight direction from the centre of its current cell in the search map to the centre of its target cell in the search map.

The agent's ground velocity v^{ground} is given by:

$$v^{\text{ground}}(\boldsymbol{s}_1, \boldsymbol{s}_2) = \sqrt{\begin{aligned} &\left(v^{\text{airspeed}} \cos\left(\theta^{\text{travel}}(\boldsymbol{s}_1, \boldsymbol{s}_2)\right) + v^{\text{wind}} \cos(\theta^{\text{wind}})\right)^2 \\ &\quad + \left(v^{\text{airspeed}} \sin\left(\theta^{\text{travel}}(\boldsymbol{s}_1, \boldsymbol{s}_2)\right) + v^{\text{wind}} \sin(\theta^{\text{wind}})\right)^2 \end{aligned}} \tag{3.10}$$

The travel direction θ^{travel} is given by:

$$\theta^{\text{travel}}(\boldsymbol{s}_1, \boldsymbol{s}_2) = \arctan 2\big(\boldsymbol{s}_2(j) - \boldsymbol{s}_1(j), \boldsymbol{s}_2(i) - \boldsymbol{s}_1(i)\big), \tag{3.11}$$

where i and j represent the indices for the x-axis and y-axis indices respectively.

The travel time of an agent between two coordinates is given by:

$$t^{\text{travel}}(\boldsymbol{s}_1, \boldsymbol{s}_2) = \frac{\|\boldsymbol{s}_1 - \boldsymbol{s}_2\|}{v^{\text{ground}}(\boldsymbol{s}_1, \boldsymbol{s}_2)}, \tag{3.12}$$

where s_1 and s_2 are the initial and final coordinates respectively.

Figure 3.6 displays an example of the travel times of an agent to each cell in the disaster environment at differing wind velocities. In this example $v^{\text{airspeed}} = 5\,\text{m}\,\text{s}^{-1}$, $\theta^{\text{wind}} = \frac{\pi}{4}$, $l^{\text{x,env}} = l^{\text{y,env}} = 10\,\text{m}$. The agent is positioned in cell $(5, 5)$, represented by the red circle.

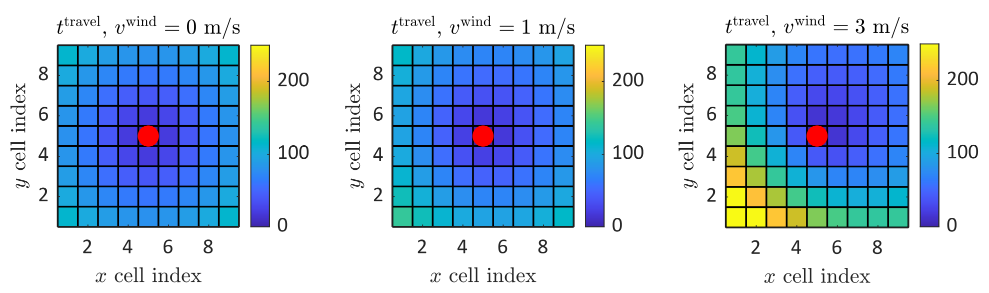


Figure 3.6: Agent travel time, $t^{\text{travel}}(\boldsymbol{s}_a^{\text{location}}, \boldsymbol{s}_m)$, with v^{wind}

3.3. Local Controller Model

The local controller model is used to model the behaviour of the local controllers for each agent, managing which tasks are allocated to each agent and which actions are performed by each agent. The local controller is assumed to be isolated for each agent and does not transfer data between other local controllers, therefore each local controller will make decisions without considering the actions of other agents.

3.3.1. FLC Local Controller

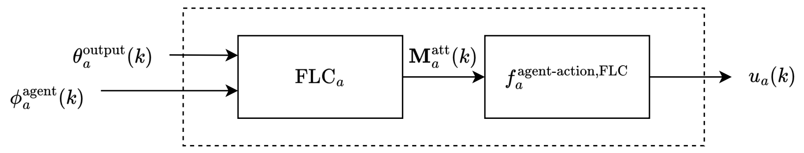

$\theta_a^{\text{output}}(k)$
$\phi_a^{\text{agent}}(k)$
FLC_a
$\mathbf{M}_a^{\text{att}}(k)$
$f_a^{\text{agent-action,FLC}}$
$u_a(k)$

Figure 3.7: FLC local controller for agent a

The FLC local controller is used as the local controller in the MPFC supervisory controller architecture. The architecture is shown in Figure 3.7, where the input to the local controller is the set of FLC output parameters that is determined via the supervisory MPC layer for the agent, $\boldsymbol{\theta}_a^{\text{output}}$, and the agent states, ϕ_a^{agent}. The FLC then calculates the attraction map, $\boldsymbol{M}^{\text{att}}(a,k)$, which is input to the agent action algorithm (Algorithm 3). The output of the agent action algorithm is then the set of control inputs to the agent model, u_a.

FLC Model

We define a Type-1 TSK (Takagi-Sugeno-Kang) FIS (Fuzzy Inference System) for the FLC, with one output, $\boldsymbol{M}^{\text{att}}(a,k)$, which is a matrix representing the attraction of each cell in the search map for an agent, a:

$$\boldsymbol{M}^{\text{att}}(a,k) = \text{FLC}_a\left(\boldsymbol{I}_a(k), \boldsymbol{\theta}_a^{\text{output}}(k)\right) \tag{3.13}$$

where FLC_a is the FLC for agent a, with inputs, $\boldsymbol{I}_a$, and output MFs, $\boldsymbol{\theta}_a^{\text{output}}$. The input MFs and rule base are made to be constant so they can be considered as part of the function FLC_a. The following inputs are selected for the FLC:

1. The **response time map**, $\boldsymbol{M}^{\text{response}}(a,k)$, is the time required for agent a at time step k to reach each cell in the search map (Equation 3.14), where $\boldsymbol{M}^{\text{response}}(a,k) \in [0,1]$.
2. The **priority map**, $\boldsymbol{M}^{\text{priority}}(k)$, is the priority of each cell in the search map (Equation 3.17), representing the priority of each cell for agent visitation at simulation time step k, where $\boldsymbol{M}_m^{\text{priority}}(k) \in [0,1]$.
3. The **fire risk time map**, $\boldsymbol{M}^{\text{fire-risk}}$, is an input representing the relative risk of cells to nearby active fires (Algorithm 2).

4. The **scan map**, $\boldsymbol{M}^{\text{scan}}$, is a measure of the certainty of the system on the values of the states measured in each cell (Equation 3.9), where $\boldsymbol{M}^{\text{scan}} \in [0, 1]$.

For each agent a, the response time map $\boldsymbol{M}^{\text{response}}(a, k)$ is calculated for every cell $\boldsymbol{s}_m$ in the map:

$$\boldsymbol{M}_m^{\text{response}}(a, k) = \frac{f^{\text{response}}(\boldsymbol{s}_a^{\text{location}}, \boldsymbol{s}_a^{\text{target}}, \boldsymbol{s}_m, \boldsymbol{t}^{\text{scan}})}{t^{\text{response,max}}}, \tag{3.14}$$

where f^{response} is given by:

$$\begin{aligned} &f^{\text{response}}(\boldsymbol{s}_a^{\text{location}}, \boldsymbol{s}_a^{\text{target}}, \boldsymbol{s}_m, \boldsymbol{t}^{\text{scan}}, \boldsymbol{\tau}_m^{\text{task}}) \\ &= \begin{cases} \boldsymbol{t}_a^{\text{travel}} + \boldsymbol{t}_a^{\text{scan}} + t^{\text{travel}}(\boldsymbol{s}_a^{\text{target}}, \boldsymbol{s}_m) & \text{if } \boldsymbol{\tau}_a^{\text{task}} = 0, \\ \boldsymbol{t}_a^{\text{scan}} + t^{\text{travel}}(\boldsymbol{s}_a^{\text{target}}, \boldsymbol{s}_m), & \text{if } \boldsymbol{\tau}_a^{\text{task}} = 1. \end{cases} \end{aligned} \tag{3.15}$$

where $\boldsymbol{t}_a^{\text{travel}}$ is the remaining travel time of agent a and $t^{\text{travel}}(\cdot)$ is a function which returns the travel time for an agent between cells s_a^{target} and s_a^{location} (Equations 3.10 - 3.12).

The maximum response time, $t^{\text{response,max}}$, is defined as the longest time an agent would require to travel between any two cells in the map and perform a scan:

$$t^{\text{response,max}} = \max_{\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3 \in \boldsymbol{s}} \big(\boldsymbol{t}_a^{\text{travel}}(\boldsymbol{s}_1, \boldsymbol{s}_2) + t^{\text{scan}} + \boldsymbol{t}_a^{\text{travel}}(\boldsymbol{s}_2, \boldsymbol{s}_3)\big). \tag{3.16}$$

The priority map is set as the predicted victim map defined in Equation 3.26:

$$\boldsymbol{M}^{\text{priority}}(k) = \hat{\boldsymbol{M}}^{\text{c,victim}}(k) \tag{3.17}$$

The *fire risk time map* parameter is given by Algorithm 2. The fire risk time map is a non-dimensional parameter that indicates the risk of each cell catching fire, given its proximity to an active fire. This parameter was selected as an input to the FLC over the *downwind map* used in the objective function evaluation (Equation 3.19), as when both were tested, the fire risk time demonstrated superior performance. The advantage of the *fire risk time map* is that it is non-linear while the *downwind map* is linear.

In Algorithm 2, we initialise $\boldsymbol{M}^{\text{fire-risk}}$ with a high value so that any cells outside of the maximum radius considered are considered not at risk of fire by the attraction calculation (Equation 3.13).

The choice of input parameters for the FLC is important as these inputs allow the agent to tune its behaviour to react to them. Therefore, it is necessary to provide the correct set of inputs such that the desired agent behaviour can be achieved. Several other input parameters were also explored when designing the FLC, including the *distance of each cell from other agents*, which could have allowed agents to manage the distance between each other, and the *cell coordinates*, which could allow agents to tune behaviour according to the cell geographical location.

A standard structure for the FLC is defined, which is used for all simulations in Section 4. We set the number of input MFs, $n^{\text{MF,in}} = 3$, the number of output MFs, $n^{\text{MF,out}} = 3$. All input MFs are set as triangular MFs and initialised with linear spacing across the range of the input, as shown in Table 3.3, where the parameters represent the vertices of each of the MFs. This results in the full set of input MFs shown in Figure 3.8.

Name	Type	Parameters
low	Triangular	[0, 0, 0.5]
medium	Triangular	[0, 0.5, 1]
high	Triangular	[0.5, 1, 1]

Table 3.3: MF parameters for FLC inputs.

Algorithm 2 Fire Time Risk Calculation

```
Input: Fire map $M^{\text{fire}}$, wind velocity $v^{\text{wind}}$
Output: Fire time risk map $M^{\text{fire-risk}}$
Initialise $M^{\text{fire-risk}} \leftarrow \mathbf{100}$ for all cells
for each cell, $m$ do
    for $r \in \{1, 2, 3\}$ do
        Get neighbours in radius $r$ around $m$
        Extract valid neighbours within grid bounds
        Apply fire rules:
        if $v^{\text{wind}} = 0$ then
            if any neighbour in radius $1$ has $M^{\text{fire}}_m = 3$ then
                $M^{\text{fire-risk}}_m \leftarrow 0$
            else if any neighbour in radius $1$ has $M^{\text{fire}}_m = 2$ then
                $M^{\text{fire-risk}}_m \leftarrow \min(M^{\text{fire-risk}}_m, 2)$
            else if any neighbour in radius $2$ has $M^{\text{fire}}_m = 3$ then
                $M^{\text{fire-risk}}_m \leftarrow \min(M^{\text{fire-risk}}_m, 2)$
            else if any neighbour in radius $2$ has $M^{\text{fire}}_m = 2$ then
                $M^{\text{fire-risk}}_m \leftarrow \min(M^{\text{fire-risk}}_m, 4)$
            else if any neighbour in radius $3$ has $M^{\text{fire}}_m = 3$ then
                $M^{\text{fire-risk}}_m \leftarrow \min(M^{\text{fire-risk}}_m, 4)$
            else if any neighbour in radius $3$ has $M^{\text{fire}}_m = 2$ then
                $M^{\text{fire-risk}}_m \leftarrow \min(M^{\text{fire-risk}}_m, 6)$
            end if
        else if $v^{\text{wind}} \leq 5$ then
            if any neighbour in radius $\leq 2$ has $M^{\text{fire}}_m = 3$ then
                $M^{\text{fire-risk}}_m \leftarrow 0$
            else if any neighbour in radius $\leq 2$ has $M^{\text{fire}}_m = 2$ then
                $M^{\text{fire-risk}}_m \leftarrow \min(M^{\text{fire-risk}}_m, 2)$
            end if
        else if $v^{\text{wind}} > 5$ then
            if any neighbour in radius $\leq 3$ has $M^{\text{fire}}_m = 3$ then
                $M^{\text{fire-risk}}_m \leftarrow 0$
            else if any neighbour in radius $1$ has $M^{\text{fire}}_m = 2$ then
                $M^{\text{fire-risk}}_m \leftarrow \min(M^{\text{fire-risk}}_m, 2)$
            end if
        end if
    end for
end for
```

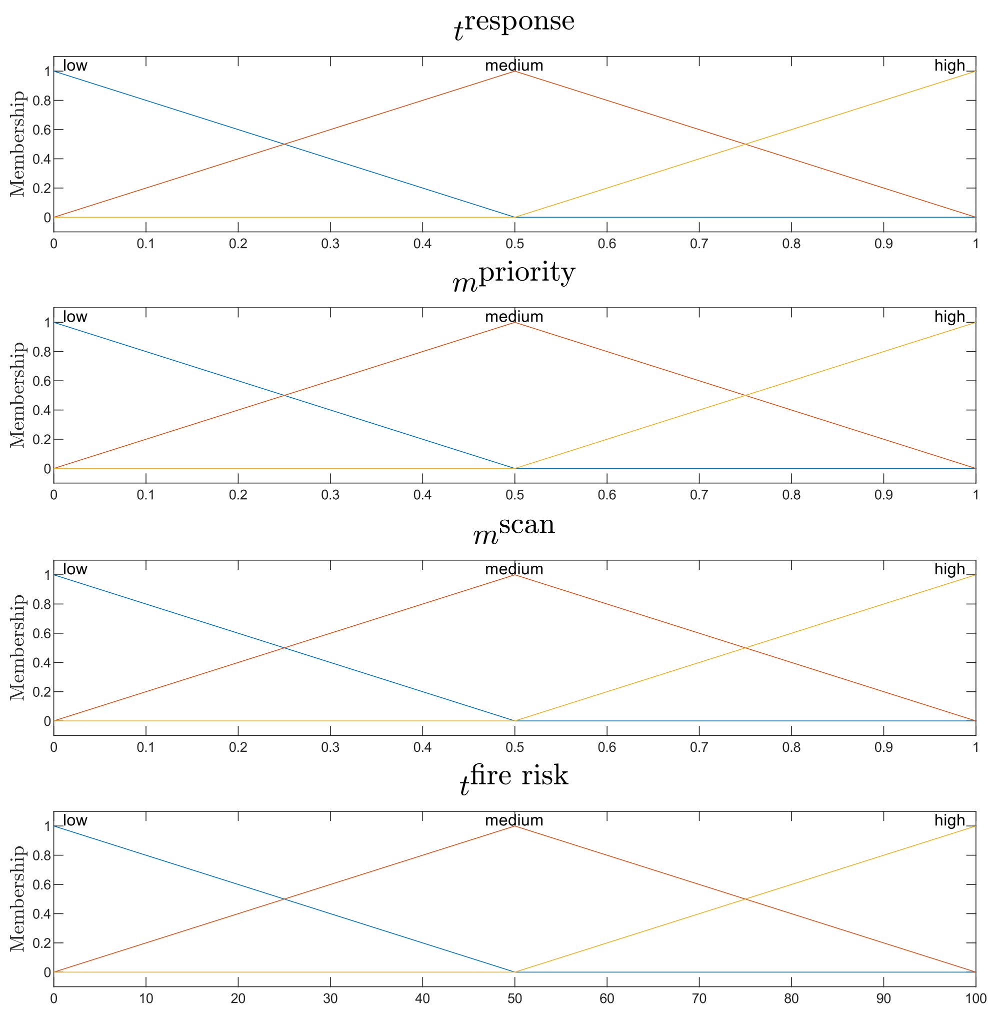


Figure 3.8: FIS Input MFs

The rule base is then defined manually according to the assumption that the attraction of cells should be higher for a lower $M^{\text{response}}(a,k)$, lower $M^{\text{fire-risk}}$, lower M^{scan}, and higher $M^{\text{priority}}(k)$ (Appendix A).

The output MFs are chosen to be linear and are initialised as shown in Table 3.4 and Figure 3.9.

Name	Type	Parameters (coefficients)
low	linear	$-1, 1, -1, -1, 0$
medium	linear	$-1, 1, -1, -1, 0.5$
high	linear	$-1, 1, -1, -1, 1$

Table 3.4: Default output MF parameters

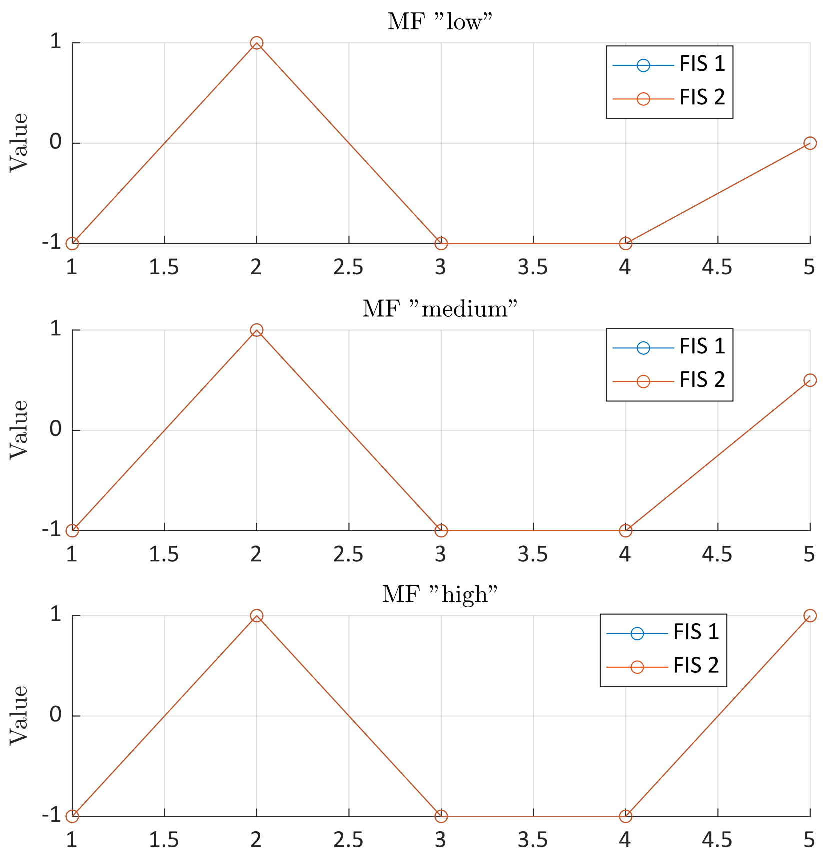


Figure 3.9: Parallel coordinate plot of default FIS Output MFs

Figure 3.10 displays an example of the inputs to and resulting output of the FLC for an agent a. In the example, the agent spatial information of the disaster environment is represented in a (4×4) grid. We can see from the *fire-risk* input, $M^{\text{fire-risk}}$, that there are active fires in the centre of the disaster environment The current agent position in cell $(2, 2)$ is visible in $M^{\text{response}}(a, k)$. In the output $M^{\text{att}}(a, k)$ it can be seen that the next cell the agent will prioritise will be cell $(3, 2)$. Inspecting the inputs, this does not appear to be the logical cell to prioritise, as the cell already has a high scan certainty. Under these conditions, MPFC may conclude that the FLC should raise the weighting of scan certainty and reduce the influence of response time, potentially causing the agent to select cell $(3, 2)$ as its next target instead.

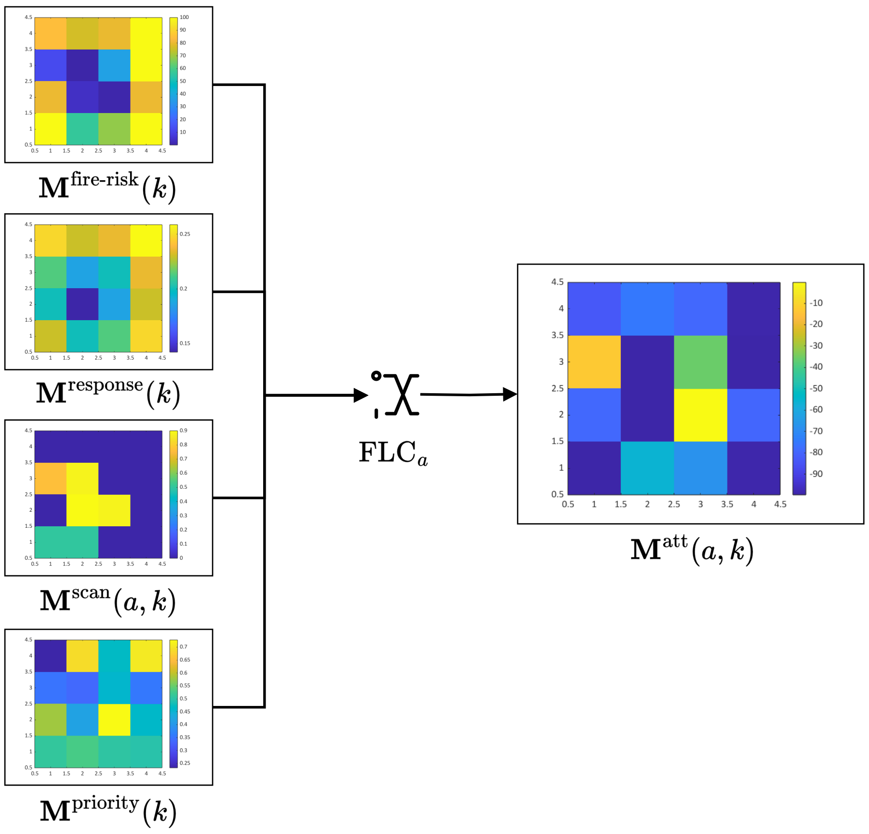

$\mathbf{M}^{\text{fire-risk}}(k)$
$\mathbf{M}^{\text{response}}(k)$
$\mathbf{M}^{\text{scan}}(a,k)$
$\mathbf{M}^{\text{priority}}(k)$
FLC_a
$\mathbf{M}^{\text{att}}(a,k)$

Figure 3.10: Inputs and Outputs of FLC

FLC Agent Action Model

We define the agent action algorithm $f^{\text{agent-action,FLC}}$, which executes the agents' tasks. The function operates as follows:

Algorithm 3 Agent Action Algorithm for MPFC/FLC, $f^{\text{agent-action,FLC}}$

```
Require: $M^{\text{att}}(a,k)$, $s_a^{\text{target}}$, $\tau_a^{\text{task}}(k)$, $t_a^{\text{scan}}$, $t_a^{\text{travel}}$, $k$
 1: if $\tau_a^{\text{task}}(k) = 0$ and $t_a^{\text{travel}} > 0$ then
 2:     $t_a^{\text{travel}}(k+1) \leftarrow t_a^{\text{travel}}(k) - \Delta t$
 3: else if $\tau_a^{\text{task}}(k) = 0$ and $t_a^{\text{travel}} \leq 0$ then
 4:     $\tau_a^{\text{task}}(k) \leftarrow 1$
 5:     $s_a^{\text{location}} \leftarrow s_a^{\text{target}}$
 6:     $t_a^{\text{travel}} \leftarrow t^{\text{travel}}(s_a^{\text{location}}, s_a^{\text{target}}(q))$
 7: else if $\tau_a^{\text{task}}(k) = 1$ and $t_a^{\text{scan}}(k) > 0$ then
 8:     $t_a^{\text{scan}}(k+1) \leftarrow t_a^{\text{scan}}(k) - \Delta t$
 9: else if $\tau_a^{\text{task}}(k) = 1$ and $t_a^{\text{scan}}(k) \leq 0$ then
10:     $\tau_a^{\text{task}}(k) \leftarrow 0$
11:     $s_a^{\text{target}} \leftarrow \arg\max M^{\text{att}}(a,k)$
12:     $t_a^{\text{scan}}(k) \leftarrow t^{\text{scan-cell}}$
13: end if
```

where $\arg\max M^{\text{att}}(a,k)$ returns the index of the cell with the maximum value in the attraction map $M^{\text{att}}(a,k)$. An agent task of $\tau_a^{\text{task}}(k) = 0$ represents travel while an agent task of $\tau_a^{\text{task}}(k) = 1$ represents scanning. At each simulation time step, this algorithm reduces the remaining time for the agent to complete its current task by the simulation time step size, Δt. If an agent completes a travel task, the agent location is updated and the agent travel time is reset for the next target cell. Otherwise, when an agent completes a travel task, the agent scan map is updated and the agent scan time is reset to $t^{\text{scan-cell}}$. The algorithm then assigns the next task for the agent.

3.3.2. MPC Local Controller

The MPC local controller is to execute the agent actions for the MPC supervisory controller. This consists of a simple function which directs the agent to travel to and scan each cell in the agent target vector, s_a^{target}. We define the agent action function $f^{\text{agent-action,MPC}}$, which updates the agent's task and position.

Algorithm 4 Agent Action Algorithm for MPC, $f^{\text{agent-action,MPC}}$

```
Require: $s_a^{\text{target}}$, $\tau_a^{\text{task}}(k)$, $t_a^{\text{scan}}$, $t_a^{\text{travel}}$, $q = 1$, $k$
 1: if $\tau_a^{\text{task}}(k) = 0$ and $t_a^{\text{travel}} > 0$ then
 2:     $t_a^{\text{travel}}(k+1) \leftarrow t_a^{\text{travel}}(k) - \Delta t$
 3: else if $\tau_a^{\text{task}}(k) = 0$ and $t_a^{\text{travel}} \leq 0$ then
 4:     $\tau_a^{\text{task}}(k) \leftarrow 1$
 5:     $s_a^{\text{location}} \leftarrow s_a^{\text{target}}(q)$
 6:     $q \leftarrow q + 1$
 7:     $t_a^{\text{travel}} \leftarrow t^{\text{travel}}(s_a^{\text{location}}, s_a^{\text{target}}(q))$
 8: else if $\tau_a^{\text{task}}(k) = 1$ and $t_a^{\text{scan}}(k) > 0$ then
 9:     $t_a^{\text{scan}}(k+1) \leftarrow t_a^{\text{scan}}(k) - \Delta t$
10: else if $\tau_a^{\text{task}}(k) = 1$ and $t_a^{\text{scan}}(k) \leq 0$ then
11:     $q = q + 1$
12:     $\tau_a^{\text{task}}(k) \leftarrow 0$
13:     $t_a^{\text{scan}}(k) \leftarrow t^{\text{scan-cell}}$
14: end if
```

The MPC controller algorithm functions in a similar way to the MPFC algorithm, however when assigning the next target cell, this is taken from the target queue, s_a^{target}, by incrementing the queue position index, q. The target queue is output from the MPC optimisation and is the list of cell indexes for the agent to scan over the MPC prediction horizon.

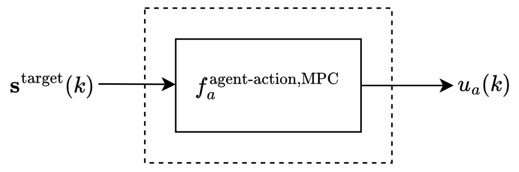

$\mathbf{s}^{\text{target}}(k)$
$f_a^{\text{agent-action,MPC}}$
$u_a(k)$

Figure 3.11: MPC Local Controller

Figure 3.11 shows the MPC local controller architecture. The input to the local controller is the target queue, and the output of the local controller is the set of control inputs to the agent model, u_a.

3.4. Supervisory Controller Model

Based on the mathematical models built for the supervisory controllers in section 2.3, we define several implementations of an MPFC and MPC supervisory controller.

3.4.1. Global Objective Function

We then define the global objective function as each simulation time step as the sum over the search area of a function of the search map states, $f^{\text{obj}}(\phi^{\text{search}}(s_m, k))$:

$$J(k) = \sum_{m=1}^{n^c} f^{\text{obj}}(\phi^{\text{search}}(s_m, k)) \tag{3.18}$$

where N is the total number of time steps in the simulation window.

We formulate the objective function to maximise the information measured by the agents for the maximum number of victims possible weighted against their exposure to fire. This results in two components to the objective function.

$$f^{\text{obj}}(\phi_m^{\text{search}}(k)) = \boldsymbol{M}_m^{\text{c,victim}}(k) \cdot \boldsymbol{M}_m^{\text{scan}}(k) \cdot \left(c^{\text{obj},1} - c^{\text{obj},2} \cdot \boldsymbol{M}_m^{\text{c,downwind}}(k)\right) \tag{3.19}$$

where $\boldsymbol{M}^{\text{c,downwind}}$ is coarsened from the downwind map, $\boldsymbol{M}^{\text{downwind}}$, given in Algorithm 5. $c^{\text{obj},1}$ is a constant to weight the non-fire proximity factor and $c^{\text{obj},2}$ is a constant to weight the fire proximity factor.

The first component contains $\boldsymbol{M}_m^{\text{c,victim}}(k) \cdot \boldsymbol{M}_m^{\text{scan}}(k)$ which multiplies cell scan certainty by the victim map to prioritise maximisation of the cell scan certainty per victim. The second component contains $\boldsymbol{M}_m^{\text{c,downwind}}(k) \cdot \boldsymbol{M}_m^{\text{c,victim}}(k) \cdot \boldsymbol{M}_m^{\text{scan}}(k)$ which de-prioritises cells which are further away from active fires and prioritises cells which are closer to active fires.

This objective function incorporates the victim map, which the agents can measure; the search map, which they can directly influence by scanning cells; and the downwind map, an uncertain dynamic state that the agents can predict.

The *downwind map* is a non-dimensional parameter which represents the relative proximity of each cell in the disaster environment to an active or catching fire, normalised in the range $[0, 1]$. This is constructed in a similar way to the fire spread probability parameter (Equation 3.3) by building direction and distance parameters, but is done for the entire disaster environment for each fire and then combined into a single parameter.

The downwind direction parameter is calculated according to Equation 3.20:

$$\boldsymbol{M}_m^{\text{downwind-direction}} = \exp\left(v^{\text{wind}}\left(c^{\text{wm1}} + c^{\text{wm2}}\left(\cos(\theta^{\text{wind}} - \theta^{\text{fire}})(\boldsymbol{s}_m) - 1\right)\right)\right) \tag{3.20}$$

We define the angle $\theta(\boldsymbol{s}_m)^{\text{fire}}$ as the angle between the cell $\boldsymbol{s}_m$ in the disaster environment and the location of an active fire at cell $\boldsymbol{s}_f$. Here, the angle is computed using the two-argument arctangent function, $\operatorname{atan2}$, which determines the angle in the correct quadrant.[2]

$$\theta(\boldsymbol{s}_m)^{\text{fire}} = \operatorname{atan2}(\boldsymbol{s}_m(j) - \boldsymbol{s}_f(q), \boldsymbol{s}_m(i) - \boldsymbol{s}_f(p)) \tag{3.21}$$

The above equation calculates $\theta(\boldsymbol{s}_m)^{\text{fire}}$ for all cells $\boldsymbol{s}_m$ in the disaster environment grid with indices $i \in \{0, 1, \cdots, n^{\text{x,env}}\}$ and $j \in \{0, 1, \cdots, n^{\text{y,env}}\}$, for an active fire located at $\boldsymbol{s}_f$.

The downwind distance map for each cell in the environment is given by:

$$\boldsymbol{M}_m^{\text{downwind-distance}} = 1 - \frac{\sqrt{(\boldsymbol{s}_m(i) - \boldsymbol{s}_f(p))^2 + (\boldsymbol{s}_m(j) - \boldsymbol{s}_f(q))^2}}{\sqrt{(n^{\text{x,env}})^2 + (n^{\text{y,env}})^2}} \tag{3.22}$$

The downwind map is then generated according to Algorithm 5.

The MPFC and MPC predict the objective function over the prediction horizon. The dummy variable k' is introduced to represent each future simulation time step within the prediction horizon. The predicted objective function is then given by:

$$\hat{J}(k') = \sum_{m=1}^{n^c} \hat{\boldsymbol{M}}_m^{\text{c,victim}}(k') \cdot \boldsymbol{M}_m^{\text{scan}}(k') \cdot \left(c^{\text{obj},1} - c^{\text{obj},2} \cdot \hat{\boldsymbol{M}}_m^{\text{c,downwind}}(k')\right) \tag{3.23}$$

where $\hat{\boldsymbol{M}}^{\text{c,victim}}(k')$ and $\hat{\boldsymbol{M}}_m^{\text{c,downwind}}(k')$ are estimated by the predictive controller.

[2] $\operatorname{atan2}(y, x)$ is a variant of the inverse tangent function that takes into account the signs of both x and y to return an angle in the range $[-\pi, \pi]$.

Algorithm 5 Downwind Map Algorithm, f^{downwind}

```
Require: $M^{\text{fire}}$, $n^{\text{x,env}}$, $n^{\text{y,env}}$, $c^{\text{wm1}}$, $c^{\text{wm2}}$, $\theta^{\text{wind}}$, $v^{\text{wind}}$
Ensure: $M^{\text{downwind}}$
 1: Initialise $M^{\text{downwind}}$ as a zero matrix of size $n^{\text{x,env}} \times n^{\text{y,env}}$
 2: for $p = 1$ to $n^{\text{x,env}}$ do
 3:     for $q = 1$ to $n^{\text{y,env}}$ do
 4:         if $M_f^{\text{fire}} = 2$ or $M_f^{\text{fire}} = 3$ then                ▷ Active or burning fire
 5:             Calculate $M^{\text{downwind-direction}}$ and $M^{\text{downwind-distance}}$ using Equations 3.20 and Equation 3.22
 6:             $M^{\text{downwind}} = \max\left(M^{\text{downwind}}, M^{\text{downwind-direction}} \cdot M^{\text{downwind-distance}}\right)$
 7:         end if
 8:     end for
 9: end for
10: $M^{\text{downwind}} = 1 - M^{\text{downwind}}$                ▷ Invert the downwind effect
11: return $M^{\text{downwind}}$
```

3.4.2. Prediction Horizon

The prediction horizon of the supervisory controllers is illustrated in Figure 3.12. As mentioned in assumption AS6, the supervisory controller prediction is assumed to be instantaneous in relation to the system model and does not need to be scheduled ahead of time. Therefore, at the beginning of each MPC step at the current time step, the supervisory controller prediction is executed and used to update the control parameters at that time step. In reality, due to the computational resources required to perform the prediction and optimisation, this would need to be scheduled ahead of the planned time to update the control parameters and an additional time constraint would be placed upon the optimisation.

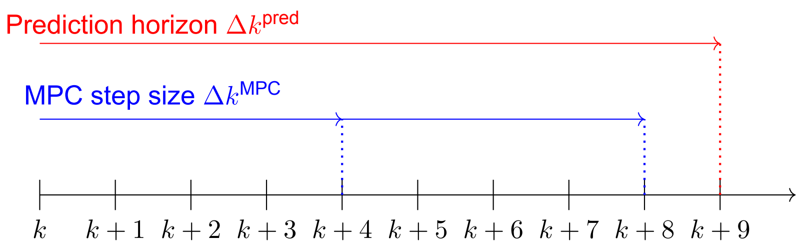

Prediction horizon Δk^{pred}
MPC step size Δk^{MPC}
k
$k+1$
$k+2$
$k+3$
$k+4$
$k+5$
$k+6$
$k+7$
$k+8$
$k+9$

Figure 3.12: MPC Controller Sequencing

Figure 3.12 illustrates the sequencing of the MPFC controller. In general, MPFC may be configured to use any form of sequencing which is desired for the various design criteria. For simplification, we fix the values for all of the sequencing design criteria instead of using variable values, and therefore use regular intervals for MPFC scheduling.

Scheduling parameters include the prediction horizon, Δk^{pred}, is the number of time steps ahead of the current time step over which the prediction is performed; the MPC step size, Δk^{MPC}, which is the number of time steps in each MPC step. At the beginning of each MPC step, MPFC is able to update the FLC with a new set of parameters. The number of MPC steps, n^{MPC}, is the number of MPC steps over the prediction horizon. Finally, we set the MPFC frequency, which is the number of time steps between each execution of the MPFC controller, equal to the MPFC step size.

In the example given in Figure 3.12, $\Delta k^{\text{pred}} = 9$, $\Delta k^{\text{MPC}} = 3$ and $\Delta n^{\text{MPC}} = 2$. At the beginning of each MPC step, a new set of output parameters, $\boldsymbol{\theta}^{\text{output}}(k)$, are output from the controller. In the illustration, two sets of FLC parameters will be output from the optimisation, $\boldsymbol{\theta}^{\text{output}}(k)$ for the current MPC step, and $\boldsymbol{\theta}^{\text{output}}(k + \Delta k^{\text{MPC}})$ for the next MPC step. This is explained further in section 3.4.4.

3.4.3. Supervisory Controller Prediction Model

In this section, we describe the models used in the prediction step of the predictive controllers.

Two top-level assumptions shape the formation of the prediction model. Firstly, as defined in assumption AS9, we assume that at any time step k, the predictive controller has perfect knowledge of all environment and agent states at that time step, acquired, for example, via satellite imagery and advanced models and datasets, apart from the victim locations which is only known once measured by an agent. Secondly, as defined in AS8, we assume that the predictive controller has a perfect representative model of the system, so that outside of uncertainties, it can predict future states of the system perfectly.

Prediction Modes Formulation

We define two simple generic approaches for estimating uncertain future states by the predictive controllers, which we term *prediction modes*. More advanced prediction modes are used in methods such as *Stochastic MPC*, where probabilistic uncertainty models are inserted into the prediction of system dynamics, but these are not implemented within the scope of this paper.

The **probability threshold** prediction mode assumes the most probable state transition always occurs. For a system with state vector $\boldsymbol{\phi}(k)$ and function $\mathcal{F}$, the predicted state at the next time step $k+1$ is given by:

$$\hat{\boldsymbol{\phi}}(k+1) = \mathcal{F}(\boldsymbol{\phi}(k)) \tag{3.24}$$

where $\mathcal{F}$ is a deterministic function that maps the current state $\boldsymbol{\phi}(k)$ to the most likely next state $\hat{\boldsymbol{\phi}}(k+1)$.

The **exact** prediction mode assumes perfect prediction of future states:

$$\hat{\boldsymbol{\phi}}(k+1) = \boldsymbol{\phi}(k+1) \tag{3.25}$$

This method is introduced to provide a baseline for the best possible prediction against which to measure the performance of other prediction modes. The simulation results are presented in section 4.4.1.

Victim Map Estimation

As mentioned above, the victim map is one of the parameters unavailable to the multi-agent system until measured by the agents.

The victim map estimation is given by:

$$\hat{\boldsymbol{M}}_m^{\text{c,victim}}(k) \begin{cases} \boldsymbol{M}_m^{\text{c,victim}}(k) & \text{if the cell is scanned} \\ c^{\text{population}} \cdot A^{\text{search}} \cdot \boldsymbol{M}_m^{\text{c,building}} & \text{if the cell is not scanned} \end{cases} \tag{3.26}$$

where $c^{\text{population}}$ is the average population density of the disaster environment and A^{search} is the cell area.

Fire Model Estimation

The fire model parameters must also be estimated due to the uncertain parameters present in the fire model.

The downwind map estimation is given by Algorithm 5:

$$\hat{\boldsymbol{M}}^{\text{c,downwind}}(k') = f^{\text{downwind}}(\hat{\boldsymbol{M}}^{\text{c,fire}}(k')) \tag{3.27}$$

where:

$$\hat{\boldsymbol{M}}^{\text{c,fire}}(k') = f^{\text{coarsen}}(\hat{\boldsymbol{M}}^{\text{fire}}(k')) \tag{3.28}$$

where f^{coarsen} represents Equation 3.7 and the estimated fire map, $\boldsymbol{M}^{\text{fire}}(k')$, is given by Equation 3.1.

3.4.4. Supervisory Controller Architectures

As mentioned previously, we define two types of supervisory controllers: MPFC and MPC. Additionally, for each controller, we define a centralised architecture, in which a single predictive controller optimises the actions of all agents in the system, and a decentralised architecture, in which a separate predictive controller optimises the actions of each agent.

Centralised MPFC

In the centralised MPFC supervisory controller architecture (Figure 3.13), a single MPC is used to tune the output parameters of all agents in the system. This architecture is suitable for cases where agents' behaviour is coupled, as the interaction of agents can be taken into account in the optimisation, but has a factor n^{a} more optimisation variables than the decentralised architecture.

The optimisation is given by:

$$
\begin{aligned}
&\max_{\boldsymbol{\theta}^{\mathsf{output}}(k)} \sum_{k'=k}^{k+\Delta t^{\mathsf{pred}}} \hat{J}(k') \\
\text{where}\quad &\boldsymbol{\theta}^{\mathsf{output}}(k) = [\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_{n^{\mathsf{a}}}]^\top \\
&\boldsymbol{\theta}_a = \left[\theta_{a,1,1,1}, \dots, \theta_{a,n^{\mathsf{MPC}},n^{\mathsf{MF,out}},n^{\mathsf{MF,in}}+1}\right]^\top \quad \forall\, a \in \{1, \dots, n^{\mathsf{a}}\}, \\
&\theta_{l,m,n}(k) \in [\theta^{\mathsf{min}}, \theta^{\mathsf{max}}] \\
&\quad \forall\, a \in \{1, \dots, n^{\mathsf{a}}\}, \quad l \in \{1, \dots, n^{\mathsf{MPC}}\}, \quad m \in \{1, \dots, n^{\mathsf{MF,out}}\}, \\
&\quad n \in \{1, \dots, n^{\mathsf{MF,in}} + 1\}.
\end{aligned}
\tag{3.29}
$$

where $\boldsymbol{\theta}^{\mathsf{output}}$ is the vector of FLC output parameters across all agents, n^{MPC} is the number of MPC steps, $n^{\mathsf{MF,out}}$ is the number of output MFs, $n^{\mathsf{MF,in}}$ is the number of input MFs, θ^{min} is the minimum constraint on the FLC output parameters, and θ^{max} is the maximum constraint on the FLC output parameters. The number of optimisation variables is given by: $n^{\mathsf{a}} \cdot n^{\mathsf{MPC}} \cdot n^{\mathsf{MF,out}} \cdot n^{\mathsf{MF,in}} + 1$.

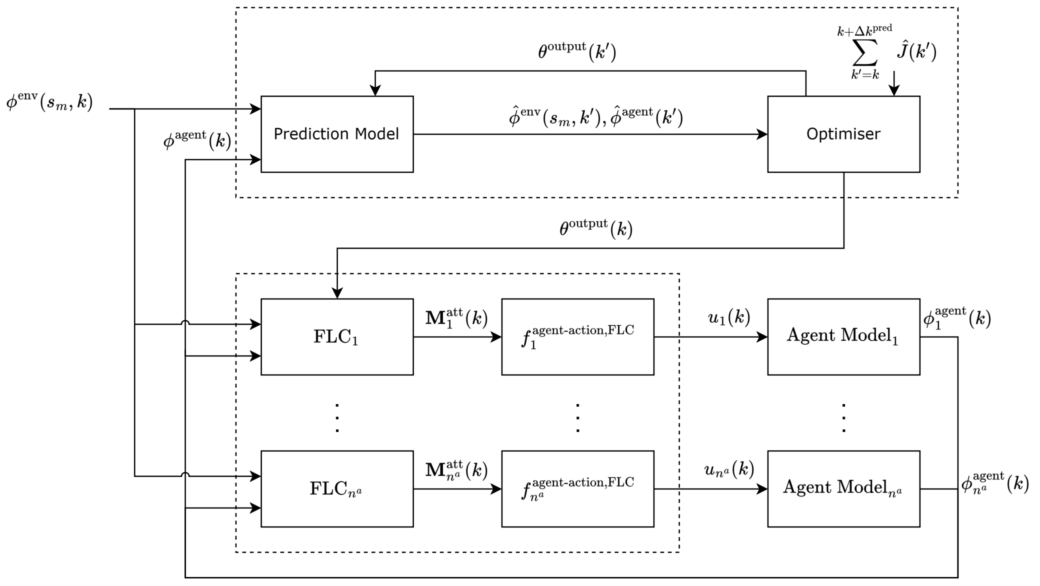

$\phi^{\text{env}}(s_m,k)$
$\theta^{\text{output}}(k')$
$\sum_{k'=k}^{k+\Delta k^{\text{pred}}} \hat{J}(k')$
$\phi^{\text{agent}}(k)$
Prediction Model
$\hat{\phi}^{\text{env}}(s_m,k'), \hat{\phi}^{\text{agent}}(k')$
Optimiser
$\theta^{\text{output}}(k)$
FLC$_1$
$\mathbf{M}_1^{\text{att}}(k)$
$f_1^{\text{agent-action,FLC}}$
$u_1(k)$
Agent Model$_1$
$\phi_1^{\text{agent}}(k)$
FLC$_{n^a}$
$\mathbf{M}_{n^a}^{\text{att}}(k)$
$f_{n^a}^{\text{agent-action,FLC}}$
$u_{n^a}(k)$
Agent Model$_{n^a}$
$\phi_{n^a}^{\text{agent}}(k)$

Figure 3.13: Centralised MPFC architecture

The controller architecture is displayed in Figure 3.13. The predictive controller is shown in the upper dashed box, which outputs the set of optimisation variables, $\boldsymbol{\theta}^{\mathsf{output}}(k)$ across all FLCs, $[\mathsf{FLC}_1, \cdots, \mathsf{FLC}_{n^{\mathsf{a}}}]$. The local controllers are shown in the lower dashed box. Each local controller is composed of the FLC module and agent action algorithm module.

Decentralised MPFC

In the decentralised MPFC supervisory controller architecture (Figure 3.14), a separate MPC is used to tune the output parameters of each agent in the system. This architecture is suitable for cases where agents' behaviour is not strongly coupled. The optimisation is given by:

$$\begin{aligned}
&\text{For each } a \in \{1,\dots,n^{\mathsf{a}}\}, \quad \max_{\boldsymbol{\theta}_a^{\text{output}}(k)} \sum_{k'=k}^{k+\Delta t^{\text{pred}}} \hat{J}_a(k') \\
&\qquad \text{where} \quad \boldsymbol{\theta}_a^{\text{output}}(k) = \left[\theta_{a,1,1,1}, \theta_{a,1,1,2}, \dots, \theta_{a,n^{\text{MPC}},n^{\text{MF,out}},n^{\text{MF,in}}+1}\right]^\top, \\
&\qquad \theta_{a,l,m,n}(k) \in [\theta^{\text{min}}, \theta^{\text{max}}] \\
&\qquad\quad \forall l \in \{1,\dots,n^{\text{MPC}}\},\ m \in \{1,\dots,n^{\text{MF,out}}\},\ n \in \{1,\dots,n^{\text{MF,in}}+1\}.
\end{aligned} \tag{3.30}$$

where $\boldsymbol{\theta}_a^{\text{output}}$ is the vector of FLC output parameters across a single agent. The number of optimisation variables is given by: $n^{\text{MPC}} \cdot n^{\text{MF,out}} \cdot n^{\text{MF,in}} + 1$.

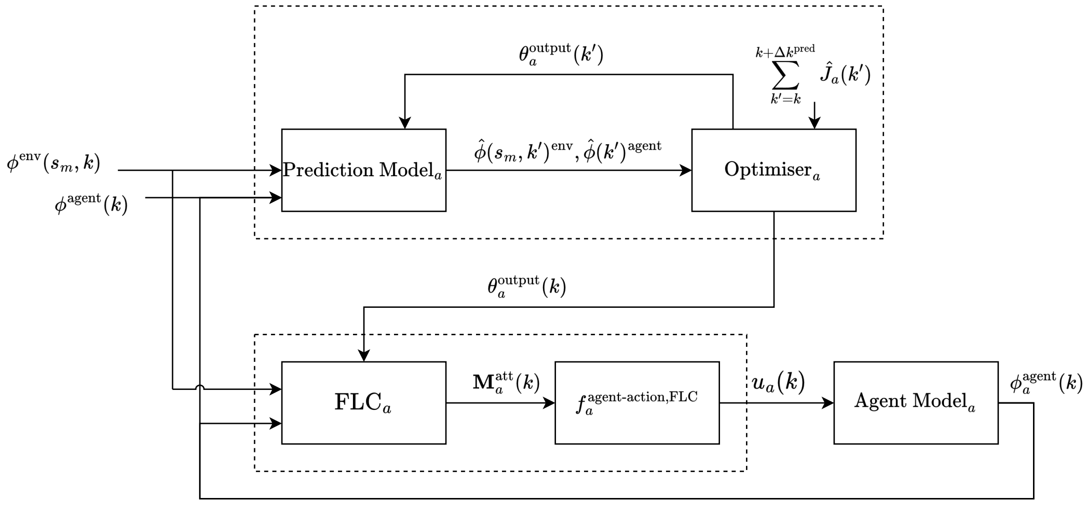

$\theta_a^{\text{output}}(k')$
$\sum_{k'=k}^{k+\Delta k^{\text{pred}}} \hat{J}_a(k')$
$\phi^{\text{env}}(s_m,k)$
$\phi^{\text{agent}}(k)$
Prediction Model$_a$
$\hat{\phi}(s_m,k')^{\text{env}}, \hat{\phi}(k')^{\text{agent}}$
Optimiser$_a$
$\theta_a^{\text{output}}(k)$
FLC$_a$
$\mathbf{M}_a^{\text{att}}(k)$
$f_a^{\text{agent-action,FLC}}$
$u_a(k)$
Agent Model$_a$
$\phi_a^{\text{agent}}(k)$

Figure 3.14: Decentralised MPFC architecture for a single agent

The decentralised MPFC controller architecture for a single agent a is displayed in Figure 3.14. The predictive controller is shown in the upper dashed box, which outputs the set of optimisation variables, $\boldsymbol{\theta}^{\text{output}}(k)$ for a single FLC, FLC$_a$, shown in the lower dashed box for the local controller.

Centralised MPC

Likewise, several architectures of MPC supervisory controllers are implemented as a baseline reference to compare against the performance of the MPFC supervisory controllers. In the centralised MPC supervisory controller architecture (Figure 3.15), a single MPC tunes the target cells of all agents over the prediction horizon. The optimisation is given by:

$$\begin{aligned}
&\max_{\boldsymbol{s}^{\text{target}}(k)} \sum_{k'=k}^{k+\Delta t^{\text{pred}}} \hat{J}(k') \\
&\text{where} \quad \boldsymbol{s}^{\text{target}}(k) = \left[\boldsymbol{s}_1^{\text{target}}, \dots, \boldsymbol{s}_{n^{\mathsf{a}}}^{\text{target}}\right]^\top, \\
&\boldsymbol{s}_a^{\text{target}}(k) = \left[\boldsymbol{s}_{a,1}^{\text{target}}, \dots, \boldsymbol{s}_{a,n^{\text{queue}}}^{\text{target}}\right]^\top \quad \forall a \in \{1,\dots,n^{\mathsf{a}}\}, \\
&\boldsymbol{s}_{a,q}^{\text{target}}(k)(1) \in \{1,\dots,n^{\text{x,search}}\}, \quad \forall a \in \{1,\dots,n^{\mathsf{a}}\},\ \forall q \in \{1,\dots,n^{\text{queue}}\}, \\
&\boldsymbol{s}_{a,q}^{\text{target}}(k)(2) \in \{1,\dots,n^{\text{y,search}}\}, \quad \forall a \in \{1,\dots,n^{\mathsf{a}}\},\ \forall q \in \{1,\dots,n^{\text{queue}}\}.
\end{aligned} \tag{3.31}$$

where s^{target} is the vector of target cell indexes over the prediction horizon across all agents. Each target cell index is a pair of cell indexes constrained as an integer in the range $\{0, 1, \cdots, n^{\text{x,search}}\}$ for the x-axis index and $\{0, 1, \cdots, n^{\text{y,search}}\}$ for the y-axis index. The number of optimisation variables is given by: $2 \cdot n^{\text{a}} \cdot n^{\text{queue}}$. Note that n^{queue} must be large enough such that the agent cannot run out of cells to scan between MPC steps, or will otherwise be stuck idling until the next MPC step.

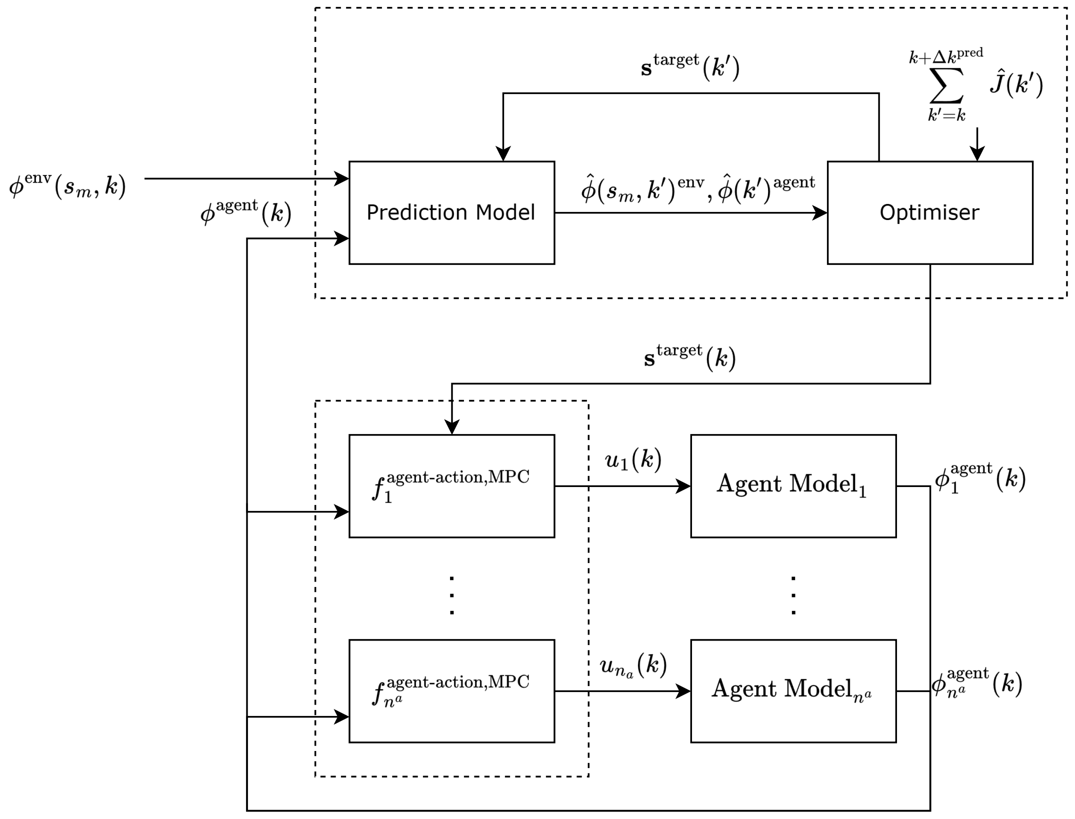

$\mathbf{s}^{\text{target}}(k')$
$\sum_{k'=k}^{k+\Delta k^{\text{pred}}} \hat{J}(k')$
$\phi^{\text{env}}(s_m, k)$
$\phi^{\text{agent}}(k)$
Prediction Model
$\hat{\phi}(s_m, k')^{\text{env}}, \hat{\phi}(k')^{\text{agent}}$
Optimiser
$\mathbf{s}^{\text{target}}(k)$
$f_1^{\text{agent-action,MPC}}$
$u_1(k)$
Agent Model$_1$
$\phi_1^{\text{agent}}(k)$
$f_{n^a}^{\text{agent-action,MPC}}$
$u_{n_a}(k)$
Agent Model$_{n^a}$
$\phi_{n^a}^{\text{agent}}(k)$

Figure 3.15: Centralised MPC architecture

The controller architecture for the centralised MPC is displayed in Figure 3.15. The predictive controller is shown in the upper dashed box, which outputs the set of optimisation variables, $s^{\text{target}}(k)$ across all agent local controllers. The output parameters, $s^{\text{target}}(k)$, are a list of target cells for each agent to scan over the prediction horizon. Due to the fixed MPC time step approach implemented in this paper, the list or target cells must be assigned a length equal to the maximum number of cells which may be visited within the prediction horizon. This is defined by the number of sequential cells each agent could scan over the prediction horizon to minimise travel time. The downside to this approach for the MPC is that if the optimal solution is not to scan sequential cells, there are many more optimisation variables than necessary, which can degrade the performance of the controller. An alternative approach that may improve performance would be to introduce a fixed number of cells in the target cells matrix and run a variable-time step MPC. However, it should be noted that the downside to this approach in a real-world implementation is that if the agents were to run out of target cells before the next MPC step, they would not have any decision-making capability in their local controller to continue performing tasks.

Decentralised MPC

A decentralised MPC supervisory controller architecture is also defined, where a separate MPC is used to tune the output parameters of each agent in the system. The optimisation is given by:

$$
\begin{aligned}
\text{For each } a \in \{1,\dots,n^{\text{a}}\}: \quad & \max_{\boldsymbol{s}_a^{\text{target}}(k)} \sum_{k'=k}^{k+\Delta t^{\text{pred}}} \hat{J}_a(k') \\
\text{where} \quad & \boldsymbol{s}_a^{\text{target}}(k) = \left[\boldsymbol{s}_{a,1}^{\text{target}}(k),\dots,\boldsymbol{s}_{a,n^{\text{queue}}}^{\text{target}}\right]^\top, \\
& \boldsymbol{s}_{a,q}^{\text{target}}(k)(1) \in \{1,\dots,n^{\text{x,search}}\} \quad \forall q \in \{1,\dots,n^{\text{queue}}\}, \\
& \boldsymbol{s}_{a,q}^{\text{target}}(k)(2) \in \{1,\dots,n^{\text{y,search}}\} \quad \forall q \in \{1,\dots,n^{\text{queue}}\}.
\end{aligned}
\tag{3.32}
$$

where $\boldsymbol{s}_a^{\text{target}}$ is the vector of target cells for agent a. The number of optimisation variables is given by: $2 \cdot n^{\text{queue}}$.

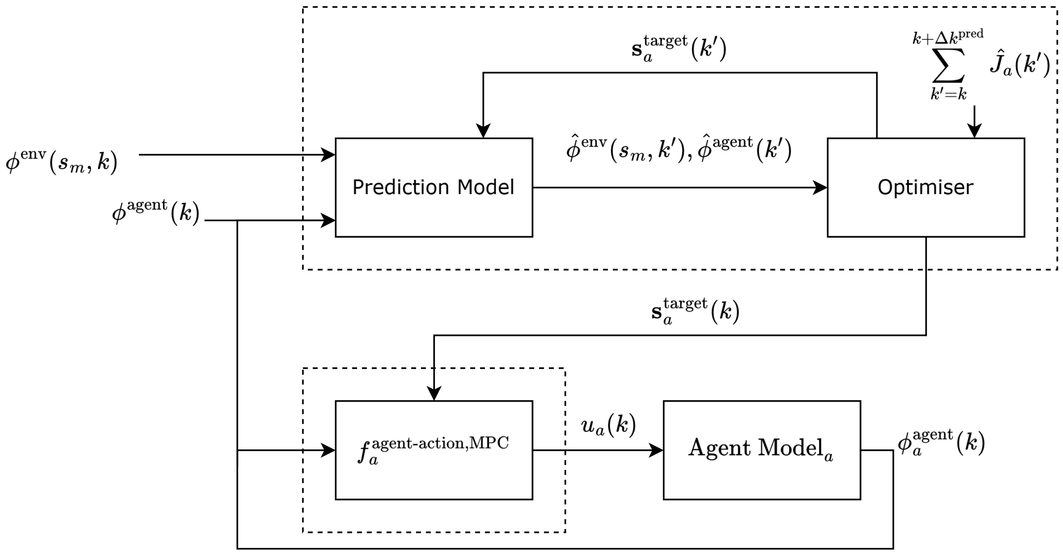

$\mathbf{s}_a^{\text{target}}(k')$
$\sum_{k'=k}^{k+\Delta k^{\text{pred}}} \hat{J}_a(k')$
$\phi^{\text{env}}(s_m,k)$
$\phi^{\text{agent}}(k)$
Prediction Model
$\hat{\phi}^{\text{env}}(s_m,k'), \hat{\phi}^{\text{agent}}(k')$
Optimiser
$\mathbf{s}_a^{\text{target}}(k)$
$f_a^{\text{agent-action,MPC}}$
$u_a(k)$
Agent Model$_a$
$\phi_a^{\text{agent}}(k)$

Figure 3.16: Decentralised MPC architecture

The decentralised MPC controller architecture for a single agent a is displayed in Figure 3.16. The predictive controller is shown in the upper dashed box, which outputs the set of optimisation variables, $s^{\text{target}}(k)$ for a local controller.

4 Results

The parameters selected to analyse the performance of the controller are the *global objective function*, $J(k)$ (Equation 3.18) and the *optimisation time*, t^{opt}, which is the time required for the predictive controller optimisation to complete. These parameters are recorded as time series data throughout each simulation.

Due to the probabilistic nature of the environment model and optimisation algorithm, the performance of the multi-agent system is inherently variable over repeated simulations. Therefore, we evaluate the mean values and confidence intervals on the performance metrics to quantify the expected performance and variability over multiple simulations. The objective function evaluated for each simulation, s, is denoted as J_s. The instantaneous mean objective function, $\overline{J}(k)$, over multiple simulations, n^{sim}, is given by:

$$\overline{J}(k) = \frac{1}{n^{\text{sim}}} \sum_{s=1}^{n^{\text{sim}}} J_s(k) \tag{4.1}$$

where n^{sim} is the total number of simulations.

Similarly, the instantaneous mean optimisation time, $\overline{t}^{\text{opt}}$, over multiple simulations is given by:

$$\overline{t}^{\text{opt}}(k) = \frac{1}{n^{\text{sim}}} \sum_{s=1}^{n^{\text{sim}}} t_s^{\text{opt}}(k) \tag{4.2}$$

We calculate 95% confidence intervals by taking the mean time series and adding/subtracting $1.96 \times$ SEM, where SEM is the standard error of the mean. We denote these intervals as $(\overline{J}_{0.025}(k), \overline{J}_{0.975}(k))$ and $(\overline{t}^{\text{opt}}_{0.025}, \overline{t}^{\text{opt}}_{0.975}(k))$, respectively.

To ensure a fair comparison between different controller architectures, we define a sequence of simulation seeds for each set of simulations. These seeds are used to initialise the random number generator for each controller architecture in each simulation, guaranteeing that the probabilistic environment variables remain consistent across equivalent simulations for each controller architecture.

To illustrate, if we set $n^{\text{sim}} = 5$ and simulate the system using both MPFC and MPC, the first simulation for each controller will use the first simulation seed, the second simulation will use the second simulation seed, and so on. This method allows us to directly compare the performance of different controllers under identical conditions.

4.1. Simulation Configuration

For each simulation, we rely on a dedicated set of configuration scripts to define all simulation parameters. The full set of configuration scripts alongside the software code can be found in the 4TU repository

[8] or the GitHub repository [9], and the table of configuration scripts for each simulation can be found in appendix B. The case study is implemented in MATLAB and all simulations are run on an **AMD Ryzen 5 3500U CPU**.

Within the simulation environment, we reformat the objective function so that a lower value is considered better (Equation 3.19), therefore all optimisations become minimisations (Equations 3.19, 3.29, 3.30, 3.31, and 3.32).

4.1.1. Global Simulation Parameters Configuration

A standard setup for the global simulation parameters is used across all simulations with the only exceptions being the simulation time, T; the MPC step size, Δk^{MPC}; and prediction horizon, $\Delta k^{\text{prediction}}$. The purpose of the various simulations performed in this case study is not to explore the influence of these parameters, even though they can significantly impact the overall performance of the system and the results of the simulation.

The simulation step size is set as $\Delta t = 15\,\mathrm{s}$, the coarsening factor is set as $c^{\text{coarsen}} = 5$, and the objective function constants are set as $c^{\text{obj},1} = 1$ and $c^{\text{obj},2} = 1$ (Equation 3.19).

4.1.2. Environment Model Configuration

The standard environment model configuration is defined as:

- Environment cell length, $\Delta x^{\text{env}} = \Delta y^{\text{env}} = 10\,\mathrm{m}$.
- Building occupancy map, $\boldsymbol{M}^{\text{building}} = 0.5 \cdot \mathbf{1}_{n^{\text{x,search}} \times n^{\text{y,search}}}$.
- Structure map, $\boldsymbol{M}^{\text{structure}} = \mathbf{1}_{n^{\text{x,search}} \times n^{\text{y,search}}}$.
- Wind velocity, $v^{\text{wind}} = 0\,\mathrm{m\,s^{-1}}$.
- Wind direction, $\theta^{\text{wind}} = -\frac{\pi}{4}$
- Fire spread constants, $c^{\text{fs1}} = 0.2$ and $c^{\text{fs2}} = 0.2$.
- Ignition time, $t^{\text{ignition}} = 120\,\mathrm{s}$.
- Burnout time, $t^{\text{burnout}} = 600\,\mathrm{s}$.
- Wind spread radius, $r^{\text{wind}} = 3$.
- Wind model constants, $c^{\text{wm1}} = 0.1$, $c^{\text{wm2}} = 0.1$, and $c^{\text{wmd}} = 0.4$.
- Max number of victims per search map cell, $n^{\text{victim}} = 5$.

Given the environment model dimensions ($n^{\text{x,env}}, n^{\text{y,env}}$), a standard initialisation for environment map parameters is also defined. Here, we illustrate a small disaster environment where $n^{\text{x,env}} = n^{\text{y,env}} = 40$. This initialisation is scaled appropriately for larger disaster environments unless otherwise noted.

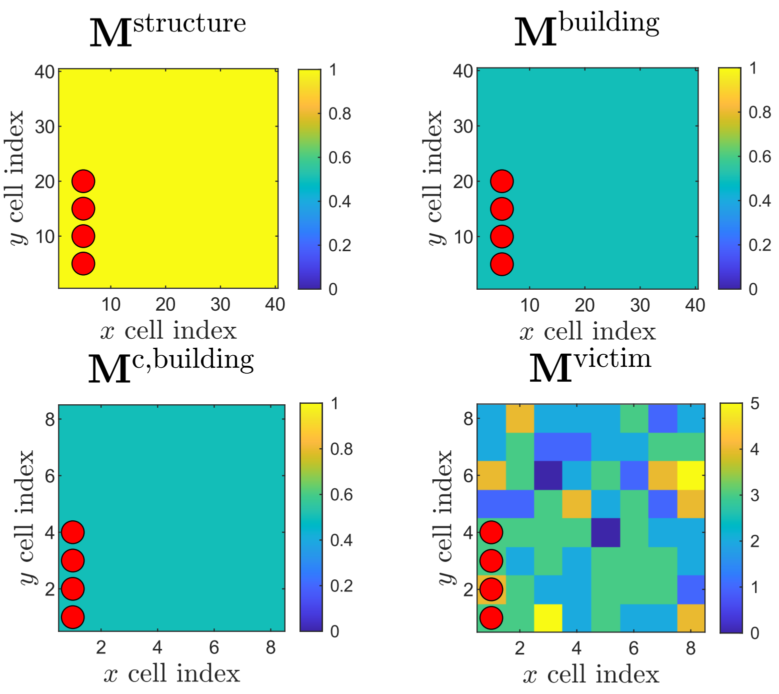


Figure 4.1: Single instance of environment setup for basic small disaster environment, red dots indicate agent positions. Each subplot represents the value of the labelled environmental state variable for the corresponding cell, from the lower limit (dark blue) to upper limit (yellow) of the parameter range.

Figure 4.1 shows the initialisation of the environment map parameters for the basic small environment with two agents. As shown, uniform distributions for $M^{\text{structure}}$ and M^{building} are defined. The figure shows one instance of the coarsened victim map, M^{victim}, where it can be seen that the distribution of victims is random over the disaster environment due to the uniform building map, however, this parameter varies with each simulation seed. Agents are initialised in a row of adjacent search maps cells in the bottom left hand corner of the disaster environment.

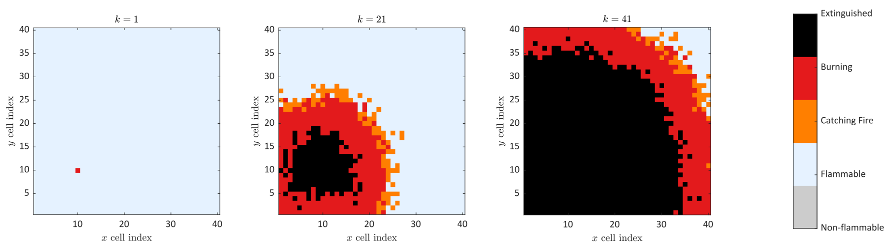


Figure 4.2: Fire spread progression for basic small dynamic environment, example simulation

Static environments are initialised with the fire map as a matrix of ones $M^{\text{fire}} = \mathbf{1}_{n^{\text{x,search}} \times n^{\text{y,search}}}$ so that there are no active fires during the simulation and there is no fire spread.

Likewise, Figure 4.2 shows the propagation of the fire model for one simulation. The fire map is initialised with a uniform distribution of flammable cells, and an active fire is initialised for four cells, where $M^{\text{fire}}[20:21, 20:21] = \mathbf{3}$.

This setup ensures the fire remains active in each simulation, rather than extinguishing prematurely.

4.1.3. Agent Model Configuration

We define the following standard agent model configuration:

- Agent airspeed, $v^{\text{airspeed}} = 5\,\text{m}\,\text{s}^{-1}$.

- Agent scan time per square metre, $t^{\text{scan}} = 0.01\,\mathrm{s\,m^{-2}}$.
- Agent sensor accuracy, $\eta = 0.9$.
- Scan certainty loss per time step, $\sigma = 0.01$.
- Agent tasks, $\tau^{\text{task}} = \mathbf{1}_{n^{\text{x,search}} \times n^{\text{y,search}}}$.

With MPFC, the agent target cells are initialised as the current agent locations, $\boldsymbol{s}_a^{\text{target}} = [\boldsymbol{s}_a^{\text{location}}]$. In contrast, for the MPC controller, the first set of target cells are initialised as the current agent locations and the remaining cells are set as ones, $\boldsymbol{s}_a^{\text{target}} = [\boldsymbol{s}_a^{\text{location}}, \quad \mathbf{1}_{n^{\text{x,search}} \times n^{\text{y,search}}}]$.

4.1.4. Controller Configuration

Three separate controller configurations are selected for the simulations, including an MPFC, MPC, and Pre-tuned FLC.

The various equations defining the optimisation step of the predictive controllers are set in section 3.4.3. To implement these optimisations in the simulation, we must select an appropriate solver, implement the optimisation bounds, and define the optimisation constraints.

In real-time control problems, a key constraint is the optimisation computation time, which must be completed before determining the next set of control inputs. Therefore, a set of optimisation constraints are imposed to reflect this and limit the time spent for each optimisation.

MPFC Configuration

The MPFC controller optimisation is configured to use a classic *pattern search* algorithm, which is selected due to its suitability for handling non-smooth, non-linear, and discontinuous optimisation problems without requiring gradient information. This makes it particularly well-suited for the complex and probabilistic nature of the environment model used in our simulations.

The key constraints for the optimisation are defined by the maximum number of function evaluations, $n^{\text{max-func-eval}} = 100$ and the maximum number of iterations, $n^{\text{max-iterations}} = 100$. The optimisation bounds are set as $\boldsymbol{l}_b = -\mathbf{1}$ and $\boldsymbol{u}_b = \mathbf{1}$, where $\boldsymbol{l}_b$ is the set of lower bounds, $\boldsymbol{u}_b$ is the set of upper bounds, and $\mathbf{1}$ is a vector of ones.

MPC Configuration

The MPC controller optimisation is configured to use a *genetic algorithm*, which is selected over *pattern search* due to its ability to handle discrete optimisation problems effectively and is well-suited for optimisation problems where the search space is large, complex, and discontinuous.

The key constraints for the optimisation are defined by the maximum number of function evaluations, $n^{\text{max-generations}} = 100$ and the population size, $n^{\text{population}} = 100$. The optimisation constraints are set as $\boldsymbol{l}_b = [1, 1]$ and $\boldsymbol{u}_b = [n^{\text{x,search}}, n^{\text{y,search}}]$, where $\boldsymbol{l}_b$ is the set of lower bounds and $\boldsymbol{u}_b$ is the set of upper bounds.

Pre-tuned FLC Configuration

The Pre-tuned FLC is configured as defined in Section 3.3, with the Fuzzy Rule base as given in A. The Pre-tuned FLC parameters are fixed throughout the simulation and provide the baseline performance for a simple local agent controller.

4.1.5. Standard Results Format

The simulation parameters are displayed using line plots for each of the simulation cases. To establish consistency across the visualisation of simulation results, we define a standard results format for each controller type and controller architecture, as shown in Figure 4.3 for sections 4.2 and 4.3. A solid line is used for centralised controllers, and a dashed line is used for decentralised controllers. FLC results are coloured green, MPFC results are coloured orange, and MPC results are coloured purple.

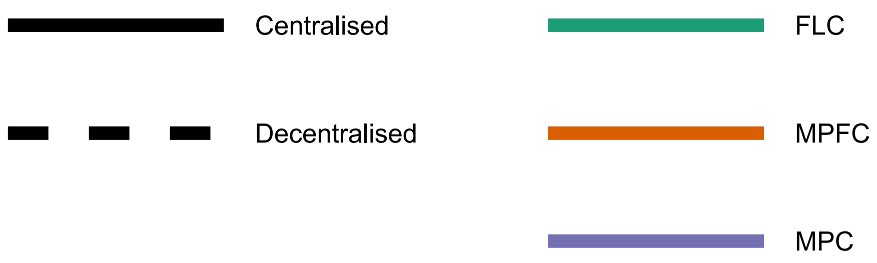


Figure 4.3: Line styles for simulation results

4.2. MPFC Performance Analysis

In the MPFC Performance Analysis, MPFC is simulated and assessed against the MPC and Pre-tuned FLC. This is done by beginning with simple simulation cases to isolate controller performance from the modelling complexities and gradually introducing more complex simulations.

4.2.1. Two-Agent System in Small Static Disaster Environment

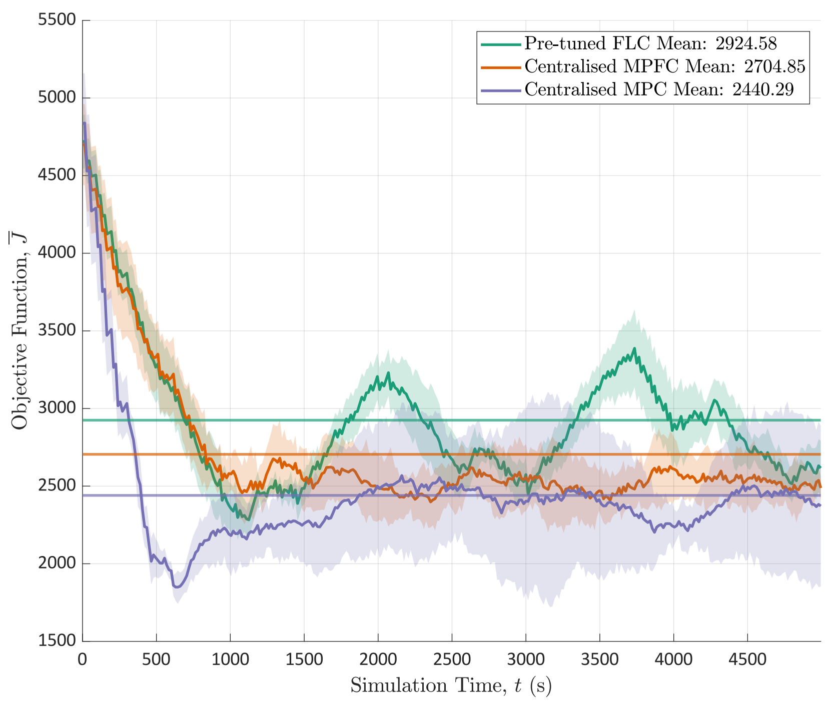


Figure 4.4: $\overline{J}(k)$ for a two-agent system in small static disaster environment, 5 simulations

Figure 4.4 presents the instantaneous objective function of the centralised MPFC, centralised MPC, and Pre-tuned FLC controllers for a two-agent system in a small static environment, simulated over

$T = 5000\,\mathrm{s}$. In the static scenario, the objective function starts around 5000, then rapidly decreases as agents scan cells and stabilises once most cells have been observed.

This illustrates the effect of the sensor accuracy component of M^{scan} (Equation 3.9) in the objective function (Equation 3.19), which has a high overall contribution to the objective function at the start of the simulation as all cells begin unscanned, and reaches a stable state when there is a balance between the rate that agents scan cells and the rate of degradation of M^{scan} due to the sensor accuracy component.

The Pre-tuned FLC demonstrates the poorest mean performance at 2925, followed by the centralised MPFC with 2705 $(-7.5\,\%)$ and the centralised MPC with 2440 $(-16.6\,\%)$.

Note that the initial performance of the Pre-tuned FLC is very close to the centralised MPFC, suggesting that the initial tuning of the FLC is well-chosen. The Pre-tuned FLC also does not manage to reach a stable objective function value, with two clear peaks, while both centralised MPFC and centralised MPC manage to reach a stable performance as they can adapt the performance of the local controllers.

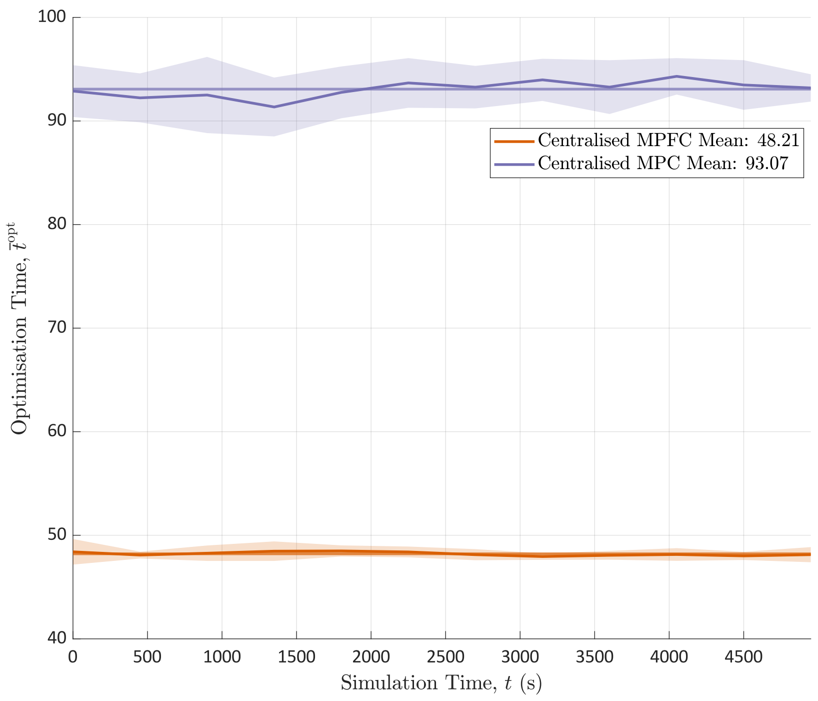


Figure 4.5: $\bar{t}^{\text{opt}}$ for a two-agent system in small static disaster environment, 5 simulations

Although centralised MPFC does not outperform centralised MPC, most of the difference in performance is during the first $1000\,\mathrm{s}$ of the simulation, where the centralised MPC is able to optimise the initial cells, while both stabilise around a similar objective function value over the remainder of the simulation. Additionally, the confidence intervals for centralised MPFC are much tighter. Figure 4.5 shows the computational optimisation time for each MPC step over the simulation. Centralised MPFC has a mean of $48\,\mathrm{s}$, almost half of the mean of $93\,\mathrm{s}$ of the centralised MPC. The optimisation time remains stable for both control methods over the simulation.

4.2.2. Two-Agent System in Small Dynamic Disaster Environment

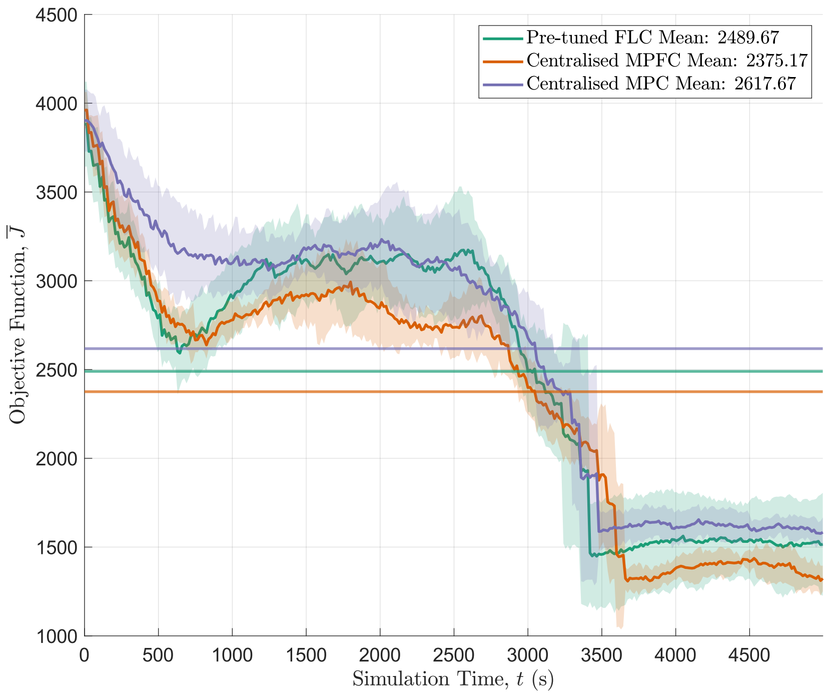


Figure 4.6: $\overline{J}(k)$ for two-agent system in small dynamic disaster environment, 5 simulations

Figure 4.6 presents the instantaneous objective function of the centralised MPFC, centralised MPC, and Pre-tuned FLC controllers for a two-agent system in a small dynamic environment, simulated over $5000\,\mathrm{s}$. The effect of the fire propagation can be seen in the objective function, with active fires reaching a peak at around $2000\,\mathrm{s}$ and most fires burning out at around $t = 3500\,\mathrm{s}$.

In this case, the centralised MPC has a mean objective function of 2618, followed by the Pre-tuned FLC with 2490 ($-4.9\,\%$) and the centralised MPFC with 2375 ($-9.3\,\%$). With the introduction of the uncertain fire spread component, the performance of the centralised MPC has degraded relative to the Pre-tuned FLC, while the centralised MPFC achieves the greatest performance.

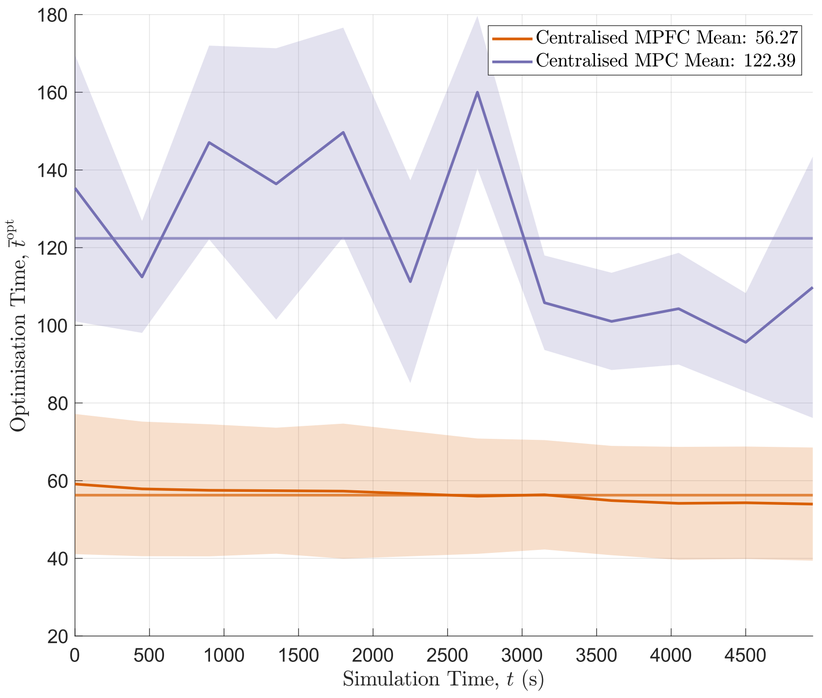


Figure 4.7: $\bar{t}^{\text{opt}}$ for a two-agent system in small dynamic disaster environment, 5 simulations

Figure 4.7 shows the computational optimisation time for each MPC step over the simulation. Centralised MPFC has a mean optimisation time of $56\,\text{s}$ while centralised MPC has as optimisation time of $122\,\text{s}$ (+$118\,\%$). The optimisation times for all control methods are higher due to the additional complexity of predicting fire spread. Unlike the static case, the optimisation time is highly variable for the centralised MPC, while the centralised MPFC maintains a stable optimisation time over the course of the simulation.

4.2.3. Four-Agent System in Small Dynamic Disaster Environment

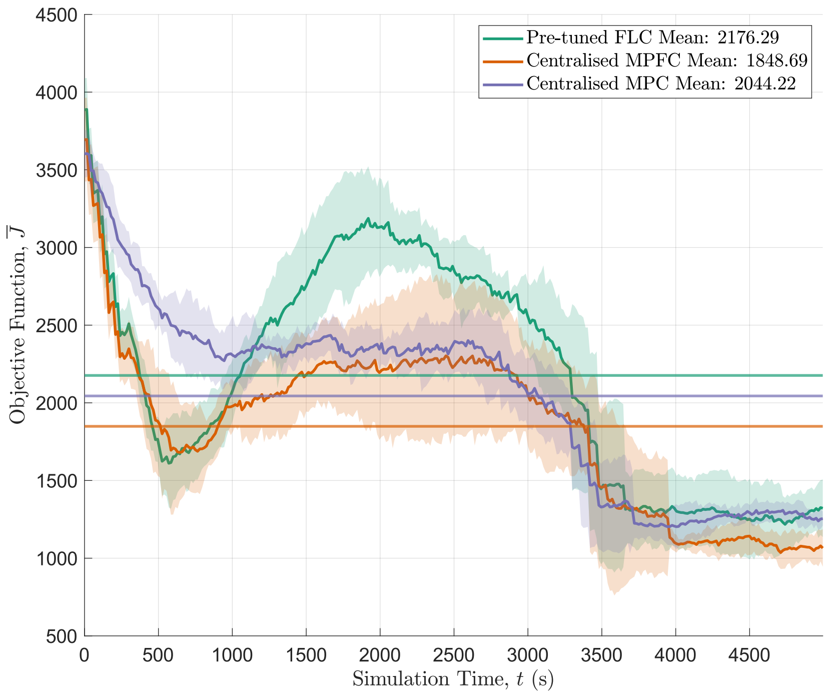


Figure 4.8: $\overline{J}(k)$ for four-agent system in small dynamic disaster environment, 5 simulations

In this simulation, the number of agents is increased to $n^{\mathsf{a}} = 4$ for the same configuration defined above in section 4.2.2. As shown in Figure 4.8, the Pre-tuned FLC demonstrates the poorest mean performance at 2176, followed by the centralised MPC with 2044 ($-6.1\,\%$) and the centralised MPFC with 1849 ($-15.0\,\%$). By introducing more agents in the same disaster environment, the objective function values are lower than in the two-agent case. Due to the introduction of more agents in the same size of environment, the agent actions become more closely coupled and the performance of the MPC and MPFC methods is improved relative to the Pre-tuned FLC, as they are able to optimise individual agent behaviours against the predicted future states.

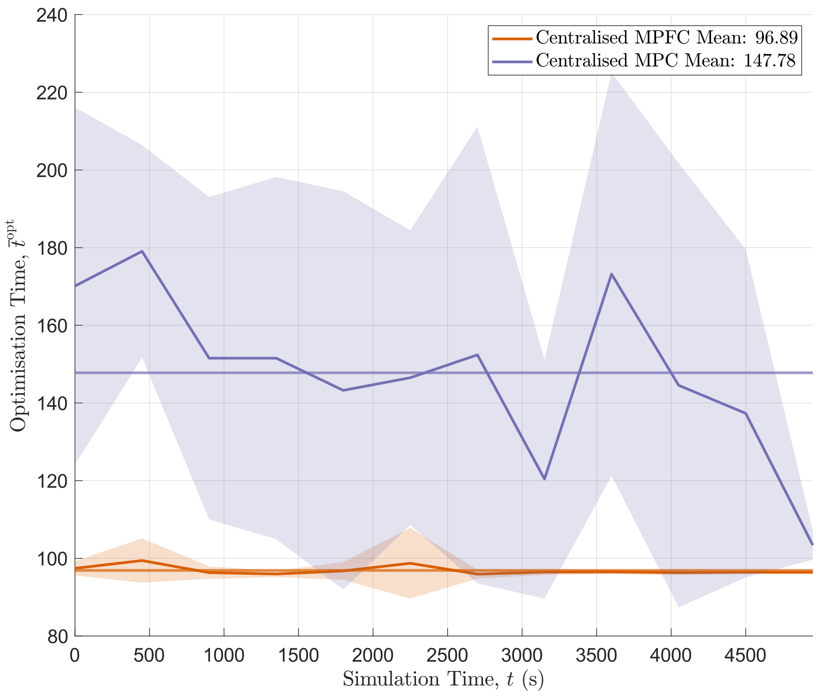


Figure 4.9: $\bar{t}^{\mathrm{opt}}$ for a four-agent system in small dynamic disaster environment, 5 simulations

Figure 4.9 shows that the centralised MPFC has a mean optimisation time of $97\,\mathrm{s}$ while centralised MPC has as optimisation time of $148\,\mathrm{s}$ (+$53\,\%$). Compared to the two-agent case, the centralised MPC optimisation time is closer to the centralised MPFC due to the greater increase in number of optimisation variables for the centralised MPFC versus the centralised MPC.

4.2.4. Decentralised vs centralised MPFC controller architectures

Maintaining the same basic small dynamic disaster environment model, the centralised MPFC and decentralised MPFC architectures are simulated to compare performance.

Centralised vs Decentralised MPFC for Two-Agent System

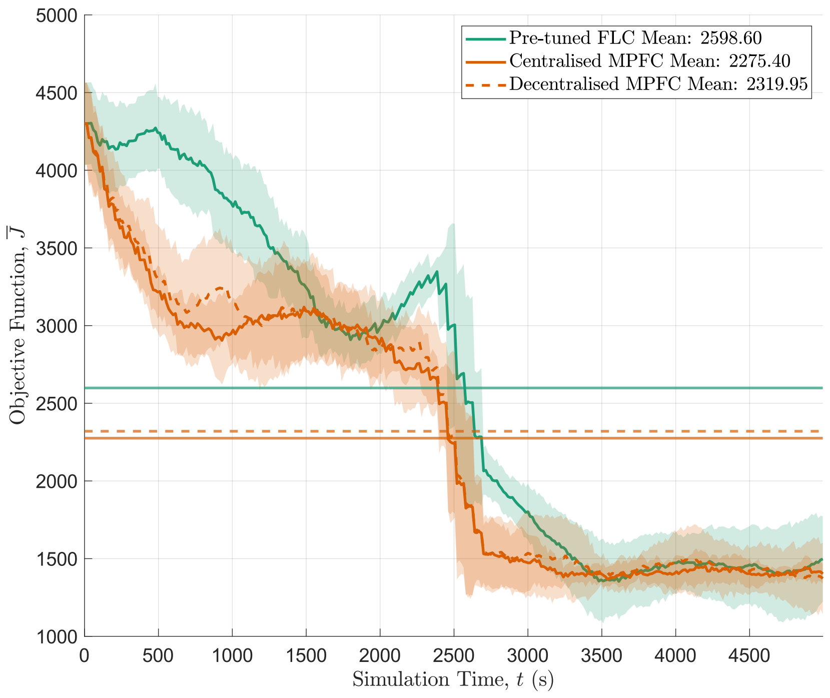


Figure 4.10: $\overline{J}(k)$ for centralised vs decentralised MPFC with $n^{\mathrm{a}} = 2$, 5 simulations

Figure 4.10 presents the instantaneous objective function of the centralised MPFC and decentralised MPFC controllers for a two-agent system in a small dynamic environment, simulated over $5000\,\mathrm{s}$. The Pre-tuned FLC demonstrates the poorest mean performance at 2599, followed by the decentralised MPFC with 2320 ($-10.7\,\%$), and the centralised MPFC with 2275 ($-12.5\,\%$). In this case both architectures achieve a similar performance, but there is a trade-off between the number of optimisation variables due to the number of agents in the system.

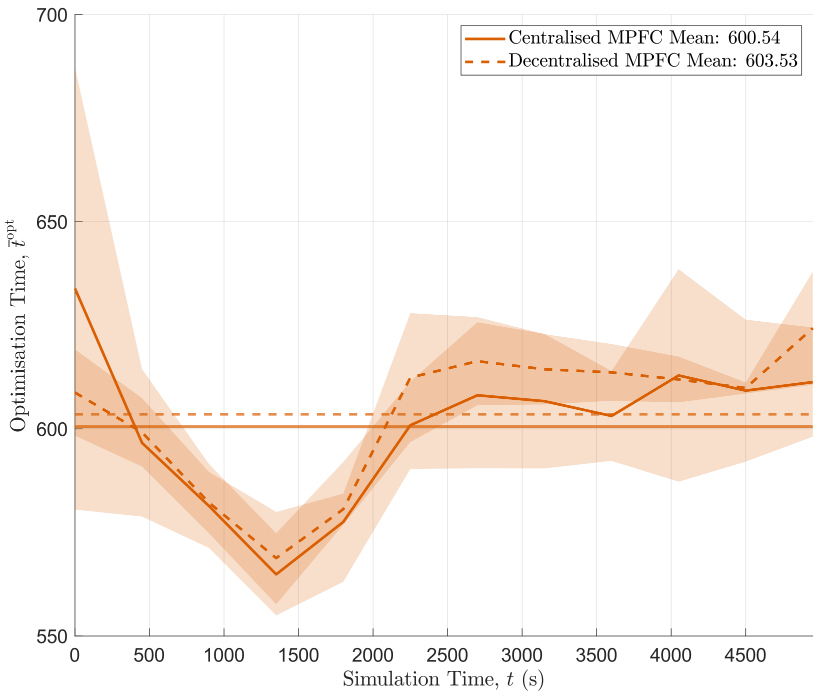


Figure 4.11: $\bar{t}^{\text{opt}}$ for centralised vs decentralised MPFC with $n^{\text{a}} = 2$, 5 simulations

Figure 4.11 shows that the centralised MPFC has a mean optimisation time of $601\,\text{s}$ while decentralised MPFC has as optimisation time of $604\,\text{s}$ (+$0.5\,\%$).

Centralised vs Decentralised MPFC for Four-Agent System

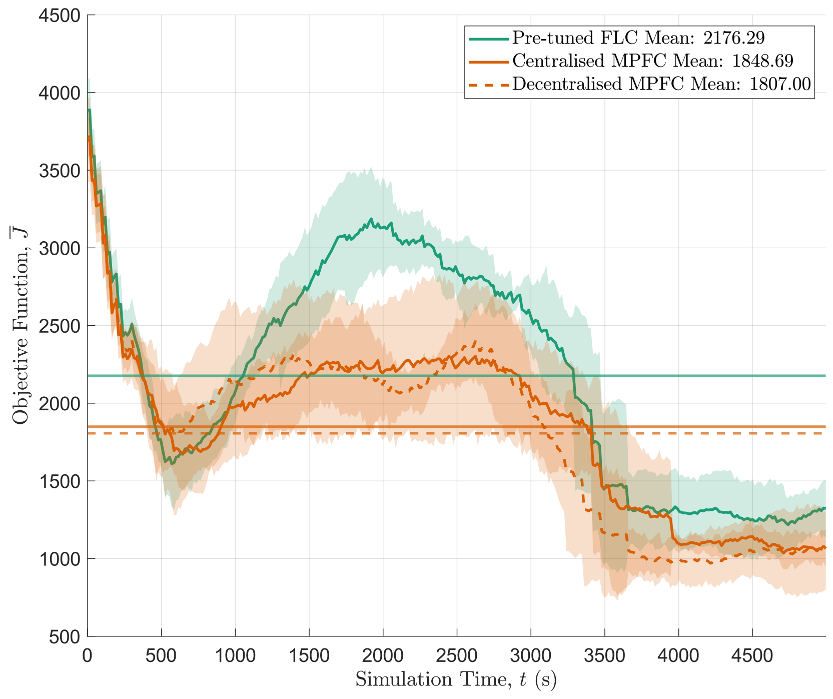


Figure 4.12: $\overline{J}(k)$ for centralised vs decentralised MPFC with $n^{\mathrm{a}} = 4$, 5 simulations

In this simulation, the number of agents is increased to $n^{\mathrm{a}} = 4$ for the same configuration defined above in section 4.2.4.

As shown in Figure 4.12, the Pre-tuned FLC demonstrates the poorest mean performance at 2176, followed by the centralised MPFC with 1849 ($-15.0\,\%$) and the decentralised MPFC with 1807 ($-17.0\,\%$). Likewise, Figure 4.13 shows that the centralised MPFC has a mean optimisation time of $97\,\mathrm{s}$ while decentralised MPFC has as optimisation time of $93\,\mathrm{s}$ ($-4\,\%$).

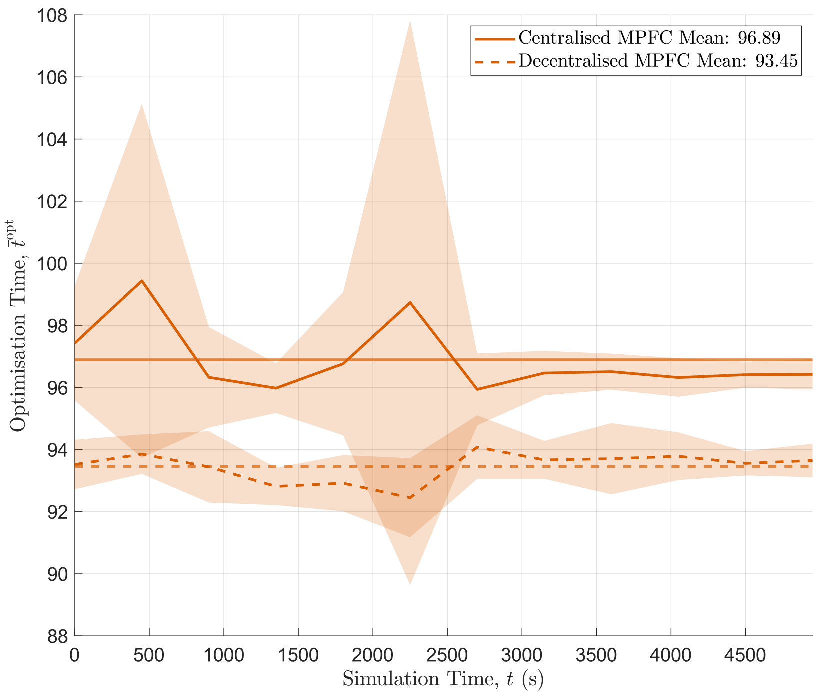


Figure 4.13: $\bar{t}^{\text{opt}}$ for centralised vs decentralised MPFC with $n^{\text{a}} = 4$, 5 simulations

In this case, the decentralised MPFC outperforms the centralised MPFC in both metrics despite the fact that agent behaviour is more coupled. This likely comes down to a trade-off between the degree of coupling between agent actions and the number of optimisation parameters which must be solved, which for centralised MPFC scales with the number of agents in the system.

4.2.5. Two-Agent System in Complex Dynamic Disaster Environment

In the previous performance analysis simulations, the environment variables were simplified as much as possible to isolate controller performance in a controlled environment. In this simulation, we analyse controller performance under more complex conditions by introducing more complex environment parameters.

Figure 4.14 shows the environment set up for a single simulation seed. We initialise $M^{\text{structure}}$ as a Perlin noise matrix to emulate clusters of buildings with different flammabilities, and M^{building} is defined by three probability density functions representing several population centres across the disaster environment.

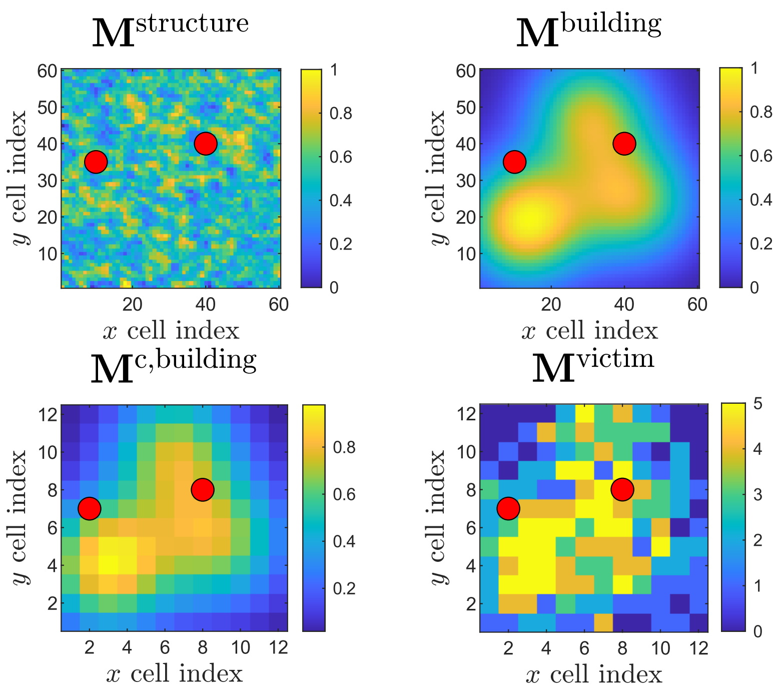


Figure 4.14: Single instance of environment setup for complex disaster environment, red dots indicate agent positions. Each subplot represents the value of the labelled environmental state variable for the corresponding cell, from the lower limit (dark blue) to upper limit (yellow) of the parameter range.

The wind velocity is set as $v^{\text{wind}} = 1\,\mathrm{m\,s^{-1}}$ and the wind direction is set as $\theta^{\text{wind}} = \frac{\pi}{4}\mathrm{rad}$. The fire model is configured to increase the influence of wind direction and velocity on fire spread and to slow down fire spread throughout the simulation. This is achieved by setting the constants of the fire and wind models as $c^{\text{fs1}} = 0.8$, $c^{\text{fs2}} = 2.5$, $c^{\text{wm1}} = 0.1$, $c^{\text{wm2}} = 1.5$, and $c^{\text{wmd}} = 0.9$. The fire map, M^{fire}, is initialised with two active fires in random cells in the disaster environment.

Figure 4.15 shows the fire spread over time for a single simulation seed. In this case, we can see that several fire fronts form and spread more strongly with the wind direction to the Northeast, while some cells do not catch fire due to their low flammability.

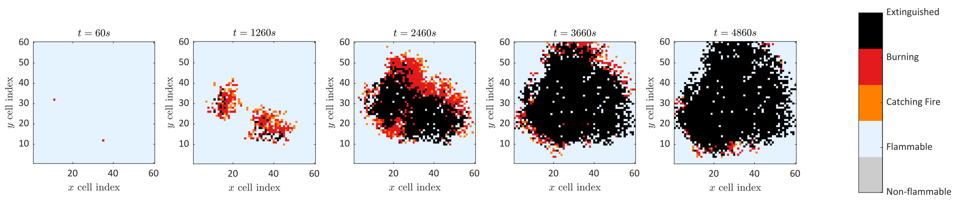


Figure 4.15: Single instance of fire spread for complex disaster environment

In this simulation case, fire spread occurs over the entire duration of the simulation, and does not have a clear peak early on as in previous simulations. As shown in Figure 4.16, the Pre-tuned FLC demonstrates the poorest mean performance at 7181, followed by the centralised MPFC with 6437 ($-10.4\,\%$) and the decentralised MPFC with 6283 ($-12.5\,\%$).

The centralised MPC achieves more stable performance over the entire simulation. This may be due to the MPC having direct control over agent target cells, which allows it to always respond to active fires regardless of the positions of the fires or agents, while MPFC can only control agent actions indirectly via the output parameters of the FLC. This highlights the importance of the design of the FLC. For instance, by choosing alternative input parameters, it may be possible for MPFC to tune agents to prioritise certain geographical areas of the disaster environment.

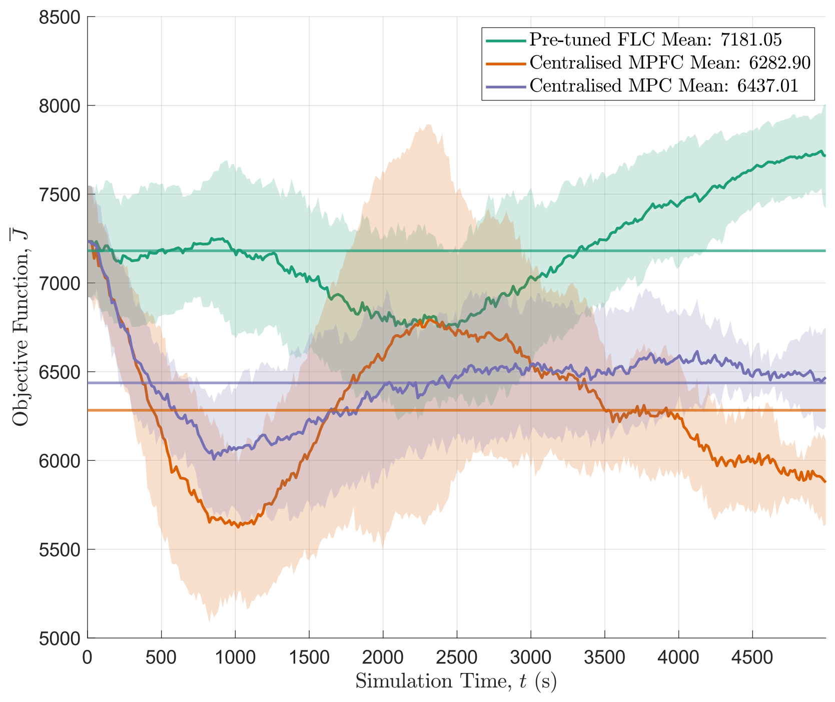


Figure 4.16: $\overline{J}(k)$ for a two-agent system in complex disaster environment, 5 simulations

4.3. MPFC Sensitivity Analysis

The objective of the sensitivity analysis is to investigate the performance of the supervisory controllers over a range of values for a given design parameter and identify the strengths and weaknesses of MPFC compared to the alternative controller architectures.

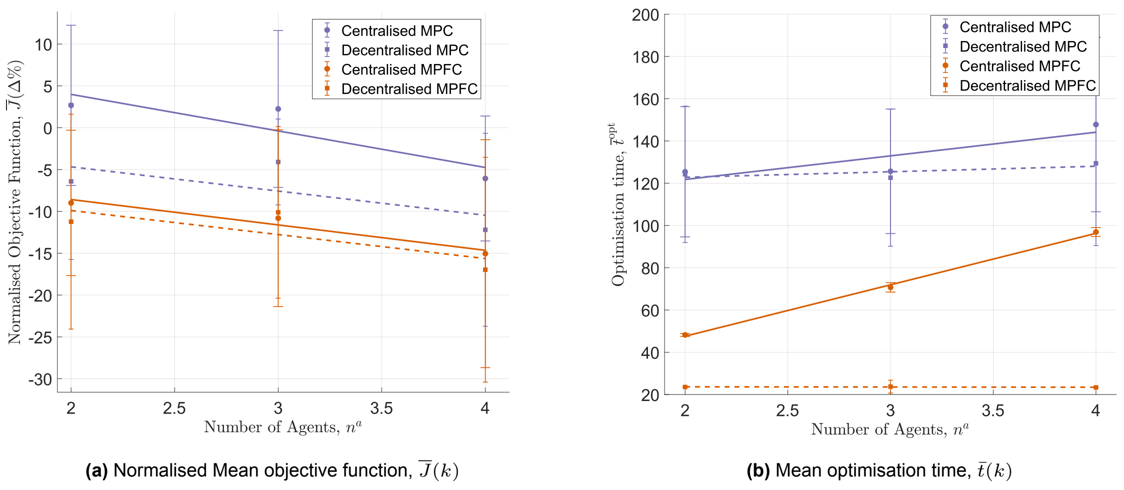
Four parameters are selected, each of which is a key design parameter in the sizing of the prediction step of the supervisory controllers, and an individual sensitivity analysis is performed for each parameter across a defined range of values.

It should be noted that many other parameters influence the overall controller performance that are outside the scope of this sensitivity analysis. For simplicity, we did not conduct a multi-parameter sensitivity analysis that explores multiple parameters simultaneously.

This method may provide additional insight into controller design if the design parameters are coupled. All sensitivity analyses are performed for the basic dynamic environment initialisation defined previously, with the design parameters adjusted according to Table 4.1.

Table 4.1: Sensitivity Analysis Parameters

Parameter	Range
Number of Agents (n^{a})	2, 3, 4
Disaster Environment Size ($n^{x^{\text{env}}} \cdot n^{y^{\text{env}}}$)	20, 40, 60
MPC Step Size ($\Delta t^{\text{MPC}}\text{s}$)	30, 75, 225, 450, 675, 900
Prediction Horizon ($\Delta k^{\text{pred}}\text{s}$)	450, 675, 900, 1125



(a) Normalised Mean objective function, $\overline{J}(k)$ **(b)** Mean optimisation time, $\bar{t}(k)$

Figure 4.17: Sensitivity with number of agents, n^{a}, 5 simulations each

4.3.1. Number of Agents

Figure 4.17a presents the normalised instantaneous objective function for decentralised and centralised MPC and MPFC architectures with the number of agents in the range $n^{\text{a}} = [2, 3, 4]$. The objective function values are normalised against the results for a Pre-tuned FLC controller and expressed as the percentage difference from the Pre-tuned FLC mean. The individual data points for each set of simulations are denoted by coloured dots, and the 95% confidence intervals are represented by error bars. To differentiate between controller architectures, centralised controller results are displayed using circular points and wider error bar tips, while decentralised controller results are displayed using square points and narrower error bar tips. Additionally, a first-order polynomial trend line is fitted to the results of each architecture to visualise the correlation with the number of agents. A solid line is used for centralised architectures and a dashed line is used for decentralised architectures.

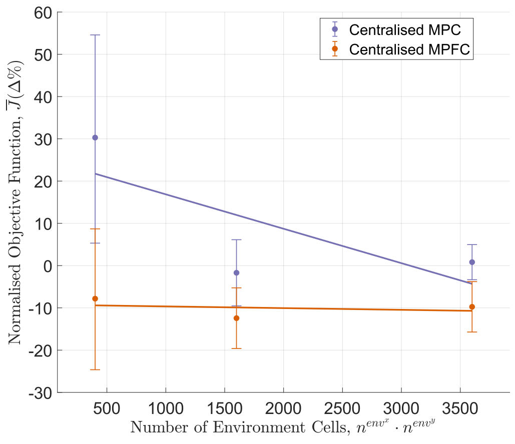
All predictive controller configurations exhibit improved objective function performance relative to the Pre-tuned FLC with n^{a}. As seen in the previous simulations, the decentralised MPFC architecture slightly outperforms the centralised MPFC architecture, and likewise for the decentralised and centralised MPC controller architectures. All controller architectures demonstrate relatively similar trends with the number of agents, although it is difficult to assert whether there is more or less correlation with the number of agents for a given controller architecture due to the range of the confidence intervals. For the full range of n^{a} tested, centralised MPFC consistently outperforms centralised MPC by around $10\,\%$ and decentralised MPFC consistently outperforms decentralised MPC by around $5\,\%$.

Figure 4.17b shows the mean optimisation times for the simulated architectures. The optimisation time for decentralised architectures remains stable with the number of agents, however decentralised architectures also require n^{a} separate optimisations to be performed instead of a single optimisation. Therefore, if these optimisations can be performed in parallel they may be more efficient than if they must be performed on the same processor. The overall optimisation time for the decentralised MPFC corresponds closely to the overall optimisation time for the centralised MPFC, while the optimisation time for the decentralised MPC per agent is close to the overall optimisation time for the centralised MPC, meaning that the overall optimisation time is in fact several times (equivalent to the number of agents) greater. The confidence intervals also clearly show that the variability in optimisation times is far lower for the MPFC architectures than for the MPC architectures.

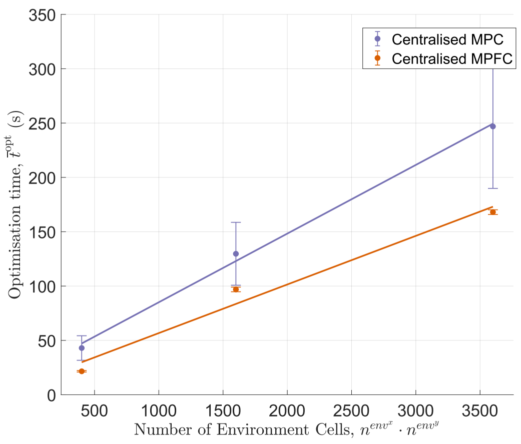
4.3.2. Disaster Environment Size

Figure 4.18a presents the normalised instantaneous objective function for the centralised MPC and MPFC controller architectures against the number of cells in the disaster environment. MPFC demonstrates around a fairly consistent $10\,\%$ improvement over the Pre-tuned FLC, while the MPC performs better in large disaster environments than smaller ones. The trend line for the MPC does not intersect the data point confidence intervals at 1600 cells, so it may not be a strictly linear relationship.

The mean optimisation times for this sensitivity analysis (Figure 4.18b) show that both have a linear correlation with the number of environment cells and do not demonstrate a clear difference in scalability between them.



(a) Normalised Mean objective function, $\overline{J}(k)$

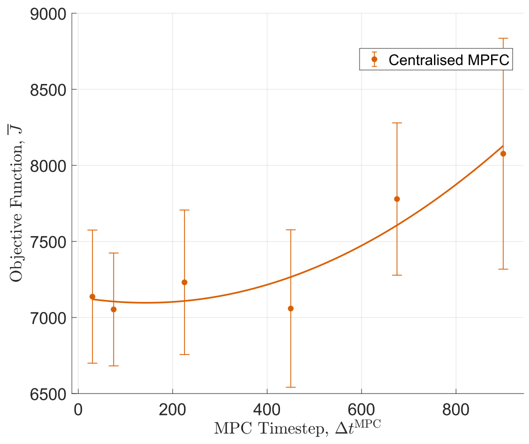


(b) Mean optimisation time, $\bar{t}(k)$

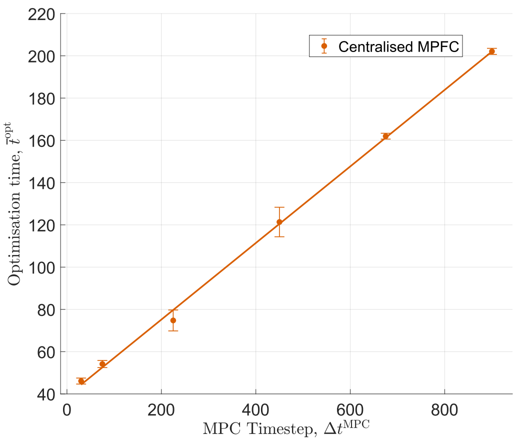
Figure 4.18: Sensitivity with environment size, $n^{\text{x,env}} \cdot n^{\text{y,env}}$, 5 simulations each

4.3.3. MPC Step Size

In our simulation, the MPC step size dictates the interval at which control parameters are updated, as well as the interval at which the MPC is called. In this sensitivity analysis, the MPC step size, k^{MPC}, is varied while the prediction horizon is maintained as $k^{\text{pred}} = k^{\text{MPC}} + 15$.



(a) Mean objective function, $\overline{J}(k)$



(b) Mean optimisation time, $\bar{t}(k)$

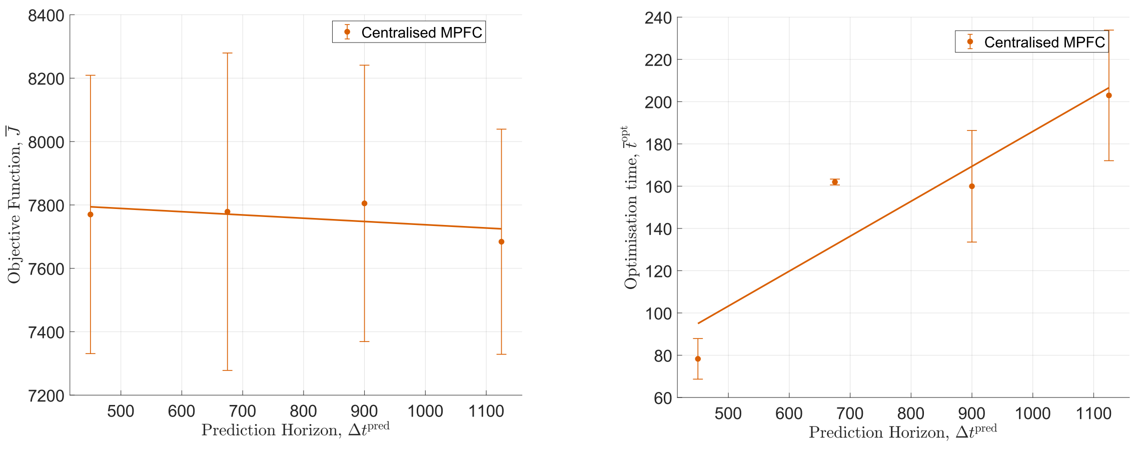
Figure 4.19: Sensitivity of MPFC with MPC step, Δt^{MPC}, 5 simulations each

The results appear to indicate roughly even performance in the range $100\,\mathrm{s}$ to $300\,\mathrm{s}$, with the MPFC performance degrading when the MPC step size is increased further.

Figure 4.18 demonstrates a strong linear correlation with optimisation time, although a decreased MPC time step will also require more frequent optimisations. An optimal selection of k^{MPC} may be to minimise it while selecting a value in the range that achieves the best overall objective function performance, possibly in the range $100\,\mathrm{s}$ to $300\,\mathrm{s}$ according to our results. The optimal range may have many influencing factors, including the response time of the agent, the environment dynamics, the accuracy of the prediction model, and many more.

4.3.4. Prediction Horizon

In this sensitivity analysis, the prediction horizon, k^{pred}, is varied while the MPC step size is constrained to $k^{\text{MPC}} = 30$. This results in the prediction horizon defining how far beyond the control horizon the MPC or MPFC controller performs its prediction.



(a) Mean objective function, $\overline{J}(k)$ **(b)** Mean optimisation time, $\overline{t}(k)$

Figure 4.20: Sensitivity of MPFC with prediction step, Δt^{pred}, 5 simulations each

Figure 4.20 shows no clear correlation between the global objective function with the prediction time step size for a strong linear correlation with the mean optimisation time (Figure 4.20). This suggests that extending the prediction horizon beyond the control horizon is of little value for the simulation implemented in this case study, but this correlation may be different in systems where agent actions have a stronger influence on the optimal future actions of the system.

4.4. MPFC Design Exploration

This section investigates potential design choices for MPFC, which may enhance its performance and computational efficiency or understand configuration options for the controller.

4.4.1. Prediction Modes

As mentioned in Section 3.4.3, two prediction modes are implemented. The *probability threshold* prediction mode is used for all simulations in the analysis as we assume the controller cannot predict future probabilistic states perfectly. In this simulation, we assess the performance of the controller with the *probability threshold* versus the *exact* prediction mode for a two-agent system in a small disaster environment. A large MPC step size is chosen to compare prediction modes as errors are more likely to accumulate over longer prediction horizons with the *probability threshold* prediction mode.

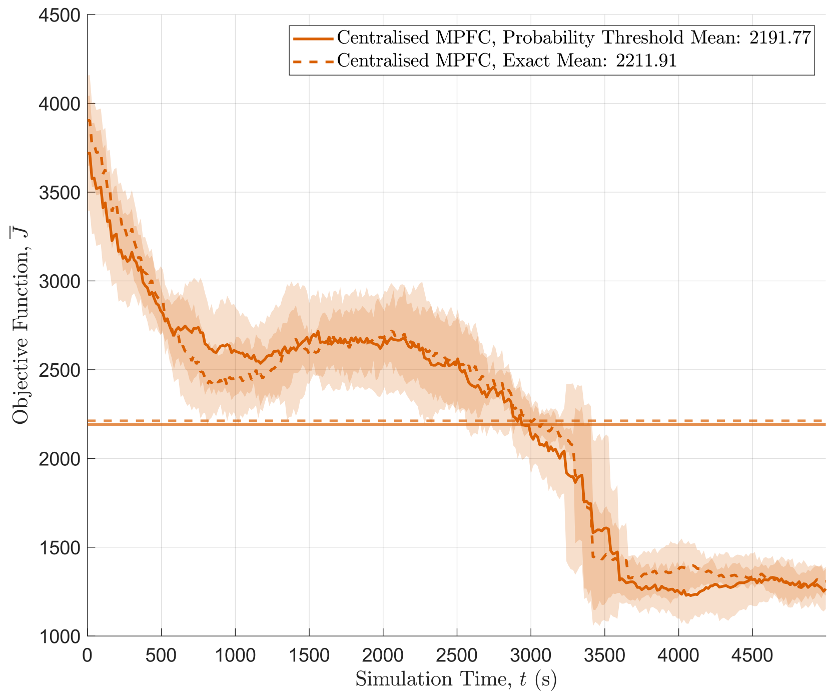


Figure 4.21: $\overline{J}(k)$ for prediction modes, $\Delta k^{\text{MPC}} = 30$, 5 simulations

The performance for two MPFC controllers using each prediction mode is shown in Figure 4.21. The predicted objective function remains consistent across both prediction modes, with both results within the confidence interval of the other.

The *probability threshold* prediction mode performs slightly better, with a mean objective function of 2191.77 against the mean objective function of 2211.91 for the *exact* prediction mode. The largest discrepancy between prediction modes is seen during the start of the simulation, when the most fire spread is occurring. The *exact* prediction mode achieves a better overall performance during the first $900\,\text{s}$ of the simulation, corresponding to the first two MPC steps, however this may have placed the agents in a less optimal situation for the remainder of the simulation.

To understand the relative difference in controller performance between these two prediction modes, a sensitivity analysis is run against t^{MPC} for centralised MPFC in the range $[450\,\text{s}, 900\,\text{s}]$. Figure 4.22 shows the objective function evaluation for each prediction mode, normalised against the performance of a Pre-tuned FLC.

Both achieve a relative decrease in objective function of around $15\,\%$ when $t^{\text{MPC}} = 450\,\text{s}$, degrading to around $14\,\%$ when $t^{\text{MPC}} = 900\,\text{s}$, however the confidence intervals remain wide.

As discussed previously, the *probability threshold* prediction mode outperforms the *exact* prediction mode in the simulations for $t^{\text{MPC}} = 450\,\text{s}$, however as expected the performance regrades relative to the *exact* prediction mode as the prediction horizon is increased due to the accumulation of prediction errors.

Overall, the performance of the *probability threshold* prediction mode correlates closely with the *exact* prediction mode, indicating it is a suitable estimation method for this simulation.

4.4.2. Type-1 vs Type-2 FLC

In this simulation, we implement MPFC in a dynamic environment with a Type-2 TSK FLC and compare it against our previous formulation using a Type-1 FLC. Type-1 FLCs use crisp MFs where each input is associated with a single degree of membership. This simplicity allows for efficient computations

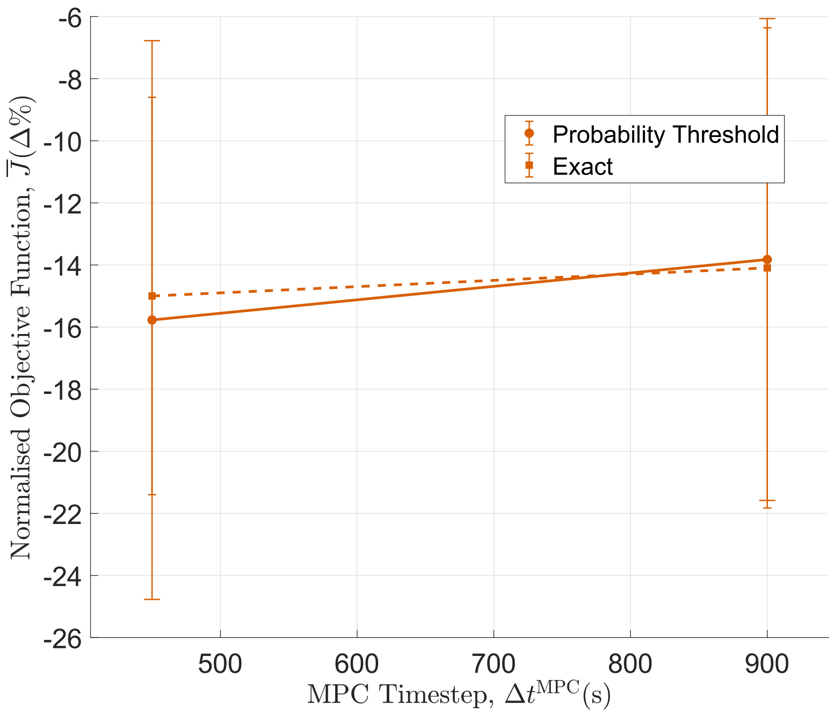


Figure 4.22: Comparison of prediction modes: sensitivity of centralised MPFC with t^{MPC}, 5 simulations each

and straightforward implementation. In contrast, Type-2 FLCs employ fuzzy MFs, which means that each input is associated with a range of degrees of membership, which can improve performance and robustness when faced with uncertain variables at the expense of more complex computations.

The results show that there is no clear difference in performance between the Type-1 FLC and the Type-2 FLC for the selected simulation scenario. A Type-2 FLC may be more suitable in scenarios where additional uncertain parameters are introduced or if the optimal system behaviour due to uncertain parameters were more complex.

Notably, the Type-1 configuration has a mean optimisation time of $700\,\mathrm{s}$ versus $654\,\mathrm{s}$ for the Type-2 configuration. This decrease in optimisation times with the Type-2 FLC may be due to faster convergence during optimisation due to the range of degrees of membership associated with each input.

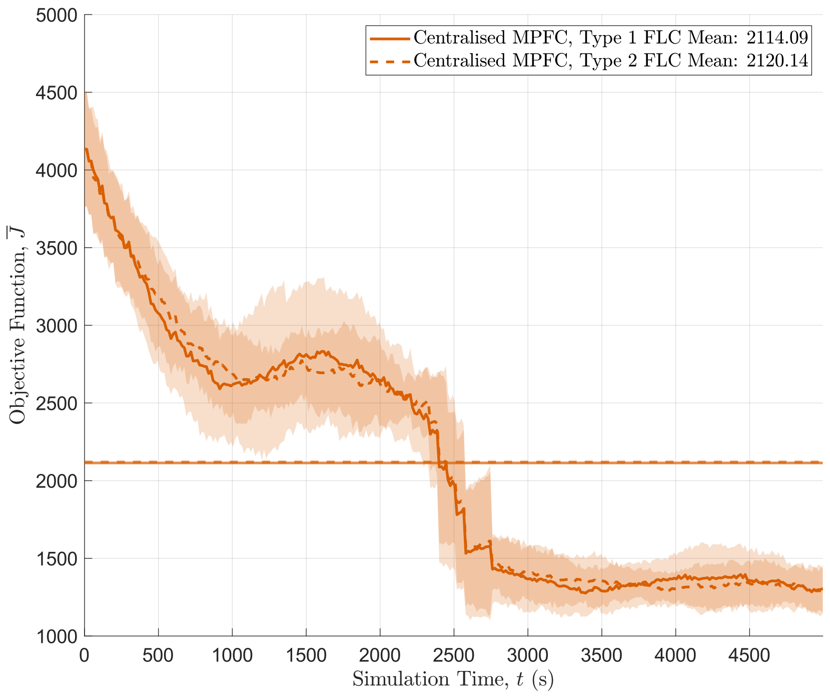


Figure 4.23: $\overline{J}(k)$ for Type-1 vs Type-2 FLC in MPFC architecture, 5 simulations

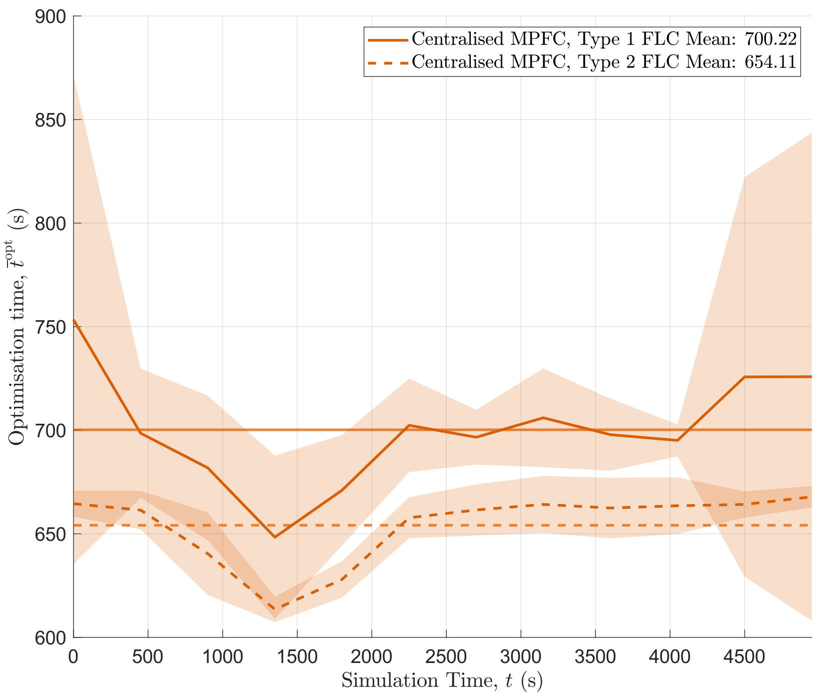


Figure 4.24: Mean optimisation time for Type-1 vs Type-2 FLC in MPFC architecture, 5 simulations

4.4.3. Local Prediction Maps

As discussed in section 4.3.2, a significant weakness of MPFC is that the optimisation time is highly dependent on the environment and search map dimensions. Due to the configuration of the control problem introduced in this case study, an agent will prioritise nearby cells over distant cells due to the lower response time to reach them. Therefore, we can assume that the FLC does not need to consider cells outside a given radius, r^{local}, in the x and y axis around the agent.

To address the issue of long optimisation times for large environments, a *local map* model is proposed to improve controller performance by limiting predictions to the local cells within a given radius of each agent, thereby reducing the computational complexity of the prediction step.

In this method, during the prediction step, a slice with dimensions $[2 \cdot r^{\text{local}} + 1, 2 \cdot r^{\text{local}} + 1]$ centred on the agent is taken for each search map state, $\phi^{\text{search}}(s_m, k)$. A local attraction map is then calculated and restored to the global map before the agent target cell assignment. Figure 4.25 illustrates the local map extraction. With this method, the computational complexity of the MPFC prediction is effectively decoupled from the search map size.

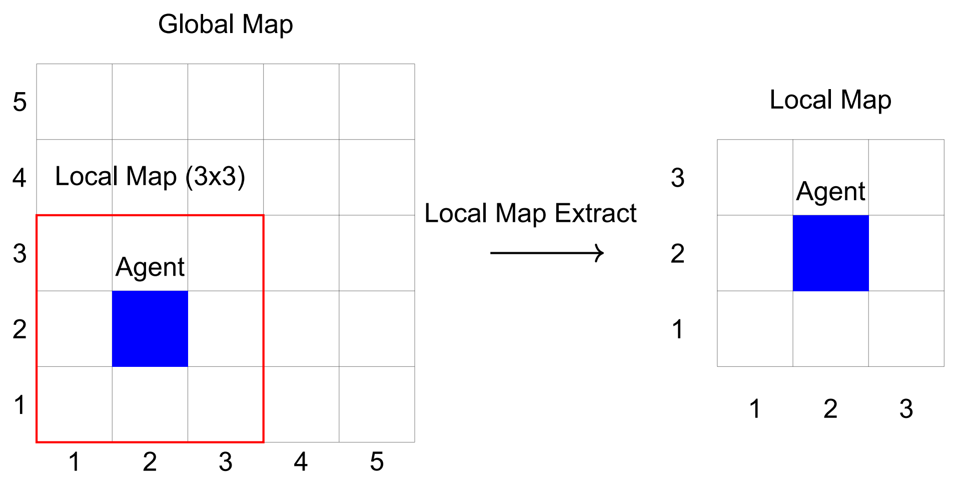


Figure 4.25: Extraction of local map from global map, where $r^{\text{local}} = 1$

Local Prediction Maps with Small Static Disaster Environment

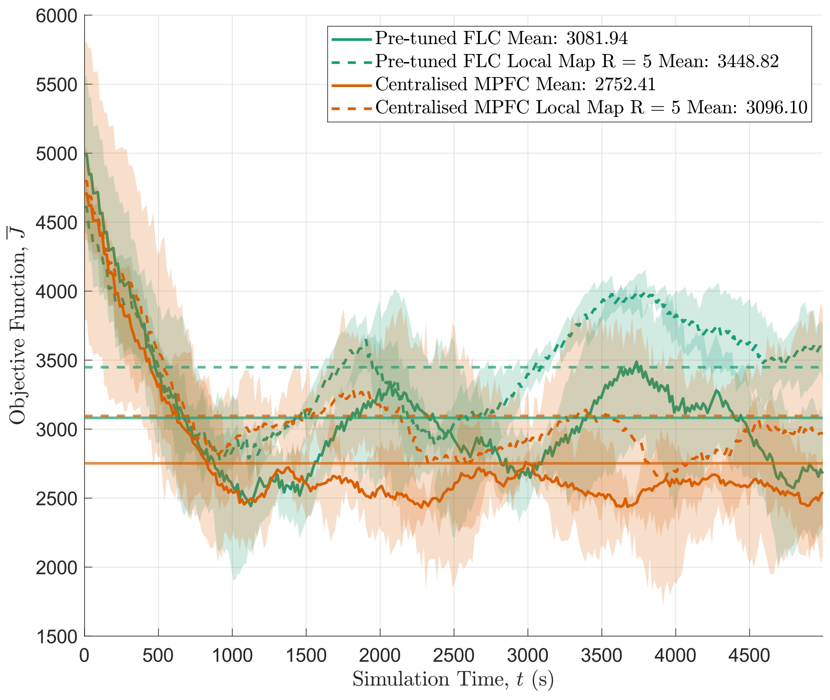


Figure 4.26: $\overline{J}(k)$ with r^{local}, 3 simulations

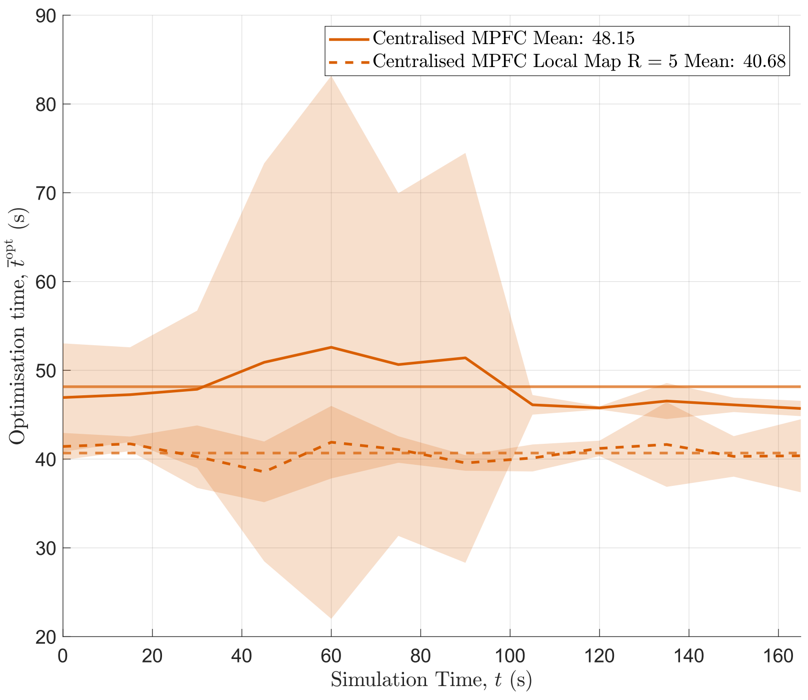


Figure 4.27: Mean optimisation time with r^{local}

First, the performance of the local map model is analysed for a two-agent system in a small static disaster environment of dimensions $n^{\text{x,env}} = n^{\text{y,env}} = 20$. We simulate MPFC and Pre-tuned FLC using both the global and local maps models with $r^{\text{local}} = 5$. The results in Figure 4.26 show that MPFC can maintain a similar performance using local maps to the Pre-tuned FLC using global maps, while the mean optimisation time is reduced from $48.2\,\text{s}$ to $40.7\,\text{s}$. The optimisation times are also far more consistent for the local maps model. The results in this simulation validate the potential for local maps to reduce optimisation times without a significant decrease in the objective function; therefore, the following simulation focuses on exploring the performance using this method for a range of values of r^{local} in a larger complex dynamic disaster environment.

Local Prediction Maps with Large Dynamic Disaster Environment

In this simulation, the local map model implementation is assessed for a two-agent system in a large simulated disaster environment of dimensions $n^{\text{x,env}} = n^{\text{y,env}} = 200$. MPFC is simulated using the local maps model for the range $r^{\text{local}} = [3, 5, 7]$.

The initialisation of the environment and agent states is shown in Figure 4.28, where red circles show the initial agent locations. The building map, M^{building}, is initialised with two population centres with Gaussian distributions; the structure map, $M^{\text{structure}}$ is fixed to ones; and the victim map M^{victim} is initialised according to the building distribution.

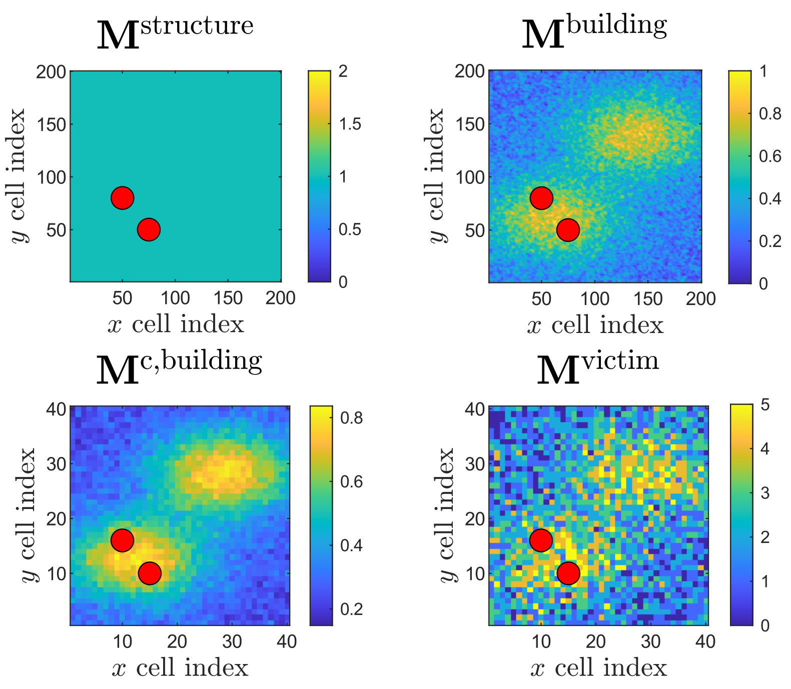


Figure 4.28: Disaster environment setup for local map simulation, red dots indicate agent positions

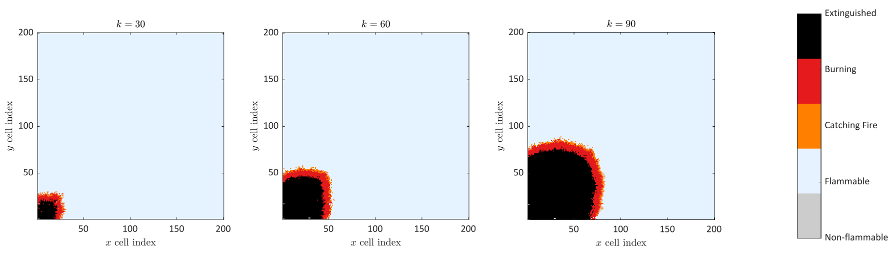


Figure 4.29: Fire spread progression for local map simulation, example simulation

The fire map is initialised with active fires in the bottom left corner. The progression of the fire spread over one of the simulations is shown in Figure 4.29. Due to the size of the disaster environment and the building map, the fire spread rapidly propagates out from the initial active fire point with a wide radial active fire *front*.

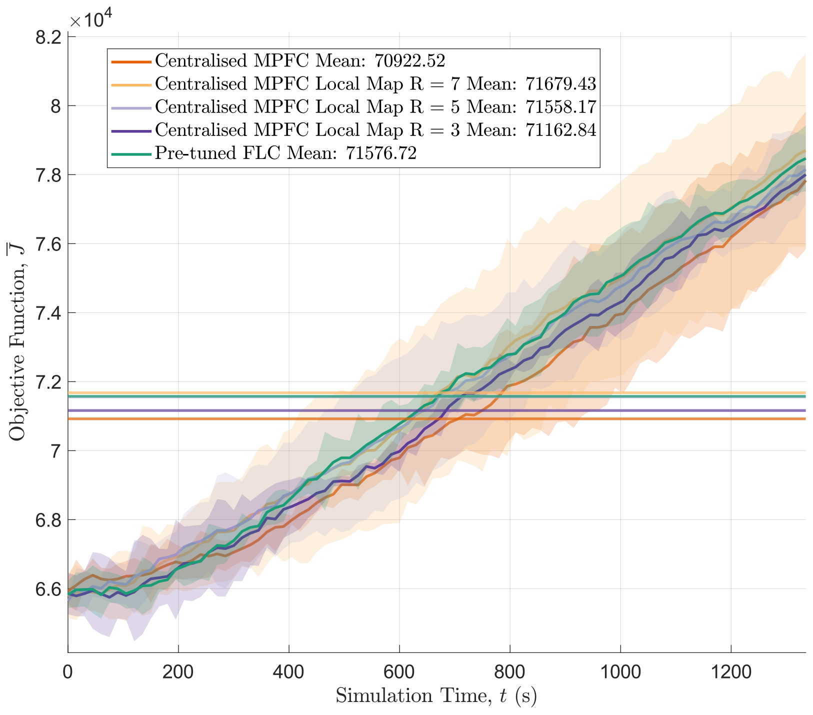


Figure 4.30: $\overline{J}(k)$ with r^{local}, 3 simulations

The results shown in Figure 4.30 show the objective function evaluated over the first $1400\,\mathrm{s}$ of the simulation, which was restricted due to the computational load of running global map simulations with this size of disaster environment. The objective function appears to increase continuously due to the size of the disaster environment compared to the number of agents and the propagation of the fire at the start of the simulation, resulting in an increasing objective function evaluated at each time step.

As expected, MPFC using global maps demonstrated the best mean performance of 70923. The remaining local map MPFC controllers achieve a performance of 71163 for $r^{\text{local}} = 3$ $(-0.3\,\%)$, and the other local map MPFC controllers performing worse than the Pre-tuned FLC. These results, combined with the confidence interval ranges, mean there is no clear trend we can extract between the value of r^{local} and mean performance.

This may be due to several contributing factors; including the limited snap-shot we are able to simulate for a disaster environment with this size, the fact that the beginning of the simulation is dominated by fire spread from a single source, meaning that the cells with the highest priority are likely neighbouring cells which have active fires at any one time, and the configuration of the weights in the objective function.

Despite the lack of correlation with r^{local}, all MPFCs using local maps demonstrated similar or improved performance than the Pre-tuned FLC with far lower optimisation times than MPFC using global maps, as shown in Figure 4.31. The second-best performing controller, MPFC with $r^{\text{local}} = 3$, has a mean optimisation time of $108\,\mathrm{s}$ compared to $794\,\mathrm{s}$ using global maps, a reduction of $86\,\%$.

These initial results indicate that this may be a feasible solution to implement MPFC which is scalable independent of the disaster environment size.

In further simulations, it may be possible to demonstrate an optimal value of r^{local}, which balances the prediction time against the radius of the local map. This may be achieved by investigating the mean performance for a given MPFC configuration over an entire SaR operation and initialising multiple

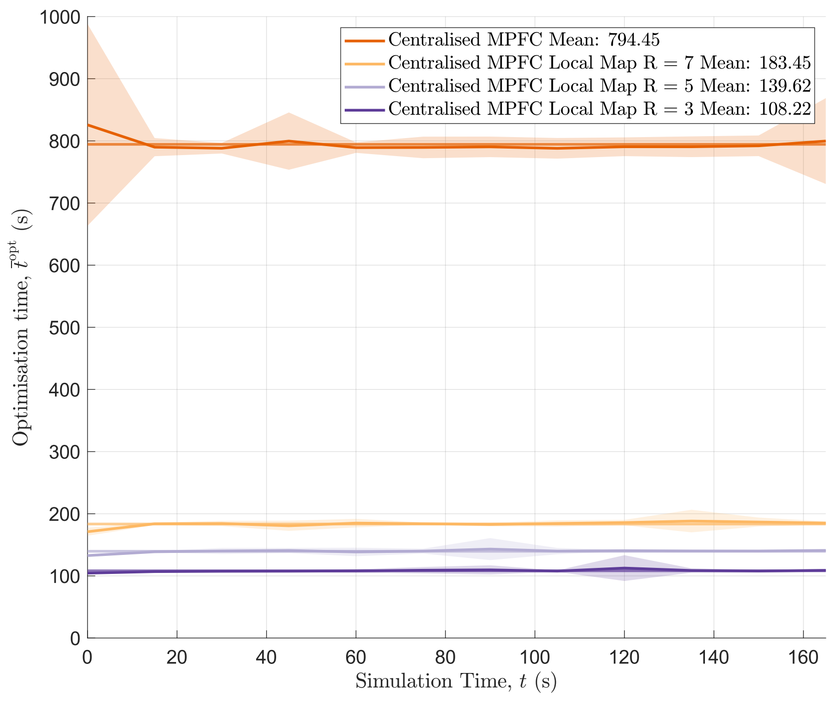


Figure 4.31: Mean optimisation time with r^{local}, 3 simulations

smaller fires across the disaster environment throughout the simulation. Further research may also explore other similar ideas in simplifying the computational intensity of the prediction.

5 Conclusions and Recommendations

Having discussed the performance, sensitivity, and design exploration of MPFC, this chapter consolidates our findings and outlines potential directions for future research. First, we summarise how each research objective was met and provide concise answers to the research questions posed in Section 1.4. We then consolidate the insights into MPFC gained from the simulation results and before presenting prioritised recommendations for future research on this topic.

5.1. Conclusion

In this research, MPFC is applied for the mission planning of a homogeneous multi-agent UAV system. We evaluated the controller's performance relative to alternative control methods, performed a sensitivity analysis of key design parameters, and explored various MPFC design configurations. To achieve this, assumptions and simplifications were made in order to limit the scope of the modelling and simulation work required to a feasible size.

The findings validate the feasibility and value of MPFC for certain applications against alternative control methods, providing clear insights into the design of an MPFC controller for multi-agent mission planning.

This project set out with clearly defined objectives and research questions aimed at advancing the understanding and application of MPFC, as outlined in Section 1.3. The following objectives were defined:

- OB1: *Specify a generic mathematical definition of the MPFC controller.*
- OB2: *Define and implement the MPFC controller model for a specific case study in simulation.*
- OB3: *Validate the performance of MPFC against alternative controller architectures via extensive computer-based simulations.*
- OB4: *Analyse the sensitivity of the MPFC controller to various design parameters.*
- OB5: *Explore design improvements to the basic MPFC controller implemented.*

To achieve these objectives, the project defined three research questions presented in Section 1.4:

- RQ1: *How does MPFC perform compared to traditional MPC and FLC controllers in a dynamic environment?*
- RQ2: *Which design parameters drive the performance of MPFC and how can they be optimised?*
- RQ3: *What design choices can be made then configuring an MPFC and in which cases should they be made?*

To address these questions, we defined and implemented MPFC for a mission planning case study, structuring the results into three sections:

- MPFC Performance Analysis (addressing RQ1): This study benchmarked the performance of MPFC against MPC and Pre-tuned FLC, providing insights into its advantages and potential limitations in dynamic settings.
- MPFC Sensitivity Analysis (addressing RQ2): This study conducted a sensitivity analysis to understand the influence of various design parameters on the performance of MPFC.
- MPFC Design Exploration (addressing RQ3): This study explored design considerations for implementing MPFC in multi-agent control scenarios, and understanding how to optimise MPFC configuration based on the needs of the use case.

Through this structured approach, the project both validates the capabilities of MPFC and provides an understanding of how its design and parameters can be fine-tuned to achieve superior performance in complex, dynamic environments.

5.1.1. MPFC Performance Analysis (Research question RQ1)

Simulations in Section 4.2 demonstrated that MPFC consistently achieved lower or comparable objective function values while maintaining tighter confidence intervals and reduced mean optimisation times. This was especially notable in environments where fire spread and other stochastic factors influenced agent behaviour.

When the number of agents was increased, MPC scaled more efficiently in both objective function performance and optimisation time, although MPFC continued outperform both methods. In comparing centralised and decentralised MPFC architectures, a clear trade-off emerged when increasing the number of agents. On one hand, centralised architectures can more effectively optimise behaviour when there is high inter-agent coupling, while on the other hand decentralised architectures limit the number of optimisation variables.

In highly dynamic and complex environments, the FLC design emerged as a potential bottleneck, since its output parameters constrain the extent to which agents can adapt their actions.

Note that an additional strength of MPFC not explored in this paper is that if communication between the supervisory controller and the agents is lost, MPFC will allow agents to continue their tasks while MPC would require agents to switch to an alternative control scheme.

Overall, these findings indicate the effectiveness of MPFC in the scenarios simulated in this case study, while demonstrating the necessity to carefully balance factors such as agent coupling, computational constraints, and FLC design.

5.1.2. MPFC Sensitivity Analysis (Research question RQ2)

Four sensitivity analyses were conducted to examine the effects of varying the number of agents, the number of environment cells, the MPC step size, and the prediction step size. In all tests, MPFC displayed more consistent optimisation times and achieved better performance than MPC within the simulated parameter ranges.

When increasing the number of agents, both MPC and MPFC improved their mean objective function relative to the Pre-tuned FLC, although centralised architectures exhibited high sensitivity in mean optimisation time, whereas decentralised approaches remained more stable. Larger environment sizes produced similar findings, except that centralised MPFC maintained a consistent objective function result relative to the Pre-tuned FLC and scaled less dramatically in optimisation time compared to centralised MPC.

The sensitivity analysis on MPC step size indicated that for the case study, MPFC demonstrated fairly consistent performance initially, before degrading in performance as the MPC step size was increased further. The results indicate that for a given case, there should be an optimal MPC step size, however more extensive simulations would be required to identify the optimal MPC step size for this case study. By contrast, the prediction step size showed no clear effect on the objective function in these simulations.

Note that in all cases, the scope of each sensitivity analysis was limited to several points with 5 simulations each due to the high computational load required to conduct each set of simulations, and more extensive sensitivity analyses may extract improved understanding of these relationships.

5.1.3. MPFC Design Exploration (Research question RQ3)

This study explored various design configurations of MPFC to understand if they enhance MPFC performance and scalability.

Two prediction modes—*probability threshold* and *exact*—were evaluated to investigate whether predicting the most likely outcome at each step provides acceptable performance of MPFC. Both modes yielded similar results, although the *probability threshold* mode exhibited some performance degradation for larger MPC steps, primarily due to error accumulation over extended prediction horizons.

MPFC was also implemented with a Type-2 TSK FIS, and comparison against the Type-1 TSK FIS configuration from earlier simulations.

The results showed a negligible difference in mean objective function between the two, although the Type-2 variant demonstrated a significant decrease in optimisation times.

Finally, the concept of *local prediction maps* was implemented and tested to improve system scalability in large environments. The results demonstrated large reductions (of up to $86\,\%$) in the optimisation time when using local prediction maps versus global maps with a small degradation in mean objective function ($-0.3\,\%$ in the best case scenario). However, results were mixed for the large dynamic simulation case and further research would be required to understand the optimal radius selection and mean performance in large simulation environments.

In summary, we showed that MPFC is a robust and computationally feasible alternative to standard MPC in multi-agent mission planning, particularly when carefully tuned for the problem size, expected environmental dynamics, and available computational resources.

5.2. Recommendations

It is the recommendation of this paper that any further research focuses on exploring the design and performance of MPFC in more detailed and complex control applications in order to further validate the feasibility and understand all of the design requirements before application with hardware in a physical lab setting or deployment in the field. In this section, several high priority research topics are identified.

5.2.1. Multi-Objective MPFC

In this case study, MPFC was applied to a single-objective optimisation problem, formulated as a mission planning for a SaR mission.

Further research with this simulation environment could explore the intricacies of MPFC design in relation to the interaction of the fire and wind models, the agent dynamics models, and more.

In reality, the system may be required to perform more complex operations. For example, in our selected case study, the multi-agent system may be required to deliver payloads to victims in the form of medical supplies and sustenance and deliver payloads to combat hazards such as active fires.

Agents may also need to consider more constraints, including monitoring their battery level and returning to recharge stations when necessary, hazards which could cause loss of the agent, and loss of signal while navigating the disaster environment.

Finally, the system may not operate in isolation within the SaR mission but also need to interface with other systems, such as directing the operations of ground teams, receiving data from external systems such as rescue helicopters or satellite imagery, and cooperating with other autonomous systems such as ground robots.

Various approaches may be taken in designing the MPFC controller to achieve this more complex system behaviour. One method could be to implement a multi-objective optimisation, in which a Pareto front of optimal solutions may be calculated against various objectives and a solution selected by the optimisation algorithm, which may the behaviour of each agent to be defined according to the predicted need. For instance, if the optimisation predicts a greater need for the delivery of medical supplies, several agents could be tuned for this behaviour, while fewer are tuned for lower-priority tasks. Likewise, to facilitate more complex behaviour, more complex FLCs could be defined with the required inputs and multiple output functions to determine the priority of each behaviour.

5.2.2. Distributed MPFC Architecture

In the MPFC model which was implemented, the simulations demonstrate that agent actions are more highly coupled when in closer proximity. They also show that the number of agents is a driving factor in the scalability of the system. Therefore, a distributed control architecture may be implemented for the predictive controller where agents within a certain radius of each other are clustered together, and each cluster is assigned a separate MPFC controller. This distributed control architecture may demonstrate an optimal balance between clustering radius and optimisation time that can outperform the decentralised and centralised architectures explored in this thesis.

5.2.3. Stochastic MPFC

In this thesis, a simple method was implemented for the prediction of future probabilistic environment states, which we termed **probability threshold** prediction mode (Equation 3.24).In this method, the prediction assumes the most likely outcome of any uncertain model at any time step. Instead, if stochastic MPC was implemented within MPFC, the controller may be more able to make optimal decisions for the agent behaviour depending on the range of possible outcomes of uncertain processes. This could also be baselined against a *random* prediction mode to use as a frame of reference for other prediction modes.

5.2.4. Imperfect Predictive Model

In assumption AS8, it is assumed that while MPFC can not exactly predict uncertain states, it does have a perfect predictive model of the environment and system.

In reality, the predictive model would be a non-perfect representation of the actual system and would accumulate prediction errors over the prediction horizon. A further study could formulate non-perfect mathematical models based on Ordinary Differential Equations, which model the actual system performance and validate the performance of MPFC with these predictive models. This would also allow the trade-off between high-fidelity and fast prediction time models to be assessed in the context of an MPFC controller.

5.2.5. Physics-Informed Neural Network MPFC

Physics-Informed Neural Networks (PINNs) are neural network architectures in which the neural network is guided by prior domain knowledge. This is implemented via imposing soft constraints based on mathematical models of a system, usually defined by Ordinary Differential Equations, during the training of the neural network. Compared to mathematical models, PINNs can be of higher fidelity with a lower prediction time while requiring far less training data than traditional neural networks.

As a predictive model within MPFC, PINNs could speed up model evaluations while maintaining high fidelity in, for instance, fire spread simulations. Future studies might benchmark PINN-based MPFC against purely mathematical or data-driven predictive models to measure trade-offs in computational cost and accuracy.

5.2.6. High-Fidelity Simulation

Finally, analysing MPFC using a higher-fidelity simulation would more closely approximate real operations. This could include 3D space models, continuous space models, detailed UAV agent dynamics and control models, UAV sensor modelling, data latency, communications models between agents and supervisory controllers, and many more. Demonstrating MPFC in such simulations is crucial to validate performance gains and ensure that assumptions scale to practical use cases, before eventually validating MPFC on physical hardware.

Closing Remarks

In conclusion, MPFC demonstrates substantial promise for the control of multi-agent systems in uncertain and dynamic environments, largely owing to the synergy between predictive control and fuzzy logic control. With further development and understanding of MPFC design considerations, alongside the recommendations identified in this section, MPFC has potential to be a viable, robust, and agile control method for multi-agent systems in complex real-world scenarios.

References

[1] Mirko Baglioni and Anahita Jamshidnejad. “A Novel MPC Formulation for Dynamic Target Tracking with Increased Area Coverage for Search-and-Rescue Robots”. In: *Journal of Intelligent & Robotic Systems* 110.4 (2024), p. 140.

[2] Xiao Cao et al. “HMA-SAR: Multi-Agent Search and Rescue for Unknown Located Dynamic Targets in Completely Unknown Environments”. In: *IEEE Robotics and Automation Letters* (2024).

[3] Bernardo Esteves Henriques, Mirko Baglioni, and Anahita Jamshidnejad. “Camera-based mapping in search-and-rescue via flying and ground robot teams”. In: *Machine Vision and Applications* 35.5 (2024), p. 117.

[4] Joana Gouveia Freire and Carlos Castro DaCamara. “Using cellular automata to simulate wildfire propagation and to assist in fire management”. In: *Natural hazards and earth system sciences* 19.1 (2019), pp. 169–179.

[5] Juliette Grosset et al. “Fuzzy Multi-Agent Simulation for Collective Energy Management of Autonomous Industrial Vehicle Fleets”. In: *Algorithms* 17.11 (2024), p. 484.

[6] Christopher de Koning and Anahita Jamshidnejad. “Hierarchical integration of model predictive and fuzzy logic control for combined coverage and target-oriented search-and-rescue via robots with imperfect sensors”. In: *Journal of Intelligent & Robotic Systems* 107.3 (2023), p. 40.

[7] Zehui Lu, Tianyu Zhou, and Shaoshuai Mou. “Drmamp: Distributed real-time multi-agent mission planning in cluttered environment”. In: *arXiv preprint arXiv:2302.14289* (2023).

[8] C. Maxwell. *Code for Model-Predictive Fuzzy Controller for Search-and-Rescue Path-planning of Multi-agent Systems*. `https://doi.org/10.4121/8050f9cb-d0b0-4149-bd24-02f13c2410db.v1`. 2024.

[9] C. Maxwell. *Integrated Model Predictive Fuzzy Control for Disaster Victim Detection Path Planning in MATLAB*. `https://github.com/craigmax-dev/Integrated-Model-Predictive-Fuzzy-Control-for-Disaster-Victim-Detection-Path-Planning-in-MATLAB`. 2024.

[10] Akira Ohgai, Yoshimizu Gohnai, and Kojiro Watanabe. “Cellular automata modeling of fire spread in built-up areas—A tool to aid community-based planning for disaster mitigation”. In: *Computers, environment and urban systems* 31.4 (2007), pp. 441–460.

[11] Savvas Papaioannou et al. “Distributed search planning in 3-d environments with a dynamically varying number of agents”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 53.7 (2023), pp. 4117–4130.

[12] Mohammad Sarbaz et al. “Hierarchical optimization-based model predictive control for a class of discrete fuzzy large-scale systems considering time-varying delays and disturbances”. In: *International Journal of Fuzzy Systems* 24.4 (2022), pp. 2107–2130.

[13] Filip Surma and Anahita Jamshidnejad. “State-dependent dynamic tube MPC: A novel tube MPC method with a fuzzy model of model of disturbances”. In: *International Journal of Robust and Nonlinear Control* (2024).

[14] Charbel Toumieh and Alain Lambert. “Decentralized multi-agent planning using model predictive control and time-aware safe corridors”. In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 11110–11117.

[15] Meng Zhou et al. “Multi-Robot Cooperative Target Search Based on Distributed Reinforcement Learning Method in 3D Dynamic Environments”. In: *Drones and Autonomous Vehicles* 1.4 (2024), p. 10012.

A

FLC Rule Base

This appendix contains the fuzzy rule base configuration used in the FLC.

$M^{\text{fire-risk}}$	M^{response}	M^{scan}	M^{priority}	M^{att}
low	low	low	low	high
low	low	low	medium	medium
low	low	low	high	medium
low	low	medium	low	medium
low	low	medium	medium	medium
low	low	medium	high	medium
low	low	high	low	medium
low	low	high	medium	medium
low	low	high	high	medium
low	medium	low	low	high
low	medium	low	medium	high
low	medium	low	high	medium
low	medium	medium	low	high
low	medium	medium	medium	medium
low	medium	medium	high	medium
low	medium	high	low	medium
low	medium	high	medium	medium
low	medium	high	high	medium
low	high	low	low	high
low	high	low	medium	high
low	high	low	high	high
low	high	medium	low	high
low	high	medium	medium	high
low	high	medium	high	medium
low	high	high	low	high
low	high	high	medium	medium
low	high	high	high	medium
medium	low	low	low	medium
medium	low	low	medium	medium
medium	low	low	high	medium
medium	low	medium	low	medium
medium	low	medium	medium	medium
medium	low	medium	high	medium
medium	low	high	low	medium
medium	low	high	medium	medium

$M^{\text{fire-risk}}$	M^{response}	M^{scan}	M^{priority}	M^{att}
medium	low	high	high	low
medium	medium	low	low	high
medium	medium	low	medium	medium
medium	medium	low	high	medium
medium	medium	medium	low	medium
medium	medium	medium	medium	medium
medium	medium	medium	high	medium
medium	medium	high	low	medium
medium	medium	high	medium	medium
medium	medium	high	high	medium
medium	high	low	low	high
medium	high	low	medium	high
medium	high	low	high	medium
medium	high	medium	low	high
medium	high	medium	medium	medium
medium	high	medium	high	medium
medium	high	high	low	medium
medium	high	high	medium	medium
medium	high	high	high	medium
high	low	low	low	medium
high	low	low	medium	medium
high	low	low	high	medium
high	low	medium	low	medium
high	low	medium	medium	medium
high	low	medium	high	low
high	low	high	low	medium
high	low	high	medium	low
high	low	high	high	low
high	medium	low	low	medium
high	medium	low	medium	medium
high	medium	low	high	medium
high	medium	medium	low	medium
high	medium	medium	medium	medium
high	medium	medium	high	medium
high	medium	high	low	medium
high	medium	high	medium	medium
high	medium	high	high	low
high	high	low	low	high
high	high	low	medium	medium
high	high	low	high	medium
high	high	medium	low	medium
high	high	medium	medium	medium
high	high	medium	high	medium
high	high	high	low	medium
high	high	high	medium	medium
high	high	high	high	medium

B
Simulation Configuration Scripts

This appendix provides a comprehensive list of the MATLAB configuration scripts used to initialise each simulation.

Table B.1: MATLAB Script Handles for Each Simulation[1]

Simulation	Main Script	Environment	Agent	FLC	Controller
4.2.1	h_s_victim_model_5000	h_env_static_40	h_a_repeat_2 h_a_repeat_2_mpc	h_init_fis_mirko_4	h_arch_fis h_arch_mpfc_output_prediction h_arch_mpc_prediction
4.2.2	h_s_victim_model_5000	h_env_dynamics_40	h_a_repeat_2 h_a_repeat_2_mpc	h_init_fis_mirko_4	h_arch_fis h_arch_mpfc_output_prediction h_arch_mpc_prediction
4.2.3	h_s_victim_model_5000	h_env_dynamics_40	h_a_repeat_4 h_a_repeat_2_mpc	h_init_fis_mirko_4	h_arch_fis h_arch_mpfc_output_prediction h_arch_mpc_prediction
4.2.4	h_s_victim_model_5000	h_env_dynamics_40	h_a_repeat_2 h_a_repeat_4 h_a_repeat_2_mpc h_a_repeat_4_mpc	h_init_fis_mirko_4	h_arch_fis h_arch_mpfc_output_prediction h_arch_mpfc_output_prediction_decentralised
4.2.5	h_s_victim_model_5000	h_env_dynamics_60_complex	h_a_repeat_2 h_a_repeat_2_mpc	h_init_fis_mirko_4	h_arch_fis h_arch_mpfc_output_prediction h_arch_mpc_prediction
4.3.1	h_s_victim_model_5000	h_env_dynamics_40	h_a_repeat_2 h_a_repeat_3 h_a_repeat_4 h_a_repeat_2_mpc h_a_repeat_3_mpc h_a_repeat_4_mpc	h_init_fis_mirko_4	h_arch_fis h_arch_mpfc_output_prediction h_arch_mpfc_output_prediction_decentralised h_arch_mpc_prediction h_arch_mpc_prediction_decentralised
4.3.2	h_s_victim_model_5000	h_env_dynamics_30 h_env_dynamics_40 h_env_dynamics_60	h_a_repeat_2 h_a_repeat_2_mpc	h_init_fis_mirko_4	h_arch_fis h_arch_mpfc_output_prediction h_arch_mpc_prediction
4.3.3	h_s_victim_model_mpc_2_pred_17 h_s_victim_model_mpc_5_pred_20 h_s_victim_model_mpc_15_pred_30 h_s_victim_model_mpc_5000 h_s_victim_model_mpc_45_pred_60 h_s_victim_model_mpc_60_pred_75	h_env_dynamics_60	h_a_repeat_2	h_init_fis_mirko_4	 h_arch_mpfc_output_prediction h_arch_mpfc_output_prediction h_arch_mpfc_output_prediction h_arch_mpfc_output_prediction h_arch_mpfc_output_prediction
4.3.4	h_s_victim_model_mpc_2_pred_17 h_s_victim_model_mpc_30_pred_30 h_s_victim_model_mpc_5000 h_s_victim_model_mpc_30_pred_60 h_s_victim_model_mpc_30_pred_75	h_env_dynamics_60	h_a_repeat_2	h_init_fis_mirko_4	h_arch_fis h_arch_mpfc_output_prediction h_arch_mpfc_output_prediction h_arch_mpfc_output_prediction h_arch_mpfc_output_prediction
4.4.1	h_s_victim_model_mpc_60_pred_75	h_env_dynamics_40	h_a_repeat_2	h_init_fis_mirko_4	h_arch_mpfc_output_exact h_arch_mpfc_output_prediction
4.4.2	h_s_victim_model_mpc_60_pred_75	h_env_dynamics_40	h_a_repeat_2	h_init_fis_mirko_4 h_init_fis_mirko_4_type2	h_arch_mpfc_output_prediction
4.4.3	h_s_victim_model_5000 h_s_victim_model_5000_local_map_r5	h_env_static_20	h_a_repeat_2	h_init_fis_mirko_4	h_arch_fis h_arch_mpfc_output_prediction
4.4.3	h_s_victim_model_5000 h_s_victim_model_5000_local_map_r3 h_s_victim_model_5000_local_map_r5 h_s_victim_model_5000_local_map_r7	h_env_dynamics_200_dualCentre	h_a_repeat_2	h_init_fis_mirko_4	h_arch_fis h_arch_mpfc_output_prediction

[1]The full set of initialisation files alongside the software code can be found in the 4TU Software Repository: `https://doi.org/10.4121/8050f9cb-d0b0-4149-bd24-02f13c2410db.v1` [8] or the GitHub repository [9].

C
Simulation Seeds

Table C.1: Simulation Seed List

Simulation	Parameter	Seeds	n^{sim}
4.2.1	-	6586, 9364, 1009, 3473, 9463	5
4.2.2	$n^a = 2$	265, 5052, 9173, 1171, 7530	5
4.2.3	$n^a = 4$	1755, 8611, 6476, 3092, 5726	5
4.2.4	$n^a = 2$	265, 5052, 9173, 1171, 7530	5
4.2.4	$n^a = 4$	1755, 8611, 6476, 3092, 5726	5
4.2.5	$n^a = 2$	803, 6063, 5333, 9967, 9982	5
4.3.1	$n^a = 2$	265, 5052, 9173, 1171, 7530	5
	$n^a = 3$	3866, 348, 8024, 8344, 1252	5
	$n^a = 4$	1755, 8611, 6476, 3092, 5726	5
4.3.2	$n^{x,\text{env}} \cdot n^{y,\text{env}} = 400$	4767, 2357, 6936, 3167, 6246	5
	$n^{x,\text{env}} \cdot n^{y,\text{env}} = 1600$	265, 5052, 9173, 1171, 7530	5
	$n^{x,\text{env}} \cdot n^{y,\text{env}} = 3600$	8721, 7857, 1151, 9093, 6561	5
4.3.3	$t^{\text{MPC}} = 30\text{s}$	5417, 2296, 8891, 3647, 6783	5
	$t^{\text{MPC}} = 75\text{s}$	9334, 9739, 826, 6031, 2898	5
	$t^{\text{MPC}} = 225\text{s}$	717, 624, 4521, 174, 5110	5
	$t^{\text{MPC}} = 450\text{s}$	9933, 4258, 9696, 6016, 7584	5
	$t^{\text{MPC}} = 675\text{s}$	9933, 4258, 9696, 6016, 7584	5
	$t^{\text{MPC}} = 900\text{s}$	9933, 4258, 9696, 6016, 7584	5
4.3.4	$t^{\text{pred}} = 450\text{s}$	2239, 7961, 6896, 8912, 833	5
	$t^{\text{pred}} = 675\text{s}$	2239, 7961, 6896, 8912, 833	5
	$t^{\text{pred}} = 900\text{s}$	2239, 7961, 6896, 8912, 833	5
	$t^{\text{pred}} = 1125\text{s}$	2239, 7961, 6896, 8912, 833	5
4.4.1	-	8904, 6149, 5712, 3194, 6791	5
4.4.2	-	9359, 4746, 839, 2634, 8288	5
4.4.3	Static Environment	3717, 2940, 4349	3
4.4.3	Dynamic Environment	8675, 2155, 278	3