COLLEGES SCIENCES
BRETAGNE POUR L'INGENIEUR
LOIRE ET LE NUMERIQUE

THÈSE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPÉRIEURE
DE TECHNIQUES AVANCÉES BRETAGNE

ÉCOLE DOCTORALE N° 648
Sciences pour l'Ingénieur et le Numérique
Spécialité : *Génie industriel, productique, automatique et robotique*

Par

Katell Lagattu

«Actuator fault recovery control strategy with Deep Reinforcement Learning - Application to Autonomous Underwater Vehicles»

Thèse présentée et soutenue à Brest, le 09/12/2025
Unité de recherche : Lab-STICC, CNRS UMR

Composition du Jury :

Rapporteurs :
Nicolas LANGLOIS — Professeur, Normandie Université, ESIGELEC-IRSEEM, Pôle Automatique & systèmes
Hélène PIET-LAHANIER — Directrice de recherche ONERA, DTIS, Palaiseau

Examinateurs :
Ignacio CARLUCHO — Assistant Professor, Heriot-Watt University, Edinburgh
Vincent CREUZE — Professeur, Université de Montpellier, LIRMM

Direction de thèse :
Benoit CLEMENT — Professeur, ENSTA Bretagne, Lab-STICC
Karl SAMMUT — Professor, Flinders University, College of Science and Engineering

Co-encadrement :
Gilles LE CHENADEC — Professeur, ENSTA Bretagne, Lab-STICC
Eva ARTUSI — Docteure, Ingénieure, Naval Group, Ollioules
Paulo E. SANTOS — Associate Professor, Flinders University / Priorianalytica

Remerciements

Je tiens tout d'abord à remercier l'ensemble des membres du jury d'avoir accepté de prendre part à l'évaluation de cette thèse. Leur expertise et la diversité de leurs horizons scientifiques ont permis d'apporter des réflexions et améliorations très constructives sur ce travail de recherche. En particulier, merci à Nicolas Langlois et Hélène Piet-Lahanier d'avoir été rapporteurs. Je remercie également Ignacio Carlucho et Vincent Creuze d'avoir été les examinateurs de cette thèse.

Je tiens ensuite à adresser mes profonds remerciements à mon équipe encadrante sans qui cette thèse n'aurait pas eu lieu. Vous m'avez apporté un soutien sans faille pendant ces trois années, en me guidant tout en me laissant prendre en maturité et en autonomie. Vous m'avez aidée à construire le sujet de thèse, former une démarche de recherche et accompagnée tout au long du développement de la thèse, toujours dans la bienveillance et la bonne humeur. Vous m'avez aidée à prendre confiance en moi et à construire des arguments solides. En particulier, merci à Benoit Clement de m'avoir toujours soutenue, encouragée et remonté le moral, tout en me donnant goût au travail de recherche. Merci pour tes nombreux retours constructifs sur mes travaux, et pour ta bonne humeur et ta patience devant mes nombreux doutes et questions. Et si tu as besoin d'une cat-sitter, tu sais qui appeler ! Je remercie également Karl Sammut pour son aide dans la construction de ma thèse et dans les essais expérimentaux, ainsi que pour la gentillesse avec laquelle il m'a accueillie à Flinders University, me permettant de passer une année et demie incroyable en Australie. Merci à Paulo E. Santos pour son aide très précieuse dans la relecture des articles, et pour nos discussions scientifiques constructives qui m'ont permis de débloquer de nombreux problèmes. Je remercie également Gilles Le Chenadec, qui m'a aidée à déterminer les failles dans mes travaux et à trouver des solutions, et m'a toujours soutenue dans la bonne humeur. Enfin, merci de tout mon cœur à Eva Artusi, qui m'a prise sous son aile dès mon arrivée à Naval Group, m'a aidée à construire ma réflexion tout au long de la thèse, et m'a poussée à me dépasser et à gagner en assertivité. Merci pour ton soutien infaillible et pour m'avoir aidée à tous les niveaux dans ma thèse, que ce soit au niveau scientifique en mettant les mains dans le code, au niveau de la rédaction des articles et du manuscrit en me faisant toujours une tonne de commentaires impitoyables mais justes, ou au niveau du moral où tu m'as toujours montré que tu croyais en moi. Tu as été une encadrante formidable qui m'a prouvée qu'on pouvait faire les choses sérieusement sans se prendre au sérieux, et tu es maintenant une amie très précieuse. C'est grâce à toi que je me suis aussi bien intégrée à Toulon, si bien que maintenant j'y emménage.

Je tiens à remercier G. Bartoli, S. Rousset, S. Deleest et A. Negre de m'avoir permis de réaliser cette thèse avec Naval Group. Mention spéciale à tous mes collègues de Naval Group qui m'ont accueillie avec beaucoup de gentillesse au début de ma thèse. Merci en particulier

à Sacha G., Margaux C., Tom L., Paul M., Jérémy P. et J-B V. pour tous les fous rires et les sorties du jeudi soir.

Merci à J-P. Diguet de m'avoir accueillie à l'IRL CROSSING et à B. Nicolas-Meunier et C. Buche de m'avoir accueillie dans les locaux de Naval Group à Adelaïde.

Je remercie également Antoine W., Quentin A., Raphaël L., Quyen P., Maëlic N., Océan N. et Marie D., mes collègues français en Australie, avec qui j'ai partagé des expériences inoubliables telles que du camping dans le bush où tout aurait pu très mal tourner, des road trips dans le désert sur des routes infinies au milieu des chevaux sauvages, et de la plongée au milieu des coraux et des tortues. En particulier, merci à Thomas C. pour ton aide précieuse pour les expérimentations et pour nos discussions dans la bonne humeur le midi. Enfin, merci profondément à Hélène L. et Antoine V. pour leurs soutiens sans faille pendant ma thèse en Australie et pour m'avoir fait sentir comme chez moi à Adelaïde. Mentions spéciales à Hélène pour notre épopée à vélo en Nouvelle-Zélande et pour nos rendez-vous au café pour « faire le point », et à Antoine pour nos discussions libératrices en voiture et nos weekends prévus au dernier moment avec pour seule idée de « partir loin ». Vous avez été comme une famille pour moi à l'autre bout de la planète.

Je remercie également mes collègues de l'ENSTA à Brest, Adrien G., Alexia T., Vincent T., Aymeric L.D, Laurent L., Raphaël V., Sylvain B., Titouan M. et Corentin C. qui m'ont permis de passer une super troisième année de thèse, remplie de pauses de midi génialissimes et de surf en eau froide.

En particulier merci à ma voisine de bureau à Brest Samira B. qui m'a soutenue dans les moments difficiles de ma troisième année, a été une formidable amie très prévenante et m'a permis de décompresser lorsque le stress se faisait ressentir.

Merci également à mes meilleur-es ami-es qui m'ont soutenue pendant ces trois années de thèse malgré la distance : Eva M. avec les innombrables vocaux qui mettent de bonne humeur, Agathe M.N. avec les gaufres belges et les fous rires aléatoires, Camille H. avec la chasse aux champignons, Julie C. avec les soirées réconfortantes à Brest, Emma M. avec sa passion pour la Bretagne, Marianne A. avec les sorties escalade et Clément B. avec les escapades improbables et le soutien intergénérationnel entre PhDs.

Enfin, je tiens à remercier du fond du cœur ma famille pour leur soutien indéfectible et en particulier ma mère, qui m'a toujours encouragée à faire mes propres choix tout en me soutenant quels qu'ils soient. Merci de m'avoir fait confiance, de m'avoir encouragée à continuer et merci d'être partie à l'autre bout du monde avec mon Papi âgé de 82 ans pour me rendre visite.

Pour finir, je veux adresser quelques mots à mon copain Antoine L. qui a souvent écouté mes répétitions, relu mes textes et m'a aidée à vivre sereinement la troisième année de thèse. Tu as redoublé de patience devant mon tempérament pas toujours facile, mon stress

et mes doutes, et tu as toujours été là pour me remonter le moral, comme par exemple en m'emmenant faire le tour du Var à moto sur un coup de tête. Je te remercie pour tout.

« Il faut se battre dans la vie. »
Fabienne Touchard

« Au pire, on ira élever des chèvres ensemble. »
Antoine Vivien

« Pas de panique ! »
Eva Artusi et Benoit Clement

Abstract

Autonomous dynamic systems, such as exploration drones, are at risk from actuator faults, which can compromise the operation or lead to a system loss. This is why it is necessary to establish a Fault-Tolerant Control (FTC) strategy to maintain system performance and stability under actuator faults. Traditional FTC methods commonly use control reallocation, redistributing forces among remaining actuators. Control reallocation requires explicit fault diagnosis, *i.e.* identifying the fault's origin and intensity in real time. Explicit fault diagnosis becomes impractical with multiple faults or limited computational capacity. These limitations have led to the exploration of Deep Reinforcement Learning (DRL) for system control. The idea of existing DRL-based FTC strategies in the literature is to leverage DRL to directly generate control inputs for the actuators, allowing the system to autonomously adapt to faults. DRL offers high adaptability for actuator fault recovery, enabling the control strategy to operate without explicit fault diagnosis. However, DRL-based control lacks transparency due to the unpredictability of DRL model outputs, which poses a challenge in guaranteeing stability of the system. This thesis addresses this challenge by aiming to demonstrate that the combination of DRL and model-based control improves the performance and stability of an autonomous dynamic system subject to an actuator fault compared to existing methods. To support this statement, the chosen methodology involves performing control reallocation using DRL within a model-based control framework, integrating DRL's adaptability with the stability benefits of model-based methods. DRL redistributes forces by adjusting the control allocation matrix, with local stability ensured through recalculated control gains in a model-based framework. Trained in simulation to address varied fault scenarios, the DRL model provides real-time control reallocation in the case of faults. This method is applied to a BlueROV2 Heavy, which is an autonomous underwater vehicle equipped with multiple thrusters, in both simulation and real-world experiments. Results demonstrate the effectiveness of DRL-based control reallocation for thruster faults in the BlueROV2 Heavy, outperforming non-model-based and non-DRL adaptive methods. Additionally, the results highlight the DRL model's generalisation capacity, as it successfully manages fault scenarios not encountered during training. These findings support a promising FTC approach to actuator fault tolerance that combines DRL's adaptability with the stability of model-based control, without any need for explicit fault diagnosis.

Contents

1 Introduction – From dynamic systems to fault management **20**
1.1 Fault Tolerant Control strategies 23
1.2 Problem statement 25
1.3 Thesis outline 27

2 Fault-tolerant control in autonomous dynamic systems **31**
2.1 Dynamic system control 31
2.2 Fault Tolerant Control strategies in the literature 40
2.3 Deep Reinforcement Learning-based Fault Tolerant Control strategies 52
2.4 Limits of existing Fault Tolerant Control methods - Gap analysis 61

3 Hybrid approach for actuator fault recovery: combining DRL and model-based control **65**
3.1 Core concept of the proposed approach 66
3.2 Deep Reinforcement Learning-based control reallocation 70
3.3 Integration of the adjusted Control Allocation Matrix in the model-based control framework 74
3.4 Conclusion 76

4 Implementation of the proposed approach on an Autonomous Underwater Vehicle **79**
4.1 BlueROV2 Heavy 80
4.2 Scenario and control design 86
4.3 Deep Reinforcement Learning algorithm selection and training process 91
4.4 Conclusion 98

5 Simulation results and real-world experiments on an Autonomous Underwater Vehicle **101**
5.1 Performance evaluation methodology 101
5.2 Simulation results 106

5.3 Experiments with a physical BlueROV2 Heavy 113

6 Conclusion **129**
6.1 Thesis outcomes 129
6.2 Research perspectives 132

Contributions/papers **135**

Appendices **152**

A UUV Thruster faults **152**

B Linearisation of BlueROV2 Heavy model **154**

C End-to-end DRL control method **161**
C.1 Environment setup 161
C.2 Simulation Results 163

List of Figures

1.1 Examples of recent autonomous dynamic systems. 21
1.2 Types of faults in autonomous dynamic systems. 21

2.1 Control autonomy of a dynamic system. 32
2.2 Control process of a dynamic system with multiple actuators. 32
2.3 Illustration of local and global stability. 38
2.4 Fault Tolerant Control (FTC) methods classification 41
2.5 Indirect adaptive FTC strategy steps 48
2.6 Deep Reinforcement Learning architecture general overview: a combination of Reinforcement Learning and Neural network. 55
2.7 Example of a transformation from $\mathbf{h}_{l-1}$ (with 3 neurons) to $\mathbf{h}_l$ (with 2 neurons). 56
2.8 End-to-end Deep Reinforcement Learning (DRL) for fault tolerance in autonomous dynamic systems. 59
2.9 Limits of existing methods, based on FTC key characteristics and literature review. 62

3.1 Fault detection scheme. 66
3.2 Hybrid controller structure overview. 67
3.3 Control law and control allocation scheme of the considered system. 69
3.4 Input and output of the DRL model. 72
3.5 Detailed proposed hybrid controller structure. 76

4.1 BlueROV2 design (source: BlueRobotics). 80
4.2 Representation of the six degrees of freedom (X. Li 2022). 81
4.3 Thruster configuration (source: BlueRobotics). 81
4.4 BlueROV2 Heavy model with Unmanned Underwater Vehicle (UUV) Simulator. 83
4.5 Illustration of the marine environment with UUV Simulator (Manhães et al. 2016). 83
4.6 Architecture of the training environment. 84
4.7 Scenario for the DRL model training. 86

4.8 Convergence curve of Particle Swarm Optimisation (PSO) technique to determine the **Q** matrix. 90
4.9 End-to-end DRL control training process. 93
4.10 Domain randomisation process during DRL training, applied to BlueROV2 Heavy. "p" is the probability rate for each case. 95
4.11 Delayed feedback process. 96
4.12 DRL model mean reward curve over number of steps. 97
4.13 Evolution of Control Allocation Matrix (CAM) weights, where $T0$'s efficiency is at 0% during one entire episode, before (left) and after (right) application of the filter. 98
4.14 Embedding of the trained DRL model in the control architecture. 98

5.1 Trajectory of the BlueROV2 Heavy during the evaluation process. 102
5.2 DRL-based Proportional-Integral-Derivative (PID) control scheme. 104
5.3 Steps of the evaluation process. 105
5.4 Time evolution of thruster outputs in case of $T0$ at 0% efficiency with the conventional adaptive controller (left) and the DRL-based controller (right). 111
5.5 Trajectory of BlueROV2 Heavy with Unity. Thruster $T0$ is at 10% efficiency. In red: BlueROV2 Heavy trajectory using the standard Linear–Quadratic Regulator (LQR) controller, and in blue: using the DRL-based LQR controller. 114
5.6 BlueROV2 Heavy in the test pool, with the localisation system. 115
5.7 Diagram illustrating the implementation of DRL-based FTC strategies on the physical BlueROV2 Heavy. 115
5.8 Ideal configuration and used configuration to inject the input commands to the physical drone. 116
5.9 Screenshot of the tools used to visualise the drone's trajectory during the experiments. 117
5.10 Mean of y error, with standard deviation across the number of episodes (or rosbags) considered. 118
5.11 Mean and Standard Deviation of the 2D Trajectory of the BlueROV2 Heavy controlled by the standard PID (left), conventional adaptive PID (middle) and DRL-based PID (right) controllers at 0% efficiency of $T0$, over 10 evaluation tests. 122
5.12 Mean and Standard Deviation of the 2D Trajectory of the BlueROV2 Heavy controlled by the standard PID (left), conventional adaptive PID (middle) and DRL-based PID (right) controllers at 40% efficiency of $T0$, over 10 evaluation tests. 122

5.13 Mean and standard deviation of the 2D trajectory of the BlueROV2 Heavy controlled by the standard LQR (left), conventional adaptive LQR (middle) and DRL-based LQR (right) controllers at 0% efficiency of $T1$, over 10 evaluation tests. 124
5.14 Mean and standard deviation of the 2D trajectory of the BlueROV2 Heavy controlled by the standard LQR (left), conventional adaptive LQR (middle) and DRL-based LQR (right) controllers at 40% efficiency of $T1$, over 10 evaluation tests. 125
5.15 Mean and standard deviation of the 2D trajectory of the BlueROV2 Heavy controlled by the standard LQR (left), conventional adaptive LQR (middle) and DRL-based LQR (right) controllers at 0% efficiency of $T2$, over 10 evaluation tests. 125
5.16 Mean and standard deviation of the 2D trajectory of the BlueROV2 Heavy controlled by the standard LQR (left), conventional adaptive LQR (middle) and DRL-based LQR (right) controllers at 40% efficiency of $T2$, over 10 evaluation tests. 125

B.1 BlueROV2 Heavy: Six degrees of freedom and transformation between body-fixed and NED frames. 154

C.1 Mean of the PID-method performance index γ over 100 episodes with the associated standard deviation. 164
C.2 Mean episodic training reward (averaged over 100 episodes) of the Soft Actor-Critic (SAC) method. The x-axis represents the number of steps, and the y-axis represents the reward value. 164
C.3 Mean of the SAC-method performance index γ over 100 episodes with the associated standard deviation. 164
C.4 Autonomous Underwater vehicle (AUV) trajectory controlled by PID in the absence of fault. 166
C.5 AUV trajectory controlled by PID, where $T5$ efficiency rate is 0.2%. 166
C.6 AUV trajectory controlled by the DRL-based controller, where $T5$ efficiency rate is 0.2%. 166
C.7 Evolution of control input forces with a PID controller, without fault. 167
C.8 Evolution of control input forces with an end-to-end DRL controller, without fault. 167

List of Tables

4.1 Chosen hyperparameters for the SAC algorithm. 92

5.1 Mean and standard deviation of performance indices for the 4 thrusters evaluated individually from 0% (total fault) to 100% efficiency with PID, conventional adaptive PID and DRL-based PID (referred to as PID/DRL in the table) in simulation. 108

5.2 Mean and standard deviation of performance indices for the 4 thrusters evaluated individually from 0% (total fault) to 100% efficiency with standard LQR, conventional adaptive LQR and DRL-based LQR controllers, evaluated over 100 episodes in simulation for each case. 110

5.3 Performance indices for $T0$ and $T2$ at 0%, 40% and 70% efficiency with standard PID, conventional adaptive PID, and DRL-based PID controllers on a physical BlueROV2 Heavy, over 10 evaluation tests. 119

5.4 p-value computation for y error across efficiency levels from 0% to 70%, using physical drone data. 120

5.5 p-value computation for yaw error across efficiency levels from 0% to 70%, using physical drone data. 121

5.6 Mean and standard deviation of performance indices for the 4 thrusters evaluated individually, with 0%, 10%, 40%, and 70% efficiency with standard LQR, conventional adaptive LQR and DRL-based LQR controllers, evaluated over 10 episodes on a physical BlueROV2 Heavy for each case. 123

5.7 p-value computation for y error across efficiency levels from 0% to 70%, using physical drone data. 123

5.8 p-value computation for yaw error across efficiency levels from 0% to 70%, using physical drone data. 124

A.1 Examples of natural events that caused AUV thruster faults. 152

A.2 Detailed types of AUV thruster faults due to environmental factors. 153

B.1 Rigid-Body parameters (C.-J. Wu 2018). 156

B.2 Added-Mass Parameters (C.-J. Wu 2018). 157
B.3 Linear and Quadratic Damping Parameters (C.-J. Wu 2018). 158

C.1 Calculation of the Two-Sample Mean Comparison Test $z_0 - z_{\frac{\alpha}{2}}$ for all fault cases, to compare the PID controller and the DRL-based controller. Positive values demonstrate the significance of the result. 165

Glossary

AUV Autonomous Underwater vehicle. 13, 15, 20, 28, 46, 60, 77, 79, 80, 82, 83, 86, 87, 93, 122, 124, 126, 131–133, 152–155, 158, 161–163, 165, 166

CAM Control Allocation Matrix. 12, 33, 46, 47, 66, 68–72, 74–77, 81, 90–92, 96–98, 103, 104, 116, 126, 130, 132, 133

DDPG Deep Deterministic Policy Gradient. 57, 92

DNN Deep Neural Network. 55, 59

DoF Degree of Freedom. 22, 71, 90, 112, 132

DRL Deep Reinforcement Learning. 11–13, 15, 16, 25–28, 31, 52, 55–61, 63, 65–68, 70, 72–77, 80, 82–84, 86, 87, 90, 91, 93–99, 102–127, 129–134, 161, 163, 165–167

EKF Extended Kalman Filter. 49, 50

FTC Fault Tolerant Control. 11, 22–29, 31, 40–43, 45, 48, 50–52, 58–63, 68, 91, 101, 103, 129, 130, 132

IMU Inertial Measurement Unit. 44, 80, 115

LQR Linear–Quadratic Regulator. 12, 13, 15, 28, 36, 37, 39, 44, 60, 61, 75, 87–91, 94, 98, 99, 103–106, 109–114, 116, 117, 122–127, 131, 132, 154, 159

LSTM Long Short-Term Memory. 59, 132

MDP Markov Decision Process. 52, 53

MIT Massachusetts Institute of Technology. 43, 44, 103–105

MPC Model Predictive Control. 37, 39

MRAC Model Reference Adaptive Control. 44, 45

NN Neural Network. 25, 26, 51, 55, 56, 59, 68

PID Proportional-Integral-Derivative. 12, 13, 15, 16, 33, 34, 44, 60, 61, 92, 93, 103–113, 116, 118–123, 127, 131, 163–167

PPO Proximal Policy Optimisation. 57, 58, 91

PSO Particle Swarm Optimisation. 12, 51, 89, 90

RL Reinforcement Learning. 52–55, 57, 61

ROS Robot Operating System. 83–85, 116

ROV Remotely Operated Vehicle. 80, 86–88, 92, 95

SAC Soft Actor-Critic. 13, 15, 57, 58, 60, 91–93, 99, 131, 164

UAV Unmanned Aerial Vehicle. 20, 22, 42, 50, 51, 59

UUV Unmanned Underwater Vehicle. 11, 83, 96, 107, 109, 113

Chapter 1

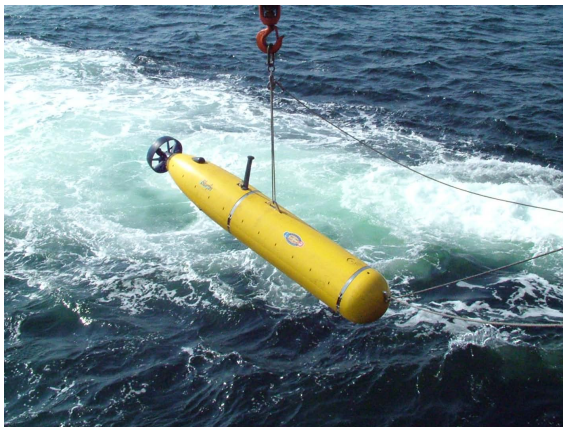
Introduction – From dynamic systems to fault management

Autonomous dynamic systems (Watson and Scheidt 2005) are the foundation of many modern technologies, ranging from robotic arms in manufacturing or surgery to autonomous vehicles, which can operate in aerial, terrestrial, marine, or underwater environments (Schwarting, Alonso-Mora, and Rus 2018; Attia et al. 2018). In robotics, a dynamic system is a system equipped with actuators whose state evolves over time in response to inputs, external conditions and internal processes. A dynamic system is considered autonomous when it uses its internal dynamics and available information obtained from onboard sensors about its current state to generate control actions that guide it towards a desired state, without requiring human intervention (Khalil 2002). The mission of an autonomous dynamic system is generally to move within its environment in order to carry out tasks such as transport, exploration, or surveillance (Goodrich, Schultz, et al. 2008; Thor I. Fossen, Tor Arne Johansen, and Perez 2009).

In the 20th century, advances in control theory laid the theoretical foundation for modern autonomous systems, covering aerial and ground vehicles as well as industrial automation (Wiener 1948). In recent decades, progress in artificial intelligence and sensor technology has led to the emergence of highly sophisticated autonomous platforms, such as Unmanned Aerial Vehicles (UAVs) (Grigorescu et al. 2020) and Autonomous Underwater vehicles (AUVs) (Sahoo, Dwivedy, and Robi 2019) (see Figure 1.1).



(a) Fox-C8 XT drone in flight (ZullyC3P 2015).



(b) Unmanned Underwater Vehicle during a US Navy exercise (Robotics 2006).

Figure 1.1: Examples of recent autonomous dynamic systems.

During their missions, autonomous dynamic systems can encounter faults, that are defined as undesired changes in at least one component of the system (Christensen 2008). The presence of faults poses significant risks that may compromise mission success and system performance (Lussier et al. 2005). Figure 1.2 lists the different types of faults occuring in dynamic autonomous systems that we can find in the literature (Blanke, Staroswiecki, and N. E. Wu 2001; Christensen 2008; Lussier et al. 2005).

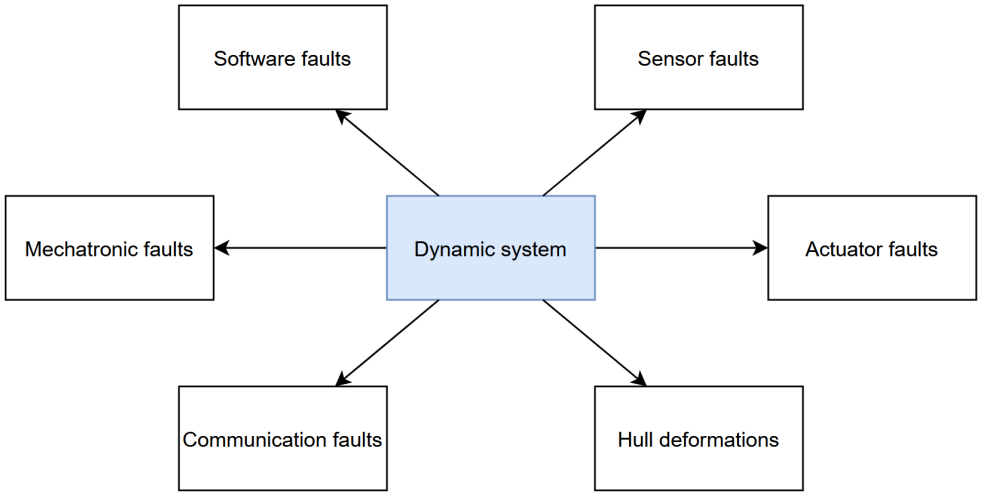


Figure 1.2: Types of faults in autonomous dynamic systems.

In this thesis, **the focus is on actuator faults which can have significant consequences on control autonomy** (Abbaspour et al. 2020): an actuator fault can cause a loss of control leading to degraded system performance such as an inability to follow the intended trajectory, or, in more severe cases, to mission failure (Ernits, Dearden, and Pebody

2010). Fault situations become even more critical in cases where the system operates in inaccessible locations, as it can cause the loss of the system with no possibility of operator intervention. The causes of actuator faults vary depending on the dynamic system and mission conditions, and can arise from environmental conditions (Park et al. 2024) as well as from internal fragilities like electrical malfunctions in system components (Abbaspour et al. 2020).

An actuator fault can be a *partial fault* or a *total fault* (Omerdic and Roberts 2004):

- A *partial fault*: this occurs when the actuator remains functional but experiences degraded effectiveness. For example, in underwater dynamic systems, corrosion due to salty water or damaged thruster blade due to a collision can reduce thrust efficiency while still allowing the system to operate (Alessandri, M. Caccia, and Veruggio 1999; Ernits, Dearden, and Pebody 2010).

- A *total fault*: this type of fault is characterised by the permanent shutdown of the actuator. For example, in an UAV, a broken rotor could lead to a loss of control, rendering the drone unable to maintain flight and increasing the risk of a crash (Saied et al. 2020).

Controllers designed solely for nominal conditions may not always ensure the desired behaviour in the presence of actuator faults (Blanke 2005). Addressing the impact of actuator faults requires appropriate control strategies to maintain the desired system motion, which involves maintaining control over the forces applied to each Degree of Freedom (DoF) of the system. In this context, the following assumption is made: **the system remains controllable despite actuator faults.** This assumption is necessary to ensure that, with an appropriate control strategy, maintaining control of the system is possible even in the presence of actuator faults. The remaining actuators, whether fully or partially functional, are assumed to allow control over all intended degrees of freedom despite the faults.

As the focus is on actuator faults, it is also assumed that **there are no other disturbances during the mission**: in particular, the dynamic system is considered free from wind or sea currents. In general, overcoming an actuator fault requires fault detection and a response that mitigates its effects by relying on the remaining actuators to maintain the desired behaviour. In the literature, the set of control strategies designed to achieve such objectives is referred to as Fault Tolerant Control (FTC) (Patton 2021), which will be explored in the next section.

1.1 Fault Tolerant Control strategies

The focus here is on FTC strategies for actuator faults in autonomous dynamic systems. This section begins by outlining the key characteristics of FTC strategies, showing how these approaches contribute to overcoming actuator faults, and then presents a historical overview along with their limitations.

1.1.1 Key characteristics of Fault Tolerant Control strategies

The aim of FTC strategies is to address several key characteristics which have been identified based on a review of several studies in the literature (Abbaspour et al. 2020; Noura et al. 2009; Patton 2021):

- Optimal recovery performance: The controller's ability to obtain the best achievable performance of an autonomous system considering the constraints and limitations imposed by the fault.

- Stability: The system's ability to return to its equilibrium state after being subjected to state deviations. A system is considered stable if, for any bounded input, the output remains bounded, and the system's state variables converge to a steady state in the absence of further disturbances (Atassi and Khalil 1999). Actuator faults can significantly disrupt the system's dynamics, potentially compromising the stability of the controlled system.

- Adaptability: The extent of the range of faults that the FTC strategy is capable of handling. This is crucial in the case where actuators can be numerous, and faults can affect multiple actuators simultaneously (De Kleer and Williams 1987), leading to a rapidly increasing number of potential faults.

- Generalisability: The system's ability to handle unforeseen faults, *i.e.* the ability of the control system to manage faults it was not specifically configured to overcome. This capability is needed for a FTC strategy when the faults are potentially unpredictable, notably due to the various shapes they can take.

- Resource-efficiency: The ability of a system to manage its operations without consuming excessive computational resources or energy. This is essential for many autonomous systems that operate in environments where resources are limited, such as underwater or space exploration. In the case of actuator faults, the FTC strategy must be performed in real-time, potentially with constrained computational power and energy.

These criteria characterise the *overall performance* of an autonomous dynamic system subject to an actuator fault and controlled by a fault-tolerant controller. FTC strategies have been extensively studied over the years, particularly as dynamic systems have increased in complexity, and reliability has become a critical concern. Although research in this field continues to evolve, significant limitations remain with respect to the criteria outlined above. These limitations are summarised here and discussed in detail in Chapter 2.

1.1.2 History and limits of Fault Tolerant Control strategies

One of the earlier approaches to FTC is robust passive control (Benosman 2011), developed in the 1980s to handle predefined actuator fault scenarios without requiring real-time adaptation (H. Wang et al. 2014). A key contribution in this domain is the use of H_∞ control techniques (Youmin Zhang and Jiang 2008), which aim to preserve system stability and performance within a bounded fault set. However, as the range of anticipated faults increases, control performance generally degrades (Jiang and X. Yu 2012), and such methods can become ineffective when dealing with unforeseen faults (Youmin Zhang and Jiang 2008). As these approaches do not adapt the controller based on the fault that actually occurs, their ability to respond appropriately to fault situations is limited.

To address the need for real-time adjustment to changing fault conditions, indirect adaptive control methods for FTC were developed by the late 1980s (P. Ioannou and Sun 1988). This technique updates the system model to better reflect the fault conditions (Gao, Cecati, and Steven X. Ding 2015). For this approach, an explicit fault diagnosis (Escobet et al. 2019), which involves identifying the type and the extent of a fault, is needed and discussed in Patton (1994).

Explicit fault diagnosis can be based on prior knowledge or on online estimation. The effectiveness of explicit fault diagnosis based on accurate prior knowledge, in terms of adaptability and generalisability, is limited by the amount of prior knowledge available about faults and their effects on the system behaviour (Patton 1994). This is particularly a challenge for systems with numerous actuators: as the number of different actuator faults increases, the knowledge required to accurately diagnose these faults escalates rapidly. In the case of explicit fault diagnosis based on online fault estimation, these methods do not require accurate prior knowledge about faults, but are computationally demanding as they incorporate real-time parameter estimation and optimisation (Poli, Kennedy, and Blackwell 2007).

By the early 1990s, direct adaptive control methods have been explored (Sanner and Slotine 1991), allowing the system to directly adjust controller gains in response to observed errors without any need for explicit fault diagnosis or modifying the system model (Petros A Ioannou and Jing Sun 2012). This type of methods aims to provide an immediate

response to faults. Direct adaptive control methods are straightforward to implement but often lack the precision needed for optimal recovery, as they do not modify the system model to reflect changing fault conditions (Petros A Ioannou and Jing Sun 2012). This can result in suboptimal recovery performance, particularly in complex systems with multiple actuators, where the dynamic interactions are more intricate and fault effects more difficult to compensate. In such systems, the inability to explicitly account for the altered dynamics caused by faults can severely limit the effectiveness of the controller, leading to degraded performance or even instability in more demanding scenarios (Petros A Ioannou and Jing Sun 2012).

To address the limitations of the previously mentioned methods, machine learning-based approaches have been explored (Quamar and Nasir 2024), along with their own challenges, forming the basis of the problem statement.

1.2 Problem statement

The main limitations of existing methods are that passive approaches face difficulties with unforeseen faults, indirect adaptive methods rely on explicit fault diagnosis, which is challenging to obtain, and direct adaptive methods lack model adaptation, which can limit recovery performance. These limitations have motivated the investigation of FTC methods using Deep Reinforcement Learning (DRL) (Mnih et al. 2015).

The goal of DRL is to obtain an agent trained to act in a specific environment by trial and error. Through repeated experiences, the agent is rewarded for actions that align with its objectives and penalised for those that do not, allowing it to improve its behaviour to handle diverse scenarios (Sutton 1998).

The main advantages of using DRL are its potential to bypass the need for explicit fault diagnosis and its capacity for adaptation and generalisation. Rather than identifying faults explicitly, DRL-based FTC strategies leverage the observed behaviour of the system to react and adapt accordingly, aiming to achieve optimal recovery performance without relying on predefined fault models (L. Liu, Zhanshan Wang, and H. Zhang 2016). This makes it particularly suitable for applications where explicit diagnosis is challenging to obtain, such as systems with numerous actuator and complex geometry.

DRL adaptability stems from training across diverse fault situations, enabling it to develop a rich knowledge base. The architecture of Neural Networks (NNs) enables a wide range of information to be processed, facilitating the learning of different situations and the associated responses (Ahmadzadeh, Kormushev, and Caldwell 2014; Fei et al. 2020; Hadi, Khosravi, and Sarhadi 2022). In addition, the DRL has a capacity for generalisation, which is particularly suitable in the case of actuator faults: it is able to find solutions to faults it

has never seen before, as long as the behaviour of the faulty system is similar to a situation it has encountered during its training. This is because NNs are function approximators, which can be trained to predict the most appropriate action in a given context. Even when faced with novel situations, the agent uses these approximators, whose effectiveness depends on the similarity between the current situation and the situations encountered in training (Kohl and Miikkulainen 2009).

Besides, once trained, DRL-based methods generally have low computational demands (Artusi, Chaillan, and Napoli 2021), making them suitable for deployment on edge computing platforms. This characteristic directly contributes to the resource-efficiency of the overall FTC strategy.

However, a drawback of employing a purely DRL-based controller is the complexity in guaranteeing system stability. This challenge arises due to the integration of NNs in DRL which, due to their non-linearity, make it hard to predict the system behaviour (Dulac-Arnold, Mankowitz, and Hester 2019).

To address this limitation, the proposed approach is to add a model-based control structure to the DRL. Controllers based on the model of the system rely on physical equations that accurately depict the system's behaviour. By evaluating these mathematical models, stability properties can be deduced, ensuring a reliable control structure (Brosilow and Joseph 2002). Combining model-based control with DRL methods takes advantage of the data-driven adaptability of DRL and the stability guarantee based on model-based control principles (Bhan, Quinones-Grueiro, and Biswas 2021). This hybrid approach has been explored in the literature (Bhan, Quinones-Grueiro, and Biswas 2021; Rizvi and Z. Lin 2019; Yaghmaie, Gustafsson, and Ljung 2022), but there is a gap concerning its use in addressing actuator faults in autonomous dynamic systems without explicit fault diagnosis, presenting an opportunity for novel contributions in this area.

Thus, the present work aims to provide elements to support the following research statement:

Research statement: **The combination of DRL and model-based control improves the overall performance of an autonomous dynamic system subject to actuator faults compared to existing methods.**

The research objective of this thesis is to develop a hybrid control system that uses the strengths of DRL in the framework of a controller based on the model of the system. The purpose for the DRL component is to enhance the controller's recovery performance, leveraging its key advantages such as adaptability, generalisability, the ability to approach optimal performance over time and low resource consumption once trained. The purpose for the model-based controller is to ensure system stability, addressing the limitation of DRL

for system control.

To achieve this objective, the proposed method is to develop a strategy allowing to perform a redistribution of the control commands among the actuators using a DRL method within a model-based control framework, without relying on any explicit fault diagnosis. The development of this hybrid control strategy requires tackling several key challenges, which can be formulated as the following research questions:

- **Question 1:** How to integrate a DRL method in a model-based controller framework in order to minimise the effects of the actuator faults on the behaviour of the vehicle?
- **Question 2:** How a DRL model can be trained to allow the hybrid controller to overcome a wide range of actuator faults, without explicit fault diagnosis?
- **Question 3:** How can system stability be guaranteed while integrating a DRL method in the control loop, in a context where dynamic behaviours are unpredictable?
- **Question 4:** How to evaluate the impact of combining DRL and model-based control on fault recovery efficiency compared to existing methods?

Given these objectives and challenges, the next section will provide an outline of the thesis and a summary of the key topics covered in each chapter.

1.3 Thesis outline

This thesis is organised into six chapters, each of which progressively builds towards the development and validation of a hybrid control approach for fault-tolerant control in autonomous dynamic systems. The following sections provide an overview of each chapter, outlining their content and how they contribute to the overall research objective.

Chapter 2 - Fault-tolerant control in autonomous dynamic systems.

This chapter aims to explain the concepts needed to understand the research statement. It begins by introducing key concepts in dynamic system control in section 2.1 to understand the incidence of actuator faults at the control autonomy level. The consequences of actuator faults lead to the need to use FTC methods. The different categories of FTC methods are set out in section 2.2 and their limitations are highlighted. These limitations lead to the exploration of DRL methods. Section 2.3 starts with introducing some key elements of DRL, then details the existing FTC methods based on DRL and shows the interest of using a control framework based on the system model. Section 2.4 summarises the limitations identified in the literature, which lead to the proposed approach.

Chapter 3 - Hybrid approach for actuator fault recovery: combining Deep Reinforcement Learning and model-based control.

A proposed approach is developed to overcome the limitations of existing FTC methods, aiming to support the research statement and to address the research questions 1, 2, and 3. The proposed approach is based on methodological choices that rely on the conclusions of Chapter 2. The structure of the proposed hybrid controller is introduced in section 3.1, using DRL within the framework of a model-based controller in order to reallocate the forces among the remaining actuators. Section 3.2 highlights how DRL can be used to redistribute forces in response to actuator faults. This section also covers the DRL training method, which is designed to take advantage of the generalisability and adaptability of DRL. Finally, the integration of the control reallocation process into the model-based controller is discussed in section 3.3. To validate the theory, tests were carried out on an autonomous system.

Chapter 4 - Implementation of the proposed approach on an Autonomous Underwater Vehicle.

The case study of thruster faults in an Autonomous Underwater vehicle (AUV) is presented in order to validate the proposed approach on an autonomous dynamic system. Actuator faults in AUVs are particularly challenging because the drones can operate in underwater locations that are difficult for an operator to access. Section 4.1 focuses on the modelling of the chosen AUV, BlueROV2 Heavy (from BlueRobotics), and details the DRL training environment in GAZEBO Simulator (Koenig and Howard 2004). Section 4.2 covers the scenario considered for testing the proposed approach and details the model-based control design for the drone. The chosen model-based control for this application case is the LQR controller (Benner 2006). In section 4.3, the application of the proposed approach to BlueROV2 Heavy is detailed, and the selection of the DRL algorithm is made. This application case leads to results in simulation and real-world conditions.

Chapter 5 - Simulation results and real-world experiments on an Autonomous Underwater Vehicle.

The results of the proposed approach on the BlueROV2 Heavy are presented, both in simulation and on a real drone, addressing the research question 4. First, section 5.1 details the evaluation method, including the criteria used to assess the overall performance, the types of faults considered, and comparisons with other control methods such as variable-gain controllers. The simulation results are then presented in section 5.2, followed by experimental results in section 5.3. The experiments were carried out in the Flinders University test tank.

The results are discussed and demonstrate the performance of the proposed controller in both simulation and real-world scenarios.

Chapter 6 - Conclusion.

The key findings of the thesis are summarised, with a reflection on the contributions of the proposed hybrid controller to the field of FTC for autonomous dynamic systems. Section 6.1 revisits the research objective and evaluates how well it has been met through the development, application, and testing of the controller. The performance of the hybrid control system is critically assessed, highlighting both its strengths and its limitations. Section 6.2 discusses the potential for further improvements, including possible future research directions.

Chapter 2

Fault-tolerant control in autonomous dynamic systems

The aim of this chapter is to explore the key concepts and existing methods of FTC regarding actuator faults. To begin with, section 2.1 will focus on dynamic system control in order to gain a better understanding of the consequences of actuator fault on both system performance and stability. To limit these consequences, existing FTC strategies will be explored and their limitations established in section 2.2. These limitations will lead to a focus on DRL-based FTC strategies that will be made in section 2.3. Finally, the concluding section 2.4 will summarise the limitations of the existing strategies, forming the basis of the methodological choices of the proposed approach.

2.1 Dynamic system control

This section provides the information that is needed to understand dynamic system control, starting with the closed-loop control in section 2.1.1. A focus on model-based controllers in section 2.1.2 will be made, as their advantage is to obtain optimal performance and to guarantee the stability of the system. The methods to establish stability will be explored in section 2.1.3 and the consequences of actuator faults on control and stability will be discussed in section 2.1.4.

2.1.1 Closed-loop control

Closed-loop control (R. C. D. R. H. Bishop 2011), also known as feedback control, is a fundamental concept in dynamic system control. Under this control scheme, the system continuously monitors its output and adjusts its inputs to minimise the difference between the desired reference and the actual output. This ability to self-correct forms the basis of control

autonomy (Siegwart, Nourbakhsh, and Scaramuzza 2011), where the system can operate without human intervention by dynamically responding to changes in its environment.

Figure 2.1 illustrates the structure of the closed-loop process (Franklin et al. 2002). The reference vector, denoted by $\mathbf{w_{ref}}$, represents the desired position, orientation and velocities of the system. This reference vector is passed to the controller, which processes the data and generates actuator inputs $\mathbf{u}$. These inputs are sent to the actuator to apply the necessary forces or torques. The system's internal dynamics then govern how these forces influence the system's motion. The observation vector containing sensor data, denoted as $\mathbf{y_{obs}}$, is sent back to the controller to continuously adjust the actions and ensure that the actual motion follows the desired one.

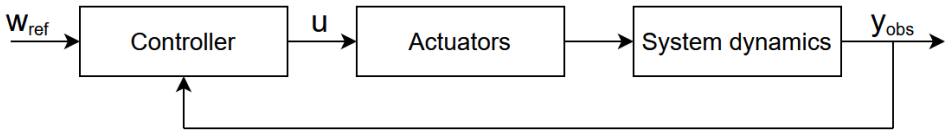


Figure 2.1: Control autonomy of a dynamic system.

Closed-loop control is crucial in autonomous dynamic systems, where feedback aims to ensure that a system can follow a trajectory despite environmental perturbations (Karl Johan Åström and Murray 2021).

In dynamic systems, the controller plays a dual role as shown in Figure 2.2.

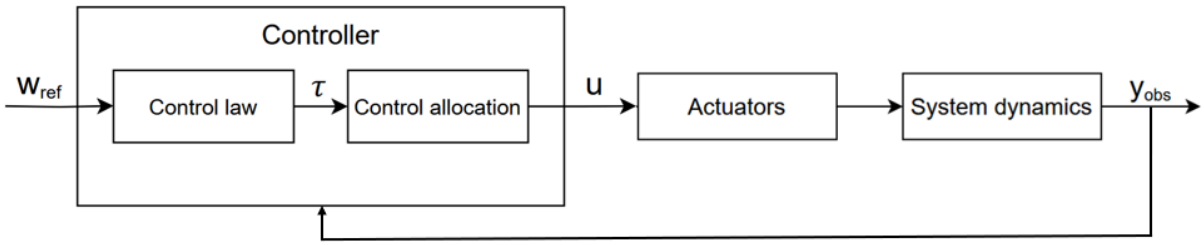


Figure 2.2: Control process of a dynamic system with multiple actuators.

First, it defines the control law (Siegwart, Nourbakhsh, and Scaramuzza 2011), a computational procedure that determines the control input $\boldsymbol{\tau}$ based on the error between the reference $\mathbf{w_{ref}}$ and observed states $\mathbf{y_{obs}}$, in order to produce the forces required to minimise this error.

In systems with multiple actuators, there is an additional step known as control allocation (Tor A Johansen and Thor I Fossen 2013). Control allocation is the process of distributing the control inputs calculated by the control law among the actuators in a

way that achieves the desired control forces. The allocation process aims to ensure that the forces are distributed among the actuators according to the geometry of the system, while respecting the physical limitations of each actuator. We express the control allocation process by:

$$\boldsymbol{\tau}(t) = \mathbf{H}\mathbf{u}(t) \tag{2.1}$$

where $\mathbf{H}$ is the Control Allocation Matrix (CAM) which maps the actuator inputs $\mathbf{u}(t)$ to the control forces in $\boldsymbol{\tau}(t)$. For a given vector $\boldsymbol{\tau}(t)$, the actuator inputs $\mathbf{u}(t)$ are computed as follows (Thor I Fossen 1999):

$$\mathbf{u}(t) = \mathbf{H}^*\boldsymbol{\tau}(t) \tag{2.2}$$

where $\mathbf{H}^*$ is the Moore-Penrose pseudo inverse (Barata and Hussein 2012) defined as:

$$\mathbf{H}^* = \mathbf{H}^T(\mathbf{H}\mathbf{H}^T)^{-1} \tag{2.3}$$

The practical implementation of control allocation requires addressing the limitations imposed by actuator saturation denoted s_i, which represents the maximum allowable magnitude of the i-th actuator input. The solution represented in Equation 2.2 is said feasible, for n actuators, if the actuator input vector $\mathbf{u}$ belongs to the set Ω expressed as follows:

$$\Omega = \{\mathbf{u} \in \mathbb{R}^n \mid \forall i \in \{1, 2, \ldots, n\},\ |u_i| \leq s_i\} \tag{2.4}$$

Depending on the control input $\boldsymbol{\tau}$ and actuator saturation, using Equation 2.2 may be unfeasible. To make the solution feasible, a common approach is truncation (Omerdic and Roberts 2004):

$$\forall i \in \{1, 2, \ldots, n\},\ u_i = \max(-s_i, \min(u_i, s_i)) \tag{2.5}$$

Regarding the control law, the Proportional-Integral-Derivative (PID) controller (Karl J Åström and Hägglund 2006), widely used in closed-loop systems, is introduced here as the foundation upon which many control strategies are built. In this thesis, we are focusing on continuous-time system models, as they provide a fundamental theoretical framework for analysing system dynamics and designing control laws.

The continuous-time control law of a PID controller is expressed by the following equation:

$$\boldsymbol{\tau}(t) = \mathbf{K_p}\mathbf{e}(t) + \mathbf{K_i}\int_0^t \mathbf{e}(\boldsymbol{\tau})d\boldsymbol{\tau} + \mathbf{K_d}\frac{d\mathbf{e}(t)}{dt} \tag{2.6}$$

where $\mathbf{e}(t)$ is the error, defined as the difference between the reference $\mathbf{w_{ref}}(t)$ and the observation $\mathbf{y_{obs}}(t)$, $\mathbf{K_p}$ is the proportional gain, $\mathbf{K_i}$ is the integral gain and $\mathbf{K_d}$ is the derivative gain (Nise 2020).

The PID controller combines three actions: the proportional term improves respon-

siveness by reacting to the current error; the integral term addresses steady-state error by accounting for the accumulation of past errors, though it may cause overshoot; and the derivative term anticipates future error based on its rate of change, helping reduce overshoot but being sensitive to noise (Karl Johan Åström and Murray 2021).

The tuning of the gains $\mathbf{K_p}$, $\mathbf{K_i}$, and $\mathbf{K_d}$ plays a critical role in the controller's effectiveness. One common heuristic method for tuning these parameters is the Ziegler-Nichols method (Patel 2020), which provides an approach to finding values for the PID gains. This approach involves setting $\mathbf{K_i}$ and $\mathbf{K_d}$ to zero and gradually increasing $\mathbf{K_p}$ until the system reaches a point of sustained oscillation, called the ultimate gain. The period of these oscillations, known as the ultimate period, is then measured. Based on these values, the Ziegler-Nichols method provides recommended formulas to set the gains for a PID controller.

While the Ziegler-Nichols method offers a quick way to tune PID controllers, it relies on trial-and-error techniques that do not account for the underlying dynamics of the system. As a result, Ziegler-Nichols tuning can't guarantee the stability of the closed-loop control (Karl J Åström and Wittenmark 2013). To overcome these limitations, model-based control (Brosilow and Joseph 2002) can be employed. Model-based control uses a mathematical model of the system to compute optimal control gains through model-based optimisation techniques. By leveraging the knowledge of the system's dynamics, this approach can ensure the stability of the closed-loop control (Brosilow and Joseph 2002).

The following section explores the principles and advantages of model-based control, illustrating how this method can be used to achieve both optimal performance and guaranteed stability.

2.1.2 Model-based control

Unlike heuristic methods, which rely on empirical rules, model-based control refers to a control law strategy that employ a mathematical model of the system to predict its behaviour and optimise performance. This allows for optimal tuning of the control gains through optimisation methods. Model-based control is widely used in autonomous dynamic systems where reliability is essential (Rizk, Chaibet, and Kribèche 2023).

System modelling

The foundation of model-based control lies in accurately modelling the system dynamics. A mathematical model takes the form of state-space representations (Brosilow and Joseph 2002) that describe how the system's state evolves over time in response to control inputs. For linear time-invariant systems, the state-space representation includes the evolution equation

(Equation 2.7) and the observation equation (Equation 2.8) (Brosilow and Joseph 2002):

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\boldsymbol{\tau}(t) \tag{2.7}$$

$$\mathbf{y_{obs}}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\boldsymbol{\tau}(t) \tag{2.8}$$

Here, $\mathbf{x}(t)$ is the state vector, which represents the internal states of the system at time t. The matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ have the following roles:

- $\mathbf{A}$ (the system matrix) defines how the current state $\mathbf{x}(t)$ evolves over time based on its current value, encapsulating the system's inherent dynamics;
- $\mathbf{B}$ (the input matrix) determines how the control input $\boldsymbol{\tau}(t)$ affects the change in the state $\mathbf{x}(t)$;
- $\mathbf{C}$ (the output matrix) maps the state $\mathbf{x}(t)$ to the observation vector $\mathbf{y_{obs}}(t)$, reflecting how the internal states are translated into measurable outputs;
- $\mathbf{D}$ (the feedforward matrix) represents the direct effect of the control input $\boldsymbol{\tau}(t)$ on the observation vector $\mathbf{y_{obs}}(t)$, bypassing the internal state dynamics, and is often zero in systems without direct control-input-to-output relationships.

For nonlinear systems, the equations can be more complex, involving nonlinear terms:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \boldsymbol{\tau}(t)) \tag{2.9}$$

$$\mathbf{y_{obs}}(t) = g(\mathbf{x}(t), \boldsymbol{\tau}(t)) \tag{2.10}$$

In either case, obtaining an accurate model is crucial, as the performance of the controller relies on the fidelity of the model in representing the system dynamics.

System identification methods (Ljung 1998) are commonly used to derive models from data. These techniques involve estimating the mathematical structure and parameters of a system based on input-output data. For example, methods like subspace identification (Katayama et al. 2005) and prediction error methods (Keesman 2011) are widely employed to estimate the system's state-space (Ljung 1998). These methods aim to fit the system's dynamic behaviour to a model by minimising the difference between observed and predicted outputs.

When the system exhibits nonlinear behaviour, models can be challenging to work with directly. A common approach to simplify the analysis and control design is to linearise the system around an equilibrium point. Linearisation involves approximating the nonlinear system with a linear model that describes the system's behaviour in the vicinity of the equilibrium (Jakubczyk and Respondek 1980).

A system can be linearised around an equilibrium point $(\mathbf{x_e}, \boldsymbol{\tau}_e)$, where $f(\mathbf{x_e}, \boldsymbol{\tau}_e) = 0$. The linearised model is derived by expanding $f(\mathbf{x}(t), \boldsymbol{\tau}(t))$ and $g(\mathbf{x}(t), \boldsymbol{\tau}(t))$ in a Taylor series

(Asghari et al. 2022) and retaining only the first-order terms:

$$\dot{\tilde{\mathbf{x}}}(t) = \mathbf{A}\tilde{\mathbf{x}}(t) + \mathbf{B}\tilde{\boldsymbol{\tau}}(t) \tag{2.11}$$

$$\tilde{\mathbf{y}}(t) = \mathbf{C}\tilde{\mathbf{x}}(t) + \mathbf{D}\tilde{\boldsymbol{\tau}}(t) \tag{2.12}$$

where $\tilde{\mathbf{x}}(t) = \mathbf{x}(t) - \mathbf{x_e}$ and $\tilde{\boldsymbol{\tau}}(t) = \boldsymbol{\tau}(t) - \boldsymbol{\tau}_e$, and the matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ are the Jacobians of the system dynamics evaluated at the equilibrium point:

$$\mathbf{A} = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\mathbf{x_e},\boldsymbol{\tau}_e}, \quad \mathbf{B} = \left.\frac{\partial f}{\partial \boldsymbol{\tau}}\right|_{\mathbf{x_e},\boldsymbol{\tau}_e} \tag{2.13}$$

$$\mathbf{C} = \left.\frac{\partial g}{\partial \mathbf{x}}\right|_{\mathbf{x_e},\boldsymbol{\tau}_e}, \quad \mathbf{D} = \left.\frac{\partial g}{\partial \boldsymbol{\tau}}\right|_{\mathbf{x_e},\boldsymbol{\tau}_e} \tag{2.14}$$

Linearising the system allows optimisation techniques developed for linear systems to be applied near the operating point (Karl J Åström and Wittenmark 2013).

Optimisation of control input

Once the system model is defined, model-based control methods aim to guarantee optimal performance by formulating and solving an optimisation problem. A cost function is defined to reflect the desired control objectives, such as minimising tracking error or reducing control effort. Various model-based controllers for linear systems have been developed in the literature to address this optimisation problem. While pole placement is one approach (Abdelaziz and Valášek 2004), a common control technique is Linear–Quadratic Regulator (Benner 2006), which aims to minimise the quadratic cost function designed to find the optimal control input $\boldsymbol{\tau}$:

$$J = \int_0^\infty \left(\mathbf{x}(t)^T\mathbf{Q}\mathbf{x}(t) + \boldsymbol{\tau}(t)^T\mathbf{R}\boldsymbol{\tau}(t)\right) dt \tag{2.15}$$

where $\mathbf{Q}$ is a positive semi-definite matrix that penalises deviations in the state variables, called the state weighting matrix, and $\mathbf{R}$ is a positive definite matrix that penalises control effort, called the control weighting matrix.

The gain matrix of the LQR controller denoted $\mathbf{K}$ is computed by first solving the algebraic Riccati equation (Kirk 2004) given by:

$$\mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{A} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P} + \mathbf{Q} = \mathbf{0} \tag{2.16}$$

where $\mathbf{P}$ is the unique positive semi-definite solution. The $\mathbf{Q}$ and $\mathbf{R}$ matrices are often tuned to meet specific performance criteria: one common approach is to use optimisation techniques

such as Particle Swarm Optimisation (PSO) (Kennedy and Eberhart 1995) to select suitable values for $\mathbf{Q}$ and $\mathbf{R}$ by minimising a performance index — for example, the integral of squared error between the system output and the reference signal. This process transforms the tuning of $\mathbf{Q}$ and $\mathbf{R}$ into a numerical optimisation problem, where each candidate solution is evaluated through closed-loop simulations. Once $\mathbf{Q}$ and $\mathbf{R}$ are defined, the matrix $\mathbf{P}$ is computed by solving the Algebraic Riccati Equation numerically. This is done using iterative algorithms (Goodwin, Graebe, Salgado, et al. 2001).

Once $\mathbf{P}$ is determined, the optimal feedback gain $\mathbf{K}$ can be calculated as:

$$\mathbf{K} = \mathbf{R}^{-1}\mathbf{B}^{T}\mathbf{P} \tag{2.17}$$

The optimal control law for LQR is then given by the proportionnal term:

$$\boldsymbol{\tau}(t) = -\mathbf{K}\mathbf{x}(t) \tag{2.18}$$

It should be noted that this matrix $\mathbf{K}$ does not vary over time during the system's mission in the standard LQR formulation. Besides, in the context of LQR, the control input $\boldsymbol{\tau}(t)$ primarily relies on the state feedback, which is a proportional term. It is possible to extend the LQR framework to achieve a control strategy that combines proportional, integral, and derivative actions by augmenting the state vector to include integral and derivative terms (Stengel 1994), which can be beneficial in practice to eliminate steady-state error or improve transient behaviour.

While LQR is designed for linear systems, Model Predictive Control (MPC) (Kouvaritakis and Cannon 2016) is frequently employed for nonlinear systems. MPC solves a finite-horizon optimisation problem at each control step by predicting future system states based on the model. The optimisation problem takes the form:

$$\min_{\boldsymbol{\tau}} \sum_{k=0}^{N} \left(\mathbf{x_k}^{T} \mathbf{Q} \mathbf{x_k} + \boldsymbol{\tau}_k^{T} \mathbf{R} \boldsymbol{\tau}_k \right) \tag{2.19}$$

subject to:

$$\mathbf{x_{k+1}} = f(\mathbf{x_k}, \boldsymbol{\tau}_k), \quad \mathbf{x_0} = \mathbf{x}(t) \tag{2.20}$$

Here, N is the prediction horizon, and the control inputs are optimised over this horizon. Rather than calculating a fixed gain matrix $\mathbf{K}$ as in LQR methods, MPC methods determine the control input through an optimisation process such as Sequential Quadratic Programming (Quirynen and Di Cairano 2021) at each time step.

The next section will focus on how the stability of a system can be guaranteed, which is crucial for autonomous dynamic systems.

2.1.3 Ensuring stability in dynamic systems

In control theory, stability refers to the ability of a system to return to a desired equilibrium state after experiencing a state deviation (Atassi and Khalil 1999). It can be classified into two main types: local stability and global stability (Atassi and Khalil 1999). Local stability refers to the system's ability to return to its equilibrium state when subjected to small perturbations around a specific operating point. In contrast, global stability indicates that the system can return to equilibrium from any initial state, regardless of its distance from the equilibrium point. Figure 2.3 illustrates these concepts. In the left diagram, local stability is depicted, showing that trajectories converge to the equilibrium point from a small neighborhood around it, while trajectories starting farther away diverge. The right diagram represents global stability, where all trajectories, regardless of their initial conditions, converge towards the equilibrium point.

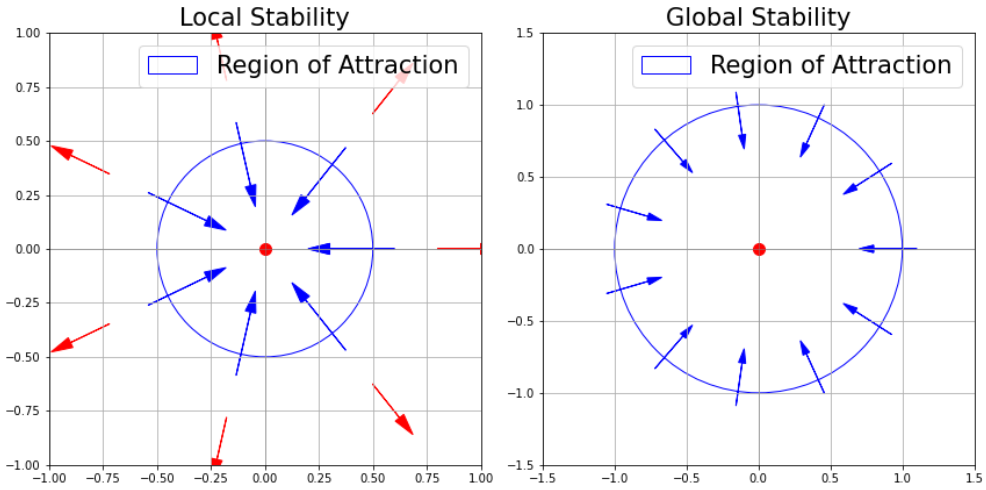


Figure 2.3: Illustration of local and global stability.

To ensure stability in control systems, various methods can be employed. One common approach is Lyapunov's method (Blanchini 1995), which involves finding a Lyapunov function $V(\mathbf{x})$ that demonstrates stability through its properties. If $\dot{V}(\mathbf{x}) < 0$ in a neighbourhood of the equilibrium point, it indicates that the system is locally stable around that point. If a Lyapunov function can be constructed such that it is positive definite in a larger region and its derivative remains negative definite, it can demonstrate global stability (Blanchini 1995).

Another widely used method is pole placement (Elliott, Cristi, and Das 1985), which involves designing feedback controllers to place the poles (the eigenvalues of the closed-loop system matrix) of the closed-loop system in stable regions of the complex plane. Another approach, known as the Terminal Cost and Constraint method (Pluymers et al. 2005), involves defining a terminal cost function and terminal constraints. The terminal cost function ensures that as the control horizon approaches the end, the predicted state is close

to an equilibrium or desired point. Terminal constraints aim to ensure that the predicted trajectories remain within safe and stable bounds, *i.e* in regions of the state space in which stability can be guaranteed using an appropriate control strategy.

Both LQR and MPC methods can be designed to guarantee global stability of the system. LQR computes an optimal feedback gain $\mathbf{K}$ that ensures the poles of the closed-loop system, represented by $\mathbf{A} - \mathbf{BK}$, are located in stable regions of the complex plane (Benner 2006). MPC can guarantee stability under certain conditions by incorporating a terminal cost and terminal constraints into the optimisation problem (Kouvaritakis and Cannon 2016). Additionally, methods like Lyapunov stability analysis can be applied to the MPC framework to provide theoretical guarantees that the closed-loop system remains stable (Kouvaritakis and Cannon 2016).

Now that the key concepts of dynamic system control have been highlighted, the next section will focus on the consequences of actuator faults on performance and stability, in order to understand the problem addressed in this thesis.

2.1.4 Consequences of actuator faults on performance and stability

Actuator faults may arise in different forms, including loss of efficiency (Omerdic and Roberts 2004), additive faults where a bias is introduced into the actuator output (Ahmed Ali, Guermouche, and Langlois 2015), or changes in saturation limits of the control inputs (Zhu, Q. Liu, and Y. Yang 2008), and they may also occur as total faults when actuators become completely inoperative. Actuator faults can impact both the performance and stability of control systems. For controllers tuned using empirical methods, such as Ziegler-Nichols, actuator faults can lead to considerable performance degradation (Steven X Ding 2021). This is primarily because the tuned gains are optimised based on the system's behaviour without any faults. When an actuator is faulty, the dynamics of the system change, rendering the previously tuned gains ineffective. This can lead to undesired translations and rotations of the system, increasing the error between the desired and actual state (Omerdic and Roberts 2004).

In the case of model-based controllers, the impact of actuator faults on performance is similar. These controllers rely on mathematical models that are valid for systems operating without faults. When an actuator is faulty, the actual system dynamics are altered, and the mathematical model becomes inaccurate (L. Li 2016).

This can be represented as follows:

$$\dot{\mathbf{x}}(t) = \mathbf{A}'\mathbf{x}(t) + \mathbf{B}'\boldsymbol{\tau}(t) \tag{2.21}$$

$$\dot{\mathbf{x}}(t) = f'(\mathbf{x}(t), \boldsymbol{\tau}(t)) \tag{2.22}$$

Here, the matrices $\mathbf{A}$ and $\mathbf{B}$ transform to $\mathbf{A}'$ and $\mathbf{B}'$ to reflect the modified dynamics due to actuator fault, and the function f representing system behaviour becomes f' for nonlinear systems. As a result, the control strategy based on the original model may no longer lead to the desired performance, leading to suboptimal behaviour.

Regarding stability, actuator faults can compromise the stability of a system in several ways. The modification of the mathematical model due to faults can invalidate the methods used in the nominal case, such as the Lyapunov method or the use of terminal constraints (Ling-Ling and Yong-Duan 2010). Besides, the change in the model matrices of the system can shift the poles of the closed-loop system, potentially moving them into unstable regions of the complex plane (Ling-Ling and Yong-Duan 2010). Consequently, systems that had a stability guarantee can become unstable due to actuator faults, leading to undesired oscillations or divergence from the equilibrium state.

Actuator faults have critical implications for both performance and stability in control systems, necessitating the use of FTC methods. Next section will explore FTC strategies in the literature, to understand how they work and their limits.

2.2 Fault Tolerant Control strategies in the literature

This section focuses on existing FTC methods applicable to actuator faults in autonomous dynamic systems, under the assumptions that the system remains controllable despite such faults and that external disturbances are not considered, as stated in the introduction. Section 2.2.1 will focus on the differences between active and passive methods. Section 2.2.2 will look at the two main categories of adaptive FTC, which are direct adaptive FTC and indirect adaptive FTC, and will establish their limits.

2.2.1 Passive *vs* active Fault Tolerant Control

Many FTC techniques have emerged during the last forty years, and can be divided into several categories illustrated in Figure 2.4 (Amin and Hasan 2019). FTC can be passive or active, depending on whether or not the control system is redesigned during the mission.

In passive control (Batlle 2005), the controller's parameters remain constant over time, regardless of changes in the system dynamics. Passive controllers are designed to operate optimally under specific operating conditions, and their performance may degrade significantly when faced with perturbations or variations in the system. To address this challenge in passive control systems, the concept of robust control is employed. Robust

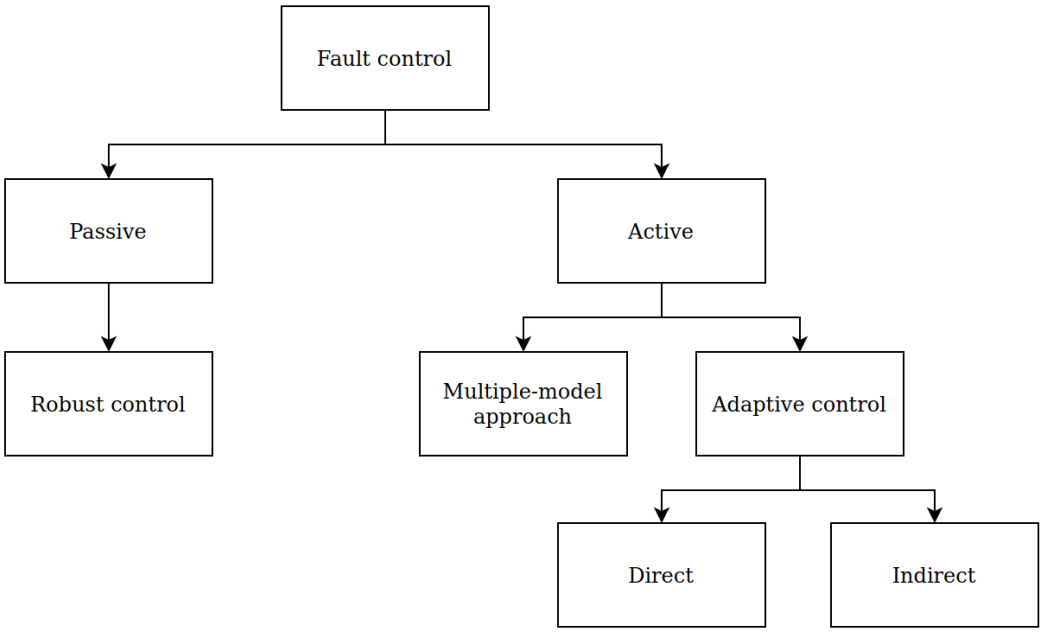


Figure 2.4: FTC methods classification

control involves designing controllers that can maintain desired performance levels even in the presence of uncertainties and variations in the system (Q. Li et al. 2008). These controllers are specifically engineered to provide stability and performance guarantees across a range of operating conditions, making them well-suited for applications where the system dynamics are not precisely known or may change over time (Dorato 1987; F. Lin 2007; Kwakernaak 1993). A passive robust controller can be used in our context to tolerate actuator faults (Benosman 2011).

Two main types of passive robust control methods for actuator faults can be found in the literature: H_∞ control (Trentelman et al. 2002) and sliding mode control (Utkin 2004). H_∞ control focuses on minimising the worst-case gain from disturbance on output, providing a robust framework that ensures stability and performance despite model uncertainties. It has been successfully applied in various fields, such as aerospace and automotive systems, showing that it can effectively manage uncertainties while maintaining system performance (Zhou and Doyle 1998). For example, in Davies et al. (2009), a passive robust controller was designed using H_∞ methods and effectively overcame faults in a system with high actuator redundancy. The insight from this work emphasises that robust control strategies, when combined with system redundancy, can provide reliable fault-tolerant control within a certain range of actuator faults.

In contrast, sliding mode control provides robustness by driving the system state onto a predefined sliding surface, ensuring that the system exhibits desired dynamics despite perturbations. In Saied et al. (2020), a passive robust controller was designed using high-order

super-twisting sliding mode techniques (Goel and Swarup 2015), and handled the faults of a redundant multirotor UAV without requiring any information about the fault. In Ahmed Ali, Guermouche, and Langlois (2015), a Super-Twisting based higher order sliding mode controller was proposed for the diesel engine air path, demonstrating robustness against parametric uncertainties as well as additive and loss-of-effectiveness actuator faults. Sliding mode control is particularly advantageous for systems with significant uncertainties, but it can introduce chattering phenomena (Quintana and Ciurana 2011), which may affect performance in practical implementations (Utkin 2004).

While passive robust controllers are effective in maintaining performance under certain conditions, they also have significant limitations, particularly regarding the generalisability of actuator faults. Their effectiveness diminishes for faults that lie outside the defined operational range (Davies et al. 2009). Passive robust controllers may struggle to adapt to unforeseen actuator faults, leading to performance degradation. Moreover, there is a trade-off between the number of faults that can be effectively managed and the overall performance of the system (Jiang and X. Yu 2012). Overparameterisation of the control strategy to accommodate various fault scenarios may result in diminished performance under normal operating conditions and for each fault situation. In Jiang and X. Yu (2012), it is shown that passive FTC method can never be optimal for all design scenarios. Consequently, active techniques are explored in the field of actuator fault recovery in autonomous dynamic systems, since they offer adaptability that passive methods cannot always guarantee.

There are two approaches of active techniques as shown in Figure 2.4: the multiple models approach (Hespanha et al. 2001) and the adaptive control approach (Karl J Åström and Wittenmark 2013). Regarding the multiple models method, a controller is designed for each fault situation. When a fault occurs, there is a switch between them to select the most appropriate one (Hespanha et al. 2001; Anderson et al. 2001; Narendra and Balakrishnan 1997; Maybeck and Stevens 1990). The disadvantage of this technique is that the design of many controllers is needed, one for each case of fault, and this can be challenging when there are various types of faults. This is why most of FTC approaches use adaptive control methods (Amin and Hasan 2019).

In adaptive control methods, a single model-based controller is used to adapt to the new situation by reconfiguring its parameters (Karl J Åström and Wittenmark 2013), and the next section discusses how such approaches have been applied in FTC.

2.2.2 Adaptive Fault Tolerant Control

Adaptive FTC strategies are an important class of methods designed to maintain performance despite faults. Adaptive FTC strategies can be classified into two main categories: direct and indirect approaches (P. Ioannou and Sun 1988). The following section introduces

the principles of both categories and highlights their advantages and limitations in the context of actuator faults.

2.2.2.1 Direct adaptive Fault Tolerant Control

Direct adaptive FTC methods (P. Ioannou and Sun 1988) continuously adjust their gains in real time based on the observed state deviation of the system, without updating the system model and without relying on fault information such as the type, location, or severity of the fault, in order to preserve performance despite the presence of faults. The main strategies that we can find in the literature to adjust the gains of a controller are explored below, and the main limitation of direct adaptive control for actuator faults is highlighted.

MIT rule

A common method is the Massachusetts Institute of Technology (MIT) rule (Mareels et al. 1987), originally proposed in the context of model reference adaptive control. It is based on gradient descent and aims to minimise the squared tracking error $J = \frac{1}{2}\boldsymbol{e}^\top(t)\boldsymbol{e}(t)$ by adjusting a parameter θ according to:

$$\dot{\theta} = -\gamma \cdot \boldsymbol{e}^\top(t) \cdot \frac{\partial \boldsymbol{e}(t)}{\partial \theta} \tag{2.23}$$

In the case of adaptive PID control, this parameter can represent $\boldsymbol{K_p}$, $\boldsymbol{K_i}$, or $\boldsymbol{K_d}$, and the partial derivative expresses how sensitive the error is to a change in that gain. In practice, simplified versions of this rule are often used, where the gains are directly updated using the error and its time characteristics:

$$\boldsymbol{K_p}(t+1) = \boldsymbol{K_p}(t) + \alpha \cdot \boldsymbol{e}(t) \tag{2.24}$$

$$\boldsymbol{K_i}(t+1) = \boldsymbol{K_i}(t) + \beta \cdot \int \boldsymbol{e}(t)dt \tag{2.25}$$

$$\boldsymbol{K_d}(t+1) = \boldsymbol{K_d}(t) + \gamma \cdot \frac{d\boldsymbol{e}(t)}{dt} \tag{2.26}$$

In these equations, α, β, and γ are tuning parameters that modify the adaptation rate. These values need to be carefully chosen: if they are too large, the controller may become unstable or overly sensitive to noise; if too small, the adaptation may be too slow to react effectively to changes or faults. The update can be performed continuously in real-time or at discrete time steps, depending on the implementation. Proper tuning of these adaptation rates is crucial

to ensure a balance between reactivity and stability. By continually observing the system's response and adapting these gains, this method aims to maintain the desired performance over time. This kind of method can be applied to systems with actuator faults. For example, the authors of Abd Latip et al. (2019) showed the implementation of a variable-gain PID control for robotic arms, highlighting how the adaptive mechanism can compensate for actuator faults without the need of any knowledge of the fault magnitude in advance. They also demonstrated that the stability of the proposed closed-loop system is guaranteed.

Similarly to the MIT rule, which adapts controller parameters based on the tracking error, gain-adaptive LQR strategies adjust the weighting matrix $\mathbf{Q}$ in response to measured deviations. For instance, in Lu et al. (2023), pitch and roll deviations measured by an Inertial Measurement Unit (IMU) are used to increase the corresponding weights in $\mathbf{Q}$, allowing the controller to adapt its behaviour in real time without requiring fault diagnosis.

Another method to adjust the gains of a controller is gain scheduling.

Gain scheduling

Gain scheduling involves adjusting control parameters based on the current operating conditions, using predefined gain schedules (Rugh and Shamma 2000). This method requires prior knowledge of the system's behaviour across various states and can be employed to handle non-linearities by switching between linear controllers valid in different operating points. In the context of fault-tolerant control, gain scheduling can be extended to include faulty conditions, with each fault scenario corresponding to a specific set of gains. However, this approach suffers from the same limitations as multiple-model strategies: it requires accurate modelling of all possible fault scenarios, and may fail to cope with unexpected or unmodelled faults. PID control with gain scheduling has been used to manage known actuator faults in aerospace applications, though its robustness remains limited outside predefined conditions (Rotondo 2017; Milhim, Youmin Zhang, and Rabbath 2010; Shin and Gregory 2007).

Model Reference Adaptive Control (MRAC)

MRAC (Nguyen and Nguyen 2018) is a model-based control strategy that dynamically adjusts its parameters to ensure that the output of a controlled system aligns with a predefined reference model that describes the desired dynamics of the system. The MRAC framework employs adaptation laws that modify control gains based on the tracking error. This mechanism aims to compensate for changes in system behaviour caused by faults, in order to maintain the desired performance of the system. The adaptation laws are designed

using stability theory, such as Lyapunov methods, to ensure that the updated gains drive the tracking error to zero and preserve closed-loop stability. MRAC can be used in the case of actuator faults to modify the control gains to counteract the impact of these faults. The authors of Arıcı and Kara (2020) demonstrated the effectiveness of MRAC on a tank system subject to actuator faults, showing how it can adaptively adjust controller parameters to ensure robust performance despite the faults. However, despite its advantages, MRAC also has limitations. One of the main challenges is its dependence on the accuracy of the reference model. If the model does not accurately represent the system dynamics, the adaptation process may lead to suboptimal performance (Dan Zhang and B. Wei 2017).

Main limitation of direct adaptive control

Even though direct adaptive control can be effective in certain configurations, its applicability becomes limited in complex actuator geometries (Tao et al. 2013). The main limitation lies in its lack of awareness of the actuator geometric configuration. Increasing the control gain, for example the proportional gain $\mathbf{K_p}$, in response to trajectory deviations may worsen the situation instead of correcting it. This is particularly challenging in overactuated systems where actuators are oriented differently, such as in omnidirectional mobile robots or underwater vehicles.

Consider, for example, a planar robot equipped with three omnidirectional wheels arranged in a triangular configuration, each capable of exerting force in a distinct direction. If one actuator suffers a partial or total fault, a direct adaptive controller that simply increases the gain on the lateral error may generate undesired torque around the centre of mass. This is because the contribution of each actuator to the force and torque vectors depends on its orientation. In such a configuration, increasing the effort on the remaining actuators does not necessarily compensate for the lost direction; instead, it can lead to increased rotation or lateral drift. The geometric coupling between the degrees of freedom implies that blindly increasing control action can introduce coupling effects that are not beneficial.

Now that we have seen that directly adjusting the gains of a controller can lead to suboptimal performances, or even degraded performance in some cases, methods that account for the system's mathematical model, including actuator geometry, are now considered. With these methods, the controller can use a new, more realistic model in case of faults to optimise its output. This type of methods is called indirect adaptive FTC (P. Ioannou and Sun 1988).

2.2.2.2 Indirect adaptive Fault Tolerant Control

Indirect adaptive FTC (P. Ioannou and Sun 1988) involves updating the system model based on information about the faults. One indirect adaptive approach is to incorporate

additional terms into the system model that specifically account for the effects of actuator faults. This method allows the controller to compensate for faults by explicitly modeling their impact on system dynamics. For instance, the authors of W. Zhang et al. (2021) and L. Yang et al. (2008) demonstrated how integrating a fault term into the state equations of a system can enhance actuator fault tolerance.

In linear systems, the state-space representation can be adjusted by modifying the matrices of the system model defined in 2.1.2 to reflect changes in system dynamics due to actuator faults. For example, the authors of A. Ghasemi, Abootalebi, and Mesbah (2014) explored the adaptation of the state-space matrices in a control system to overcome actuator faults, showing improved robustness. In the case of nonlinear systems, this may involve updating the evolution function defined in section 2.1.2 to incorporate the effects of faults, as in Zadeh and R. Ghasemi (2020), where nonlinear models were adjusted to reflect actuator faults.

The most common method in the literature to handle actuator faults involves control reallocation (Omerdic and Roberts 2004), which aims to modify the Control Allocation Matrix, or CAM, defined in section 2.1.1 and denoted $\mathbf{H}$. This is why the focus is now on control reallocation methods, which allows the faulty system to optimise its performances (Omerdic and Roberts 2004). Control reallocation methods assume that information about the fault is available. We begin by describing existing control reallocation methods that assume that the needed fault information is fully known. We then focus in more detail on methods that aim to obtain this fault information, including explicit fault diagnosis, and on their limits.

Control reallocation methods

The principle of control reallocation methods is to rely on system overactuation, also referred to as actuator redundancy, in order to maintain control despite actuator faults. This is achieved by modifying the CAM so that it reflects the current faulty actuator configuration.

In case of a total actuator fault, a simple method is to remove the corresponding column in the CAM (Choi and Kondo 2010; Caccia et al. 2001; Podder and Sarkar 2001). This method was experimentally validated on an underwater drone actuated by eight thrusters in Caccia et al. (2001) and on an AUV actuated by four horizontal thrusters and two vertical thrusters in Choi and Kondo (2010). This approach works if the system is largely overactuated — that is, if it has many more actuators than required to control all degrees of freedom — but may not be sufficient to compensate fully for the actuator faults as the geometric configuration of the actuators is not taken into account.

A more advanced control reallocation method consists in redistributing the control

forces among the actuators in order to compensate for faults, taking into account not only total faults but also partial faults, where actuators remain partially functional despite being affected. This approach takes the geometric configuration of the actuators into account: rather than discarding faulty actuators entirely, they adapt their influence within the control allocation process by modifying the CAM using actuator fault information. The redistribution approach varies according to the type of actuator fault considered, as will now be discussed.

One approach involves modelling the faults using a fault vector that represents the reduction in efficiency of the actuators (Omerdic and Roberts 2004). This vector adjusts the output of each actuator proportionally based on the fault severity. For instance, let the fault vector be represented as:

$$\mathbf{f} = [f_1, f_2, \ldots, f_n]$$

where each f_i indicates the performance level of the ith actuator. The thrust output T_i for each actuator can be expressed as:

$$T_i = f_i \cdot T_{i,\text{nominal}}, \quad \text{for } i = 1, 2, \ldots, n \tag{2.27}$$

where $T_{i,\text{nominal}}$ is the nominal output capacity of the ith actuator and n is the number of actuators.

To incorporate these performance adjustments into the control allocation process, a weight matrix $\mathbf{W}$ can be introduced, like in Omerdic and Roberts (2004) and L. Yang et al. (2008). This matrix is applied to the CAM to reflect the efficiencies of the actuators and can be represented as:

$$\mathbf{H}' = \mathbf{W}\mathbf{H} \tag{2.28}$$

where $\mathbf{H}$ is the original CAM. The weight matrix $\mathbf{W}$ is defined as a diagonal matrix with the elements of the fault vector $\mathbf{f}$:

$$\mathbf{W} = \text{diag}(f_1, f_2, \ldots, f_n) \tag{2.29}$$

Another type of actuator faults modelling that we can find in Zhu, Q. Liu, and Y. Yang (2008) involves changes in saturation of the control input. Let's consider partial faults with new saturation limitations s_i, expressed as in Equation 2.30, for each control input u_i, with $1 \leq i \leq r$.

$$-s_i \leq u_i \leq s_i, \ with \ 1 \leq i \leq r \tag{2.30}$$

In Zhu, Q. Liu, and Y. Yang (2008), the coefficients of the weight matrix $\mathbf{W}$ are

adapted to the fault according to Equation 2.31:

$$w_i = 1 + \alpha(\frac{1}{s_i} - 1),\ with\ 1 \leq i \leq r \tag{2.31}$$

with α a positive coefficient.

In the case of one or multiple partial faults, modifying the elements of **W** based on the fault vector allows to compensate for actuator faults (Omerdic and Roberts 2004). In the event of a total fault, the knowledge of the geometric configuration of the actuators should be incorporated into the method to fully compensate for the faults and avoid increasing the deviation.

Control reallocation methods rely on knowledge of fault information, which will now be examined in detail.

Acquisition of fault information

Control reallocation methods needs information about the fault that can be obtained by fault detection, fault isolation and fault identification as shown in Figure 2.5 (Nioras 2020). Fault isolation and fault identification can be grouped together as explicit fault diagnosis. The term *explicit* comes from the fact that the fault information is numerical values, such as efficiency losses, accessible by the user. This term is introduced in anticipation of the use of *implicit* fault diagnosis in the proposed approach detailed in the next chapter.

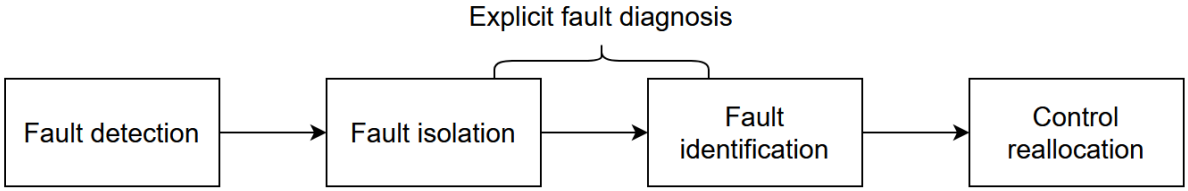


Figure 2.5: Indirect adaptive FTC strategy steps

Fault detection provides a binary indication of the system's operational state: whether it is functioning normally or experiencing faults. Fault detection can be carried out using methods based on a comparison between the observed and expected system's performance (Alessandri, M. Caccia, and Veruggio 1999). When the estimation of the system state is accurate, detecting the presence of a fault is generally not challenging, particularly in the case where no external disturbances are considered, as discrepancies between expected and observed behaviour can be attributed to fault occurrence.

Fault diagnosis is divided into isolation and identification. Fault isolation consists of determining the location of the fault within the system's components or subsystems. Fault

identification is about modelling the fault and estimating its severity (Marzat et al. 2012). The type and level of information required depend on both the nature of the actuator fault and the chosen reallocation method. In some cases, a basic indication of fault severity is sufficient—for example, selecting among a few discrete fault states such as "nominal", "degraded", or "failed" may allow for an appropriate control response. In other cases, a more accurate characterisation of the fault is necessary. This can involve estimating a continuous parameter, such as an efficiency coefficient representing the actuator's remaining performance (Omerdic and Roberts 2004). The required level of detail also depends on the degree of model fidelity needed to successfully compensate for the fault (Caccia et al. 2001).

Once the fault is diagnosed, the control reallocation can be performed, and its efficiency will depend on the accuracy of the fault diagnosis.

In many practical situations, no sensor is available to provide the controller with direct information about the fault (Abci et al. 2020), making analytical methods necessary for obtaining explicit fault diagnosis. Over the last fifty years, significant advances have been made in analytical methods for diagnosing actuator faults, and these have been the subject of several surveys (Willsky 1976; Katipamula and Brambley 2005; Gao, Cecati, and Steven X. Ding 2015; Ávila Okada et al. 2021). There are several ways of classifying explicit fault diagnosis methods, and in this thesis we choose to classify them into two categories: approaches based on prior exhaustive knowledge of faults — that is, methods that require predefined fault models or fault signatures — and online diagnosis methods that do not require a priori knowledge of faults.

Regarding the methods based on prior exhaustive knowledge, we can mention the observer-based methods, the signal-based methods and the data-driven methods (Gao, Cecati, and Steven X. Ding 2015).

- **Observer-based methods** continuously estimate the output of a process based on the system model and on its known parameters (Gao, Cecati, and Steven X. Ding 2015). The Kalman filter (Welch and G. Bishop 1994) is a widely used observer-based method for fault isolation in dynamic linear systems. An example of its application can be found in Alessandri, M. Caccia, and Veruggio (1999) where a bank of Extended Kalman Filters (EKFs) (Ribeiro 2004) was employed to detect and isolate faults in the actuators of an underwater drone. Each filter was designed to predict the system's output under both nominal and specific fault conditions, requiring prior knowledge of the fault scenarios to configure the observer accordingly. By comparing the output of the observer that closely matches the observed output, the results in Alessandri, M. Caccia, and Veruggio (1999) showed that it was possible to isolate and identify the presence of a fault and determine which actuator was affected.

However, it is important to note that this solution is applicable only to a limited

number of specific fault scenarios and quickly becomes limited when we also take into account multiple actuator faults simultaneously. This limitation arises from the increasing complexity and computational demand of the observer when multiple faults are considered (Daum 2021; Kaniewski 2020). As the number of simultaneous actuator faults increases, the bank of EKFs would have to grow exponentially to consider every possible fault combination, which would make the process impractical.

- In contrast, **signal-based methods** involve analysing output signals obtained from the system to identify abnormal behaviour or deviations from expected patterns (Gao, Cecati, and Steven X. Ding 2015). This approach relies on comparing measured signals with predefined thresholds or patterns to detect the presence of faults. By monitoring the signal characteristics deviations can be observed and used to isolate and identify faults. The most recent signal-based methods for fault diagnosis involve time-frequency domain methods.

Time-frequency domain methods combine time and frequency information to analyse the non-stationary behaviour of signals. Techniques such as wavelet analysis (Dengsheng Zhang 2019), Hilbert–Huang transform (Huang 2014), and Wigner–Ville distribution (Debnath 2002) offer insight into how signal frequencies evolve over time. Among them, wavelet-based approaches are particularly effective for fault detection, as they allow localisation of disturbances in both time and frequency domains. In Yin et al. (2019), the authors extract low-frequency trend components from control voltage and surge speed using wavelet approximation and least squares fitting. These are combined with detailed components to forward-predict the signals, from which thruster fault features are derived. The method enables accurate fault detection and isolation, with minimal prediction error, demonstrating the potential of wavelet-based diagnosis when prior knowledge is available.

Signal-based methods offer advantages including real-time capability. However, these methods require precise prior knowledge about faults in order to obtain an accurate diagnosis such as the degree of loss of efficiency of an actuator, and the accuracy of the diagnosis determines the efficiency of the fault-tolerant controller in the context of indirect adaptive FTC. Besides, the signal-based method focuses on extracting key features from output signals for fault diagnosis, but it tends to overlook the system's dynamic inputs. As a result, its diagnosis performance may suffer when faced with unknown input disturbances or unbalanced conditions (Gao, Cecati, and Steven X. Ding 2015).

- **Data-driven methods** build upon signal-based approaches by leveraging machine learning algorithms to identify and analyse patterns, trends, and anomalies in the data collected from various systems or processes. These methods rely on the availability of large volumes of data, which can include sensor readings, performance metrics, historical records, and other relevant information. In Junhao Li, Mou, and Dehua Zhang (2021), a robust fault-tolerant control method for quadrotors in a UAV was presented using a data-driven

approach. The authors began by formulating the multiplicative fault dynamic model of the quadrotor UAV using Newton-Euler principles (Ardema 2004). Then, they created adaptive radial basis functions with NNs (Buhmann 2000), employing sliding mode control to approximate the unknown function inherent in the system's fault model in real-time. The integration of an adaptive mechanism continuously estimated both the NNs weights and fault-related information during system operation. The results demonstrated the effectiveness of the method, achieving stable control and fault diagnosis with minimal error, even under varying fault conditions.

However, the main disadvantage of data-driven methods is that an extensive historical data of symptoms and associated diagnosis is required for accurate diagnosis, which can be challenging in the case of numerous potential faults.

We have just seen approaches that rely on prior knowledge of faults. An alternative approach known as *online fault diagnosis* (Jiashuai Li et al. 2024) requires no prior knowledge of faults, instead relying on real-time parameter estimation methods. These methods leverage real-time data to iteratively update system parameters, optimising them to closely match observed behaviour. By aligning estimated parameters with actual system responses, faults can be diagnosed as they occur, even without pre-existing fault information. Recursive least squares algorithms (Benesty et al. 2011; Shashoa, Emhemmed, and Yakhlef 2013) are popular techniques in this domain, as they iteratively adjusts parameters to minimise prediction errors and monitor system dynamics (Cimpoesu, Ciubotaru, and Stefanoiu 2013). More advanced approaches utilise embedded optimisation algorithms like PSO (Kennedy and Eberhart 1995) and Genetic Algorithms (Mirjalili and Mirjalili 2019), which mimic evolutionary processes to search for optimal parameter values (Singh, Tripathi, and Khanna 2024; Wen and Han 1995). For example, the authors of Jiashuai Li et al. (2024) developed an online fault diagnosis strategy for aerial robots experiencing actuator faults. Their approach combined sliding mode control with a fault estimation algorithm to compensate for faults in real-time. The results demonstrated improved fault-tolerant performance with and reduced estimation errors during actuator faults in aerial robots.

Online parameter estimation methods can be computationally intensive and time-consuming in real-time scenarios, especially in complex systems with many parameters (Bosch and Klauw 2020; Poli, Kennedy, and Blackwell 2007). Furthermore, the parameterisation of such algorithms is often complex and highly sensitive to initial conditions, which can significantly impact their convergence and performance (J. Zhao, W. Wang, and Sheng 2018).

Main limitations of indirect adaptive control

We have seen that the most promising FTC method in overcoming actuator faults is

the control reallocation approach. However, this method relies on the accuracy of explicit fault diagnosis. Existing fault diagnosis methods have several limitations: some rely on prior exhaustive knowledge of faults, which is challenging in the case of potentially numerous faults, and others require a lot of computational resources, which can lead to limitations in embedded systems in terms of available computing power, or if the system needs to react quickly. To avoid relying on explicit fault diagnosis, DRL-based FTC methods are being explored in the literature (L. Liu, Zhanshan Wang, and H. Zhang 2016; Hua, L. Li, and Steven X. Ding 2022). The use of DRL makes it possible to bypass the need for explicit diagnosis by relying solely on observation of the behaviour of the dynamic systems subjected to faults. Besides, once trained, a DRL model can be deployed with relatively low computational requirements (Yahmed et al. 2023). The following section focuses on DRL-based FTC methods in the literature.

2.3 Deep Reinforcement Learning-based Fault Tolerant Control strategies

The limitations of the above mentioned methods have led to the exploration of DRL-based FTC. The aim of this section is firstly to give an overview of the key concepts of DRL in section 2.3.1, in order to gain a better understanding of how this approach works. Then, in section 2.3.2, DRL-based FTC methods will be detailed and their limitations established, leading on to section 2.3.3, which explores existing methods combining DRL and model-based control for FTC.

2.3.1 Key concepts of Deep Reinforcement Learning

To understand how DRL-based controllers can bypass the need for explicit fault diagnosis, this subsection outlines the key theoretical foundations of DRL.

Markov Decision Processes and Reinforcement Learning framework

Let's first define the concept of Markov Decision Process (MDP) (Puterman 1990), which provides the basic framework of Reinforcement Learning (RL) (Sutton 1998). An MDP represents a mathematical framework used to describe an environment in which an agent makes decisions under uncertainty. A MDP is defined by a tuple (S, A, T, R), where S represents a finite set of states, A is a finite set of actions, $T(s_t, a_t, s_{t+1})$ defines the state transition probabilities given the current state and action, and $R(s_t, a_t)$ is the reward function. The agent receives a reward r_t for moving from state s_t to s_{t+1} by performing

action a_t. The action selection process is described through what is known as the policy $\pi : S \to A$. It serves as a mapping from the state s_t to actions a_t and can have either a deterministic or stochastic nature. The fundamental assumption in an MDP is the first order Markov property, which states that the future state s_{t+1} is dependent only upon the current state s_t and action a_t, and independent of all preceding states and actions.

In an MDP, an RL agent interacts with its environment to learn optimal decision-making strategies. The environment can be represented by different means, such as an actual physical space where the agent is evolving, or by something more abstract such as a set of rules.

The goal for the RL agent is to maximise the expected cumulative reward expressed in the objective function $J(\pi)$:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right] \tag{2.32}$$

where γ is the discount factor that balances future rewards. The agent aims to find the optimal policy π^* that maximises this expected return.

During the training phase, the agent learns to make decisions by interacting with its environment through a series of episodes. An episode refers to a complete sequence of interactions where the agent starts from an initial state and continues until a predefined termination condition is met, while a step denotes a single interaction within that episode, where the agent takes an action and receives feedback in the form of a reward and then reaches the next state. To find the optimal policy, the agent employs a trial-and-error approach during the training phase, balancing exploration of new actions with exploitation of learned actions to navigate through the environment and to learn from experience.

The exploration-exploitation trade-off is a critical aspect of RL, where an agent faces the challenge of balancing the exploration of unknown actions and the exploitation of known, high-reward actions. A common method to quantify this trade-off is the epsilon-greedy strategy (Sutton 1998), where with probability ϵ, the agent explores by choosing a random action, and with probability $1 - \epsilon$, it exploits by selecting the action with the highest estimated value (Sutton 1998). In practice, ϵ is decayed during training to progressively shift from exploration to exploitation.

Two widely recognised conditions to assure the optimality of the found policy involve *completeness* and *consistency*. Completeness ensures that all possible policies are considered during optimisation, while consistency ensures that the optimisation continually refines the policy towards optimality, avoiding oscillation between policy updates. Attaining both completeness and consistency is notably challenging in practice, largely due to the substantial computational effort required to explore vast state and action spaces and the challenges

posed by the unpredictable and variable nature of many environments (Sutton 1998).

The ideal approach is to train the RL model directly on the physical autonomous dynamic system, as it provides the most accurate representation without requiring a model. However, due to constraints such as time, cost, safety, or limited fault reproducibility, simulations are often used instead. These rely on models that approximate the real system as closely as possible to enable efficient and safe training.

Policy-based and Value-based learning

In RL, there are two main approaches for learning policies during the training phase: *policy-based methods* (Nachum et al. 2017), which directly update the policy to maximise cumulative reward, and *value-based methods* (Mnih et al. 2015), which deduce the best actions by first estimating the value of states or state-action pairs.

For *policy-based methods*, the focus is on directly updating the policy through a parameter denoted θ. The policy is then denoted $\pi_\theta(s)$. The parameter θ is updated iteratively during the training phase, which refers to the process of improving the performance of the agent, through techniques such as gradient descent (Ruder 2016; Amari 1993; Andrychowicz et al. 2016). In policy-based RL, the goal is to optimise θ to maximise the expected cumulative return by adjusting the policy's behaviour in response to observed rewards and state transitions.

In *value-based methods*, the goal is to approximate the expected cumulative return in each state by recursively considering future rewards and state transitions. For this purpose, each state is linked to a value known as the state-value $V(s)$. This value approximates the expected sum of future rewards when starting from state s, based on actions taken according to the policy. Essentially, the state-value measures the quality of a state. The value function $V(s)$ is updated through the Bellman update rule (Barron and Ishii 1989) (Eq. 2.33). Unlike policy-based methods, which optimise the policy directly, value-based methods focus on estimating the value of states or state-action pairs to guide decision-making.

$$V(s) = \max_a \left(\sum_{s'} P(s'|s,a) \left(r(s,a,s') + \gamma \mathbb{E}_\pi [V(s')|s'] \right) \right) \tag{2.33}$$

Here, $\max_a$ denotes the maximum over all possible actions. The equation expresses the value of a state s as the maximum expected sum of rewards when taking the best action a in state s and then following the optimal policy π.

An important distinction between reinforcement learning algorithms lies in whether they follow an *on-policy* or *off-policy* approach (Sutton 1998). On-policy methods learn from data generated by the current policy being optimised, which can limit sample efficiency.

In contrast, off-policy methods learn from data generated by a different policy stored in a replay buffer. This enables better sample reuse and thus higher sample efficiency but often requires stabilisation techniques due to the mismatch between behaviour and learned policy.

Deep Reinforcement Learning

Deep Reinforcement Learning (Mnih et al. 2015) is a machine learning approach that combines RL and Deep Neural Networks (DNNs) as shown in Figure 2.6. A DNN refers to a neural network architecture with multiple hidden layers, which enables it to learn complex patterns and representations from data. DRL leverages DNNs to approximate the policy and/or the value functions. The primary advantage of using NNs in DRL is that it significantly enhances adaptability in managing diverse and complex environments. This adaptability stems from the structure of NNs, which can effectively encode intricate patterns and relationships through their multi-layered architecture.

Each layer in a neural network consists of neurons connected by weights that process information through both linear transformations and non-linear activation functions like ReLU or Sigmoid (Y. Li 2017).

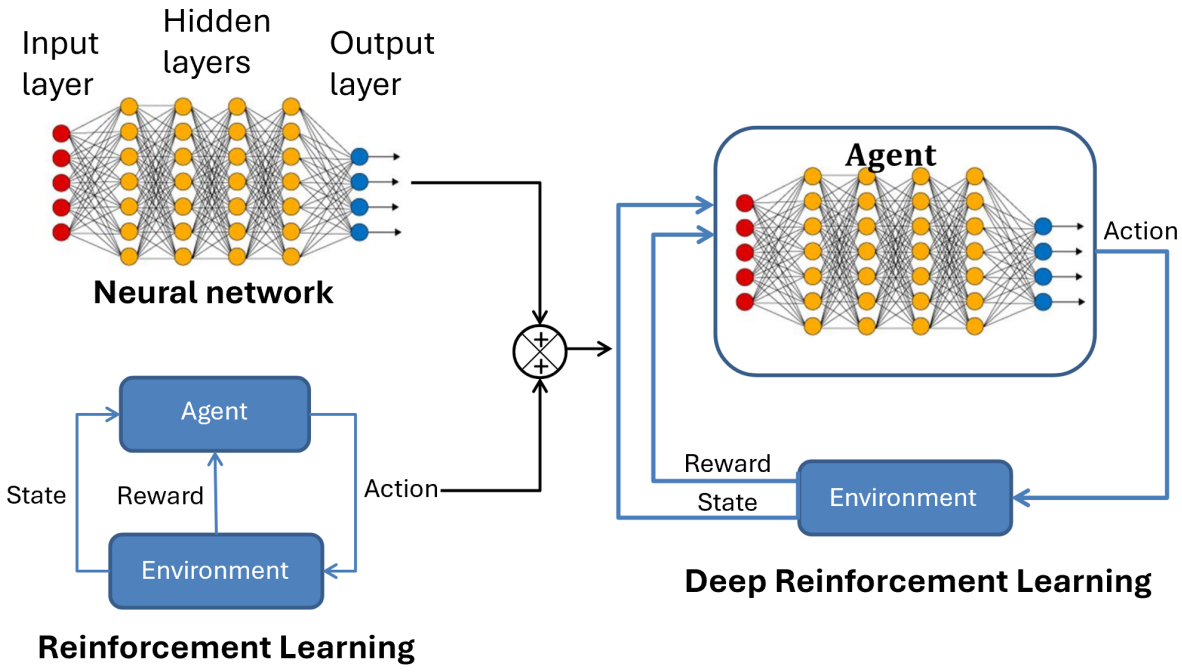


Figure 2.6: Deep Reinforcement Learning architecture general overview: a combination of Reinforcement Learning and Neural network.

For a given layer l, the output $\mathbf{h}_l$ is (François-Lavet et al. 2018):

$$\mathbf{h}_l = \sigma(\mathbf{P}_l \mathbf{h}_{l-1} + \mathbf{q}_l)$$

where $\mathbf{P}_l$ represents the weight matrix between layers $l-1$ and l, $\mathbf{q}_l$ is the bias vector, and σ is the activation function. Figure 2.7 illustrates a transformation between two successive layers.

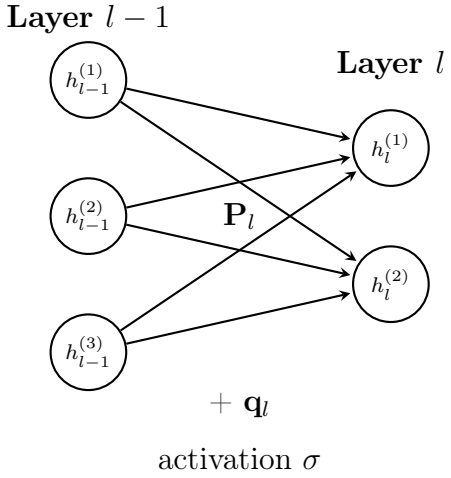


Figure 2.7: Example of a transformation from $\mathbf{h}_{l-1}$ (with 3 neurons) to $\mathbf{h}_l$ (with 2 neurons).

The combination of linear operations and non-linear activations enables the network to learn complex patterns. The ability to store large amounts of information comes from the number of parameters in $\mathbf{P}_l$ and $\mathbf{q}_l$. If each layer has n neurons, the weight matrix $\mathbf{P}_l$ contains n^2 parameters. When confronted with diverse situations during training, involving variable environmental changes and unpredictable events, the NN can adaptively adjusts its weights in order to respond as effectively as possible to the different situations encountered during training. This constant refinement of network parameters enables the DRL agent to improve its policy towards optimality even in complex environments subject to uncertainty (Kohl and Miikkulainen 2009).

Another attribute of DRL is its ability to generalise, enabling it to use learned experiences to make decisions in unseen states. Generalisation in both policy and value-based approaches is achieved via function approximation, commonly using NNs. Policy-based approaches employ function approximators to extrapolate learned policies to novel states. Similarly, value-based approaches utilise function approximators to estimate the values of unknown states based on those previously learned. However, the effectiveness of generalisation depends on the capacity and training quality of the function approximator, as well as on the degree of similarity between unvisited states and those encountered during training (Sutton 1998). Building upon these foundations, several DRL algorithms have been developed.

DRL algorithms

DRL algorithms refer to methods that implement RL with neural network function approximation. There are several DRL algorithms (Nagendra et al. 2017; Kim and Jung 2021). Among them, the most widely used in recent literature for controlling autonomous dynamic systems evolving in varying environments are Proximal Policy Optimisation (PPO) (Schulman et al. 2017), Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al. 2015), and Soft Actor-Critic (SAC) (Haarnoja et al. 2018). PPO, a stochastic on-policy algorithm, is commonly employed for its balance between performance and computational efficiency, providing stable policy updates, but with a low sample efficiency (Schulman et al. 2017). DDPG, a deterministic off-policy algorithm, is frequently used for continuous control tasks, although it is sensitive to noise and disturbances (Tan 2021).

SAC, a stochastic off-policy algorithm, is sample efficient and particularly effective in tasks requiring adaptability, leveraging entropy maximisation to improve exploration and performance in complex varying environments (Haarnoja et al. 2018). These properties make it particularly relevant in the context of actuator faults, where the controller must adapt to varying system conditions. The SAC algorithm is now examined in more detail to better understand its advantages.

The SAC algorithm employs an actor-critic architecture, combining both policy-based and value-based learning principles. It uses one actor network to approximate the policy and two critic networks to estimate the state-action value functions, denoted as $Q(s_t, a_t)$. The state-action value function extends the concept of the value function $V(s_t)$ by assessing the expected return of taking a specific action a_t in a given state s_t and then following the policy $\pi_\theta(a_t|s_t)$. Each critic maintains a target network for stable training through soft updates.

The SAC algorithm adopts a stochastic policy approach and incorporates principles of maximum entropy reinforcement learning, aiming to maximise both the expected reward and the entropy of the policy in order to encourage exploration. Although potentially noisier, the stochastic nature of the policy improves robustness and exploration (Haarnoja et al. 2018), which are desirable properties in the context of actuator fault recovery. The actor network is trained to maximise the following objective function:

$$J(\theta) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim D} \left[\alpha \log(\pi_\theta(a_t|s_t)) - Q_\phi(s_t, a_t) \right] \tag{2.34}$$

where θ and ϕ are the parameters of the actor and critic networks respectively, D is the replay buffer, and α is the temperature parameter that controls the trade-off between exploration

and exploitation. The critic networks are trained to minimise the following loss function:

$$J_Q(\phi) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim D}\Bigg[\Big(Q_\phi(s_t,a_t) - \Big(r_t + \gamma\,\mathbb{E}_{a_{t+1}\sim\pi_\theta}\Big[Q_{\bar{\phi}}(s_{t+1},a_{t+1}) - \alpha\log(\pi_\theta(a_{t+1}|s_{t+1}))\Big]\Big)\Big)^2\Bigg] \tag{2.35}$$

where $\bar{\phi}$ denotes the parameter of the target critic network and r_t is the immediate reward. The pseudocode of the SAC algorithm is presented in Algorithm 1.

Algorithm 1 Soft Actor-Critic

```
Initialise actor network π_θ, critic networks Q_{φ1}, Q_{φ2}, and target critics φ̄_1, φ̄_2
Initialise replay buffer D
for each interaction step with the environment do
    Sample action a_t ~ π_θ(a_t|s_t)
    Execute a_t, observe reward r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample mini-batch from D
    Update critics by minimising J_Q(φ_1) and J_Q(φ_2)
    Update actor by maximising J(θ)
    Update target critics: φ̄_i ← τφ_i + (1 − τ)φ̄_i
end for
```

SAC is an off-policy algorithm which allows efficient reuse of past experiences and improves sample efficiency compared to on-policy algorithms like PPO which must collect new data from interactions with the environment after each policy update. Besides, SAC offers a strong balance between exploration and exploitation through entropy maximisation, improving adaptability in dynamic environments. SAC also presents certain limitations: the algorithm relies on multiple neural networks which increases the overall computational cost. Moreover, the stochastic nature of the policy, while beneficial for exploration and robustness, can introduce high variance in performance (Haarnoja et al. 2018).

Now that the key concepts of DRL have been discussed, the next section will highlight its use in the FTC area.

2.3.2 Deep Reinforcement Learning-based Fault Tolerant Control methods

In the context of FTC, multiple methods of end-to-end DRL (Chopra and Roy 2020) exist in the literature (Farivar and Ahmadabadi 2015; Zhanshan Wang et al. 2015; L. Liu, Zhanshan Wang, and H. Zhang 2016; Hua, L. Li, and Steven X. Ding 2022; Ahmed, Quinones-Grueiro, and Biswas 2022). End-to-end DRL is a purely machine learning approach

where a DRL model learns directly actuator inputs, without relying on any standard control framework (Chopra and Roy 2020) as shown in Figure 2.8.

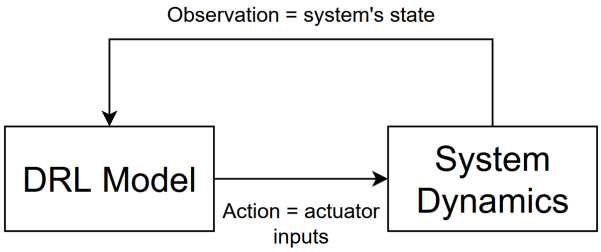


Figure 2.8: End-to-end DRL for fault tolerance in autonomous dynamic systems.

Some existing end-to-end DRL-based methods demonstrate the ability to handle faults directly from interaction with the environment, without relying on an explicit fault diagnosis.

In Qu et al. (2023), a DRL-based control scheme without any need for explicit fault diagnosis for a system with model uncertainties was presented. The proposed approach used a DRL method that combines a Long Short-Term Memory (LSTM) network (Y. Yu et al. 2019), a type of recurrent neural network designed to capture temporal dependencies in sequential data, with the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm (Dankwa and Zheng 2019), an actor-critic method for environments with a continuous action-space. The control inputs generated by NNs included surge forces and yaw torques. The results showed that the system effectively minimised deviation from the desired path under varying conditions. In a similar way, in Fei et al. (2020), a fault-tolerant control strategy employing DRL techniques was described and experimentally validated using a real quadcopter UAV. The agent was trained on scenarios both without faults and with randomly applied system faults. The trained agent could subsequently handle the considered faults without the need for diagnosis.

These end-to-end methods take advantage of the adaptability of DRL for FTC and some of them show that it can be used to overcome actuator faults without the need for explicit fault diagnosis. The conclusion we draw from this is that this ability is achieved by allowing the DRL agent to learn control policies based solely on observations of the system's behaviour in faulty conditions, learning an implicit diagnosis of the fault. Through continuous interaction with the system, the DRL agent identifies patterns in the system dynamics that correspond to actuator faults. It then adapts the control strategies accordingly, allowing to maintain performance. However, the stability of an end-to-end DRL controller is challenging to guarantee as explained in section 2.3.1. The outputs of this type of controller are mainly unpredictable (Dulac-Arnold, Mankowitz, and Hester 2019). This is due to the fact that DNNs are composed of multiple layers of nonlinear functions. These nonlinearities make the input–output mapping highly complex and difficult to characterise analytically,

thereby preventing the use of classical control theory tools to formally establish stability guarantees. In addition, end-to-end DRL methods for the control of autonomous dynamic systems require a wide amount of historical data (Doukhi and D.-J. Lee 2021).

In order to bring stability to an end-to-end controller and reduce the amount of data required for learning, methods in the literature explore the combination of DRL and model-based methods, the definition of which is given in section 2.1.3. Next section will focus on the combination of DRL and model-based control for FTC.

2.3.3 Combination of Deep Reinforcement Learning and model-based control for Fault Tolerant Control

In the literature, the papers using combinations between DRL and model-based controller for adaptability and stability purposes mainly focus on system uncertainties (Rizvi and Z. Lin 2019; Yaghmaie, Gustafsson, and Ljung 2022).

The effectiveness of combining DRL and model-based control has been explored and proven: some papers use DRL to learn the gains of a PID controller (Shuprajhaa, Sujit, and Srinivasan 2022; X.-S. Wang, Cheng, and S. Wei 2007; Sedighizadeh and Rezazadeh 2008; Chaffre, Le Chenadec, et al. 2021). For example, in Chaffre, Le Chenadec, et al. (2021), the SAC algorithm was used to adapt the gains of a PID controller for an AUV, and the results were promising, demonstrating effectiveness in both simulation and real-world applications.

Other works use DRL to optimise an LQR controller (Rizvi and Z. Lin 2019; Yaghmaie, Gustafsson, and Ljung 2022; Bradtke 1992; Görges 2019). The conclusions drawn from these papers are that DRL learning is more ressource-efficient when it comes to learning controller gains rather than direct actuator inputs because it allows the DRL agent to operate on a lower-dimensional action space, and that the adaptive nature of DRL allows model-based controllers to be used in contexts with uncertainty. The stability of this type of hybrid controller can be proven by placing constraints on the adapted controller parameters. This can be achieved by adding upper and lower limits, also called stability margins, on the adapted gains that the DRL modifies, ensuring that the adaptive controller remains within a stable operating region (Umar et al. 2024).

Another method involves using Lyapunov-based stability criteria described in section 2.1.3, where the controller is designed so that any adaptations made by the DRL agent satisfy a Lyapunov function, ensuring that the overall system's energy decreases over time (Rizvi and Z. Lin 2023).

Even if existing hybrid methods mainly focus on external disturbances, we can find a few papers using this combination for actuator faults.

The approach proposed in Bhan, Quinones-Grueiro, and Biswas (2021) combined model-based control methods with DRL for FTC. They used a PID controller and introduced

DRL to adaptively tune the PID parameters when faults occur. Their method focused on estimating fault-related parameters, using techniques such as the Unscented Kalman Filter to detect changes in system dynamics due to faults. These estimated parameters were fed into the DRL agent, which then adjusted the PID controller parameters to mitigate the impact of the faults, ensuring the system could continue functioning in a degraded mode. The authors demonstrated their approach on an octocopter trajectory-tracking task with actuator faults and showed that the system could recover its performance by dynamically updating the PID parameters when fault-related parameter deviations were detected. However, the method requires an explicit fault diagnosis and the article did not provide a formal proof of stability for the overall system. In addition, the paper used a direct adaptative method the limitations of which we saw in section 2.2.2.1.

In contrast, in Ma and Peng (2021), instead of combining RL and PID control, the authors presented a FTC scheme using RL and an LQR-like controller. The control strategy introduced the integral of the tracking error as a new state to create an augmented system for control design. The RL method was employed to compute the optimal feedback control gains online, adapting to actuator faults and system uncertainties without requiring prior knowledge of the system dynamics.

Additionally, an observer-like reference model was incorporated to handle external disturbances and uncertainties adaptively. The authors demonstrated that, with appropriate parameter tuning, the proposed controller ensured uniform boundedness of the closed-loop system and that the tracking error asymptotically approached zero. Numerical simulations confirmed the effectiveness of this approach on an aircraft, showing that it could accommodate faults and maintain performance. However, in this method, the RL component directly modified the LQR controller's gain in real-time, making it a direct adaptive control method like the previously mentioned method.

Now that the DRL-based FTC methods have been detailed in this subsection, the next section will summarise the main limits of existing FTC methods and highlights the gap in the literature.

2.4 Limits of existing Fault Tolerant Control methods - Gap analysis

In this chapter, we have explored the main types of FTC control strategies that we can find in the literature: passive robust control, direct adaptive FTC, indirect adaptive FTC and DRL-based FTC. Figure 2.9 summarises the limits of existing FTC methods, based on the literature review and the key characteristics of fault-tolerant controller defined in the introduction.

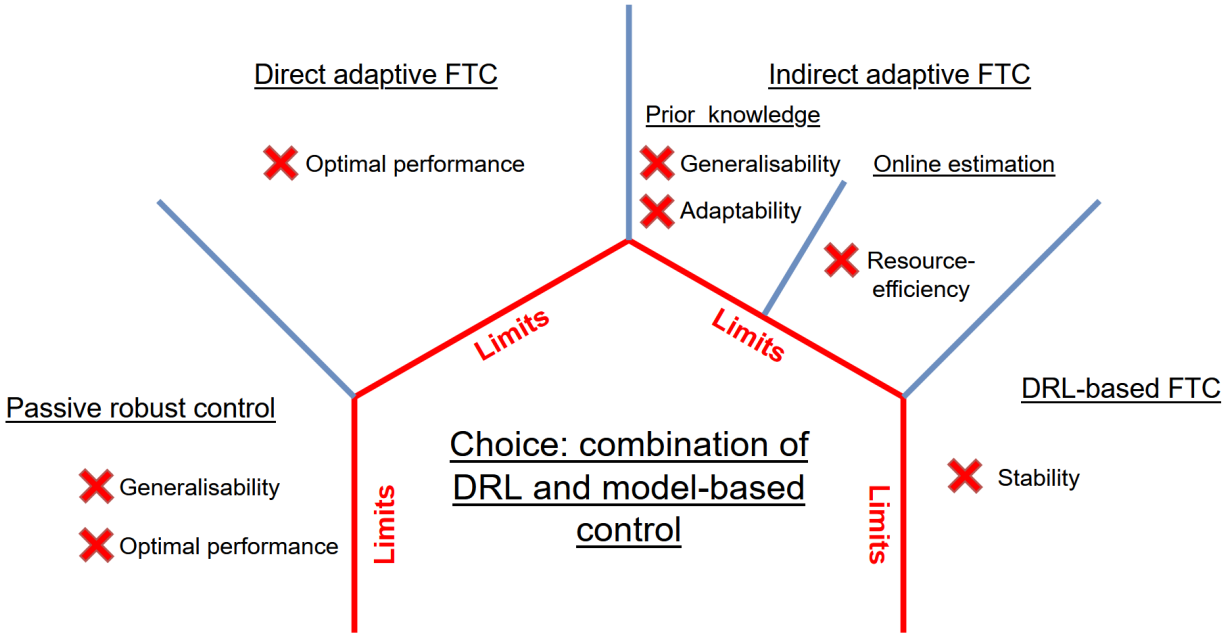


Figure 2.9: Limits of existing methods, based on FTC key characteristics and literature review.

Passive robust control, while effective within defined operational ranges, struggles with unforeseen actuator faults, as previously mentioned. Its inability to generalise beyond a fixed set of conditions leads to significant performance degradation outside the anticipated fault scenarios. Moreover, attempts to overparameterise the controller to accommodate more fault cases can reduce its performance under normal operating conditions. Passive FTC methods are not optimal for all design scenarios, often forcing a trade-off between robustness and system performance. For this reason, adaptive techniques are generally preferred for actuator fault recovery due to their superior performance.

Direct adaptive FTC, on the other hand, offers a simple and immediate response by adjusting control gains directly. However, this approach lacks precision, particularly when there is no accurate information about the actuator faults. Increasing gains without fault information can exacerbate the problem, potentially causing actuator saturation and worsening the system's performance due to the geometric configuration of the actuators. This lack of nuanced control adjustment can lead to suboptimal performance.

Indirect adaptive FTC methods propose a more nuanced control adjustment by using control reallocation to redistribute the control forces among the remaining actuators, but are heavily dependent on the accuracy of explicit fault diagnosis. Existing fault diagnosis techniques have limitations, often requiring exhaustive prior knowledge of potential faults or significant computational resources. This can be challenging for embedded systems with limited computing power, or in scenarios where the system must react rapidly to faults.

To address these issues, researchers are increasingly exploring end-to-end DRL-based FTC methods, which have the potential to bypass explicit fault diagnosis as discussed earlier. This is achieved by allowing the DRL agent to learn control policies based solely on observations of the system's behaviour in faulty conditions, thus learning an implicit diagnosis of the fault. However, ensuring the stability of an end-to-end DRL controller remains a significant challenge: this type of controller produces outputs that are difficult to predict, making it more complex to guarantee consistent performance and stability across various operational conditions.

As illustrated in Figure 2.9, the limitations of the existing FTC methods discussed in this chapter motivated the development of a strategy that combines DRL with model-based control, aiming to leverage the stability and adaptability offered by such a controller. The potential of combining the advantages of a purely DRL-based control and of a model-based control, and the limited exploration of this approach in the literature, led us to form our research statement in this thesis: the combination of DRL and model-based control improves the overall performance of an autonomous dynamic system subject to actuator faults compared to existing methods.

The effectiveness of this type of hybrid controller has been demonstrated on systems with uncertainty, as stated previously, but these existing controllers are of the direct adaptive type, which is not ideal to overcome actuator faults as they do not take into account the configuration of the actuators. We have seen that this type of method is limited in the case of complex systems. The major gap in the literature concerns the use of a combination of DRL and indirect adaptive control to overcome actuator faults.

This is why the proposed approach in this thesis, supporting the research statement, adopts an indirect adaptive method, as it allows the controller to rely on a more accurate system model. The core concept of the proposed approach is to perform control reallocation using DRL in a model-based control framework, allowing to bypass the need for explicit fault diagnosis which presents major challenges. This proposed approach will be presented in the following chapter.

Chapter 3

Hybrid approach for actuator fault recovery: combining DRL and model-based control

This chapter introduces the approach proposed in this thesis, which consists in **performing control reallocation without relying on explicit fault diagnosis, by integrating DRL within a model-based control framework**. This aims to support the research statement that combining DRL with model-based control enhances the overall performance of an autonomous dynamic system subject to actuator faults, when compared with existing methods. The framework of the proposed approach is based on the assumptions stated in the introduction, namely the absence of external disturbances and the system remaining controllable despite the fault situation. Based on the gap identified in the literature, the proposed hybrid strategy targets situations where direct adaptive methods are not sufficient — where the geometry of the actuators prevents effective compensation through simple gain adjustment — and where explicit fault diagnosis is challenging to obtain, either due to the high number of potential faults or the complexity of their effects on the system. As mentioned in section 2.3.2, the main motivation for using DRL is its ability to implicitly recognise and adapt to faults without requiring explicit diagnosis. This ability is used in the proposed approach to reallocate control efforts in real time to mitigate the effects of faults, solely based on system behaviour. This chapter is divided into three sections. Section 3.1 will present the core concept of the proposed hybrid controller, as well as its application framework and general structure. Section 3.2 will focus on the use of DRL to perform control reallocation, and section 3.3 will highlight how to integrate the DRL method in the model-based controller with the objective of ensuring stability.

3.1 Core concept of the proposed approach

The general overview of the proposed approach and the advantages of using DRL for control reallocation in a model-based control framework in case of actuator faults will be detailed in the section 3.1.1. Then, section 3.1.2 will focus on the type of system suitable for applying this method and on the type of faults considered.

3.1.1 General overview

This section begins by describing how the control strategy transitions from nominal model-based control to an adaptive DRL-based approach in the case of a fault situation, and then presents the general structure of the proposed control strategy.

Transition from nominal to adaptive control

In the proposed approach, DRL is employed only when an abnormal behaviour is detected in the system. Unlike explicit fault diagnosis, this detection — which consists in determining whether a fault is present without providing further details — is facilitated by the assumption that no other disturbances affect the system. This abnormal behaviour is detected by analysing the difference between the desired state and the current state of the system. When this difference exceeds a predefined threshold (defined according to the desired deviation tolerance), a fault is detected and a DRL algorithm is employed to adjust the CAM as shown in Figure 3.1:

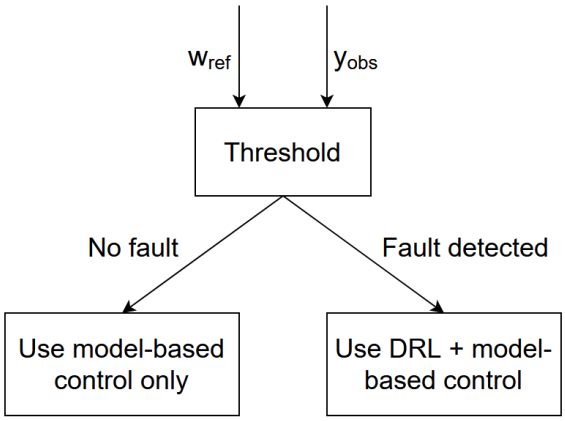


Figure 3.1: Fault detection scheme.

where $\mathbf{w_{ref}}$ and $\mathbf{y_{obs}}$ are the reference vector and observation vector defined in section 2.1.1. This detection relies on a simple binary indicator that flags deviations from expected behaviour, which is different from performing a detailed fault diagnosis. This choice is driven

by the need to maintain optimal control under normal operating conditions. Model-based controllers are well-suited for nominal conditions, as they can provide global stability and efficiency without requiring adjustments. By limiting the use of DRL to scenarios where faults or anomalies are present, we can preserve the benefits of these model-based controllers, ensuring that the system operates optimally when no changes are necessary. Using DRL may introduce unpredictability into the controller, which should be avoided to preserve the reliability of the system during nominal operation.

Furthermore, this selective use of DRL reduces the amount of learning required for the DRL model, as it only needs to focus on adjusting the system during actuator fault conditions. This simplifies the training process, enabling faster convergence and better performance in abnormal situations. This limited scope ensures that the DRL model does not have to learn or replicate the behaviour of the model-based controller during normal operations, which would be redundant but could also increase the complexity of the learning process.

Overview of the hybrid controller structure

Having detailed in which situation the DRL method will be used, the proposed controller structure is shown in Figure 3.2.

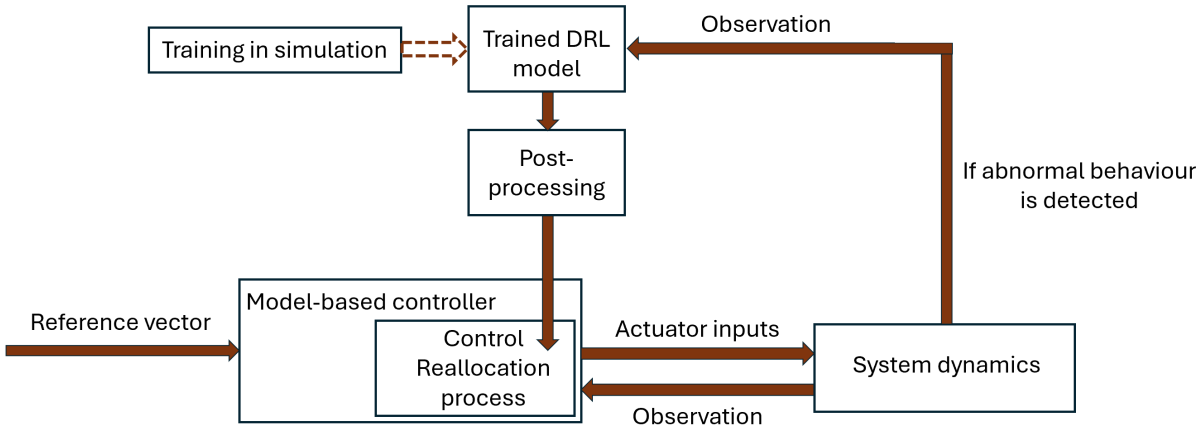


Figure 3.2: Hybrid controller structure overview.

The goal of the proposed approach is to perform control reallocation without the need for explicit fault diagnosis, using a pre-trained DRL model as shown in Figure 3.2. In this context, "pre-trained" means that the DRL model is trained offline, and then deployed during operation without further learning. To achieve control reallocation, the input to the DRL model receives the necessary information to implicitly identify patterns of actuator faults

based on the system's behaviour, in order to take actions accordingly: recognising actuator faults from system behaviour requires an observation vector that includes the current state of the system, as we can see in Figure 3.2. Furthermore, it can be observed in Figure 3.2 that the controller is not divided into two separate blocks with a control law and an allocation law, as is commonly done as shown in Figure 2.2. In the proposed approach, the controller directly outputs the actuator inputs, and the DRL model adjusts the CAM to perform control reallocation, which is integrated into the control law equations of a model-based controller. In addition, a post-processing stage is applied to the DRL model output in order to smooth variations in the control signals.

The aim of the approach is to improve the overall performance of an autonomous dynamic system in case of actuator faults. The overall performance of a system depends on the key characteristics of FTC strategies detailed in the introduction of this thesis, which are the optimal recovery performance, the stability, the adaptability, the generalisability and the resource-efficiency of the fault-tolerant controller. The proposed combination addresses these key characteristics and provides the underlying motivation for the approach.

In terms of optimal recovery performance, DRL methods aim to gradually converge towards an ideal strategy — in our case, for force reallocation. Regarding adaptability, DRL can potentially handle a wide range of training situations by leveraging the capacity of NNs to approximate complex, non-linear functions. Third, concerning generalisability, DRL can manage previously unseen faults by applying knowledge from similar situations encountered during training. This capacity for generalisation arises from the use of NNs, which act as function approximators as explained in section 2.3.1.

To complement these learning-based strengths, this work adopts a model-based control framework to provide a structure in which the DRL policy operates. The model-based component incorporates known system dynamics, serving as a foundation for stability, while the DRL component enhances robustness to faults by adjusting the force distribution in real time. The addition of a model-based component provides a structure that can help constrain the DRL policy and reduce the risk of instability (Osinenko, Dobriborsci, and Aumer 2022).

Finally, once training is completed, DRL controllers can be deployed with low computational cost, especially when lightweight models are used. This supports the resource-efficiency requirement of real-time applications on embedded systems, and may be more practical than online estimation techniques requiring continuous updates, as discussed in section 2.2.2.1.

Now that we have seen the general structure of the proposed approach and detailed why using DRL to perform control reallocation theoretically enables the overall performance of the system to be improved compared to existing methods, the next section will justify the choices of the working framework of our proposed approach, including the considered systems and the considered fault model.

3.1.2 Methodological framework

The proposed approach is carried out within the framework of a system that can be controlled using a model-based controller, whose structure is presented in Figure 3.3.

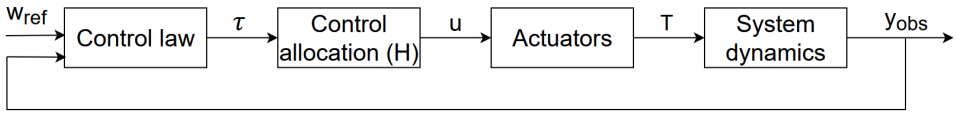


Figure 3.3: Control law and control allocation scheme of the considered system.

As previously defined, $\mathbf{w}_{\text{ref}}$ denotes the reference vector, $\boldsymbol{\tau} \in \mathbb{R}^n$ is the control input vector composed of n force components, $\mathbf{u} \in \mathbb{R}^r$ represents the actuator input vector with r actuators, $\mathbf{T} \in \mathbb{R}^r$ is the actuator output vector, and $\mathbf{y}_{\text{obs}}$ denotes the observation vector, all expressed in the earth-fixed coordinate frame.

As seen in section 2.1.2, there are several ways of representing the state equations of a system, depending on whether it is linear, or non-linear. In the case of the proposed approach, the objective is to adjust the CAM denoted $\mathbf{H}$, defined in Equation 2.1, to perform control reallocation while maintaining the stability of the closed-loop. To allow the explicit integration of the matrix $\mathbf{H}$ into the system equation used by the controller, and to enable a direct approach for ensuring stability, as detailed in the following sections, we restrict our analysis to linear or linearised systems around an equilibrium point. In this context, the dynamics of the considered system can be written as:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\boldsymbol{\tau} \tag{3.1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the state vector, $\boldsymbol{\tau} \in \mathbb{R}^m$ is the control input vector (i.e. the forces and torques applied to the system), $\mathbf{A} \in \mathbb{R}^{n \times n}$ is the state matrix describing the linearised dynamics of the system, and $\mathbf{B} \in \mathbb{R}^{n \times m}$ is the input matrix that maps control inputs to their effect on the state.

We saw in section 2.2.2.2 that several approaches to model actuator faults exist in the literature. In the proposed approach, actuator faults are modelled using a time-invariant fault vector that proportionally adjusts the output of each actuator according to its efficiency. This method offers a continuous and realistic representation of partial faults, which is well suited for learning-based control strategies. Thus, we use a fault vector $\mathbf{f}$, which represents the efficiency of each actuator under fault conditions. This vector adjusts the output of each actuator proportionally based on the severity of the fault. Let $\mathbf{f}$ be defined as:

$$\mathbf{f} = [f_1, f_2, \ldots, f_n]$$

where f_i represents the efficiency of the ith actuator, with $0 \leq f_i \leq 1$, where $f_i = 1$ indicates

full efficiency and $f_i = 0$ corresponds to a total fault. In the case of the proposed approach, it will be necessary to simulate actuator faults, both to train the DRL model and to be able to evaluate the method, in simulation and real-world conditions. It is challenging to act on the actuator output without actually damaging the actuators, this is why faults will be simulated by acting on the actuator input vector $\mathbf{u}$ defined in section 2.1.1. For simplicity, it is assumed that there is a linear relationship between the input and output of the actuators. Consequently, the fault can be modelled and simulated by directly applying the vector $\mathbf{f}$ to the actuator input $\mathbf{u}$. We use the Hadamard product (Horn 1990), an element-wise multiplication, to model the scaling of actuator inputs due to faults. Each actuator input under faulty conditions u'_i is scaled by its efficiency factor f_i and can be expressed as:

$$u'_i(t) = f_i \circ u_i(t), \quad \text{for } i = 1, 2, \ldots, n$$

where u_i is the input of the ith actuator in fault-free conditions, and n is the total number of actuators. The actuator faults we are considering in this thesis can occur on several actuators simultaneously, but the faulty system must remain controllable, in accordance with the assumption of the introduction chapter.

The actuator saturation constraints are taken into account and a truncation is applied to ensure the feasibility of the control inputs as described in Equations 2.4 and 2.5.

This section has presented the core concept, motivations and framework of the proposed approach. The following sections present the methodology of the proposed approach by describing the main components of the diagram in Figure 3.2. The next section specifically focuses on the DRL-based control reallocation process for actuator fault recovery.

3.2 Deep Reinforcement Learning-based control reallocation

In this section, we will start by exploring the reallocation method using DRL, focusing on how this approach is employed to manage actuator faults effectively. The training process of the DRL model will also be presented, highlighting how it learns to reallocate control inputs in the presence of faults.

3.2.1 Introduction of the weight matrix

In the proposed control allocation strategy, the CAM $\mathbf{H}$ relates the actuator inputs $\mathbf{u}$ to the control input $\boldsymbol{\tau}$ through the standard allocation equation $\boldsymbol{\tau}(t) = \mathbf{H}\mathbf{u}(t)$.

To handle actuator faults, a weight matrix $\mathbf{W}$ is introduced to redistribute the control input forces among the system actuators. The weight matrix $\mathbf{W}$ is defined such that, in the

absence of faults, it contains only ones:

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 1 & 1 & \cdots & 1 \\ 1 & 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \end{bmatrix}$$

We use the Hadamard product for the relation between $\mathbf{H}$ and $\mathbf{W}$ and to model the scaling of actuator inputs due to faults. The control allocation equation becomes:

$$\boldsymbol{\tau}'(t) = (\mathbf{W}(t) \circ \mathbf{H}) \cdot (\mathbf{f} \circ \mathbf{u}(t)) \tag{3.2}$$

where the Hadamard product $\mathbf{W}(t) \circ \mathbf{H}$ can be expressed as follows:

$$\mathbf{W}(t) \circ \mathbf{H} = \begin{bmatrix} W_{11}(t)H_{11} & W_{12}(t)H_{12} & W_{13}(t)H_{13} & \cdots & W_{1r}(t)H_{1r} \\ W_{21}(t)H_{21} & W_{22}(t)H_{22} & W_{23}(t)H_{23} & \cdots & W_{2r}(t)H_{2r} \\ W_{31}(t)H_{31} & W_{32}(t)H_{32} & W_{33}(t)H_{33} & \cdots & W_{3r}(t)H_{13r} \\ \vdots & \vdots & \vdots & \ddots & \\ W_{n1}(t)H_{n1} & W_{n2}(t)H_{n2} & W_{n3}(t)H_{n3} & \cdots & W_{nr}(t)H_{nr} \end{bmatrix} \tag{3.3}$$

where n is the number of actuators and r the number of desired forces in the control input.

Even though the expression for $\mathbf{W}$ cannot be made explicit in the general case because it depends on the system and the geometric configuration of the actuators, it is possible to show that there exists a $\mathbf{W}$ matrix which can maintain control over the forces which form the reference vector $\mathbf{w_{ref}}$.

Since the system remains controllable in the event of actuator faults, this means that for a given $\boldsymbol{\tau}$, a matrix $\mathbf{H}'$ exists such that:

$$\boldsymbol{\tau}(t) = \mathbf{H}' \cdot (\mathbf{f} \circ \mathbf{u}(t)) \tag{3.4}$$

for all integer i from 1 to n and integer j from 1 to r, and if H_{ij} is not equal to zero, we can express the values of the weight matrix by:

$$W_{ij} = \frac{H'_{ij}}{H_{ij}} \tag{3.5}$$

In the case where H_{ij} is equal to zero, this means that the corresponding actuator is not used to control the associated DoF, and $W_{ij}H_{ij}$ remains equal to zero regardless of the value of W_{ij}. Thus, there is a weight matrix capable of redistributing the forces in the CAM in order to overcome the faults modelled by the fault vector. Regarding the bounds of $\mathbf{W}$,

they depend on the system and the characteristics of the actuators. Determining the bounds of $\mathbf{W}$ involves simulating the most severe fault scenario (by setting as many actuators as possible to 0% efficiency before the system is no longer overactuated), and then heuristically identifying the bounds of $\mathbf{W}$ to compensate for this situation. The idea is that if these bounds can effectively address the most severe fault scenario, they will also be sufficient to compensate for less severe faults. Having introduced the weight matrix, the next section will concentrate on its update using a DRL-based method.

3.2.2 Learning the weight matrix with DRL

This section presents the method used to perform control reallocation based on a DRL strategy in the presence of actuator faults.

General principle of the DRL-based reallocation method

The purpose of DRL is to update the CAM using the weight matrix $\mathbf{W}$. In the event of actuator faults, this update takes place at each time step, which corresponds to the reception of observation data. Since the objective is to bypass the explicit fault diagnosis, it is not possible to break down the problem of control reallocation into smaller subproblems depending on the fault. Therefore, a single DRL model is used to perform control reallocation and aims to overcome all potential actuator faults. The output of the DRL model is then the set of values of the matrix $\mathbf{W}$, as shown in Figure 3.4.

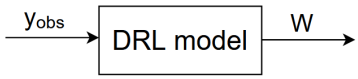


Figure 3.4: Input and output of the DRL model.

The input of the DRL model is the system observation vector, which contains information about the position, orientation, and velocities. This means that the DRL model relies on the implicit recognition of fault patterns in the system's behaviour in order to redistribute the forces among all the actuators, without any need for explicit fault diagnosis. If using the matrix $\mathbf{W}$ leads to an unfeasible solution (in the sense defined in section 2.1.1), the actuator input vector is truncated using the Equation 2.5.

Training process and reward definition

The aim of the DRL-based control reallocation method is to maintain the control of the desired forces contained in the reference vector $\mathbf{w_{ref}}$. The training phase of the DRL model consists of episodes, as defined in section 2.3.1. During the training phase, as summarised in Algorithm 2, the DRL model is exposed to various reference vectors $\mathbf{w_{ref}}$, which are randomly drawn from a uniform distribution at the start of each episode. This ensures that the model learns to adapt to a wide range of faulty situations. During training, the autonomous system is controlled by a model-based controller, while the DRL model is trained on this controlled system. With regard to the application of actuator faults during training, a fault vector $\mathbf{f}$ is also randomly drawn from a uniform distribution for each episode, enabling the DRL model to experience a wide range of potential actuator fault situations. The fault vector must ensure that the system remains controllable despite the fault.

Keeping the control of the forces despite faults means eliminating undesired translations and rotations due to actuator faults. The undesired translation, denoted e_T, and undesired rotation, denoted e_R, can be expressed as the difference between $\mathbf{y_{obs}}$ and $\mathbf{w_{ref}}$. The reward expression, whose function is explained in section 2.3.1, is designed to minimise unwanted translations and rotations. To achieve this, we employ the negative exponential of the translation and rotation errors, as this approach effectively penalises larger errors more significantly, aiming to provide rapid convergence towards the ideal action policy. The derivatives of the translation and rotation errors are added in the negative exponential to penalise rapid changes in velocity and angular rate, thus avoiding erratic movements. The number of steps in an episode, denoted as step_{max}, varies based on the considered scenario and on the types and number of potential actuator faults.

Algorithm 2 Step Processing and Reward Calculation

```
for each episode do
    Randomly choose a reference vector w_ref.
    Randomly choose and apply a fault vector f
    for each step do
        if step < step_max then
            r = − exp(e_T + e_R + ė_T + ė_R)
        else
            End of episode
        end if
    end for
end for
```

Choice of the DRL algorithm

The choice of the DRL algorithm used for this training is a key element of the method. As explained in section 2.3.1, there are several types of DRL algorithms. When choosing an appropriate DRL algorithm to update the weight matrix $\mathbf{W}$, several important criteria must be considered. First, the DRL-based control reallocation method requires handling multiple continuous input and output variables. Therefore, the selected algorithm should be well-suited for environments where continuous actions are required, in order to allow smooth adjustments in response to actuator faults.

Additionally, efficiency in learning from a limited number of samples is crucial for autonomous dynamic systems. In practical applications, a large amounts of interaction data may not be available due to physical constraints or time limitations. Hence, the DRL algorithm should be designed for sample efficiency, ensuring that the model can learn effectively from fewer interactions and still achieve a high level of performance.

In addition to sample efficiency, the chosen DRL algorithm must be capable of retaining a large amount of information throughout the learning process. This is particularly important in our training process where the system needs to account for multiple variables simultaneously and be confronted with multiple fault situations.

Furthermore, the selected DRL algorithm should excel in exploration, allowing it to discover effective control strategies while avoiding suboptimal solutions. Exploration efficiency ensures that the system thoroughly evaluates a wide range of potential actions, leading to better redistribution of forces in the case of actuator faults. An algorithm with strong exploration capabilities will improve the system's adaptability.

Having described the DRL-based method used to adjust the CAM in the presence of actuator faults, the next step is to integrate this modified matrix into the model-based control framework. The following section details how the updated CAM is embedded in the control architecture to ensure stability.

3.3 Integration of the adjusted Control Allocation Matrix in the model-based control framework

The proposed approach aims to perform control reallocation while maintaining the stability of the closed-loop system. In control theory, explicitly modelling time-varying parameters in the state-space equations is necessary for analysing and potentially preserving the stability of a controller operating under such variations (Mohammadpour and Scherer 2012). This is why the adjusted CAM, needs to be explicitly incorporated into the system dynamics.

To achieve this integration, the standard linear evolution equation involving the control output $\boldsymbol{\tau}$ (see Equation 3.1) is rewritten to instead include the actuator input vector $\mathbf{u}(t)$ and the adjusted CAM. This is why the system evolution is defined as follows:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\left(\mathbf{W}(t) \circ \mathbf{H}\right)\mathbf{u}(t) \tag{3.6}$$

The control objective is then to design a model-based controller whose output is the actuator input vector $\mathbf{u}(t)$. The class of controllers targeted by the proposed approach includes those for which the gain matrix $\mathbf{K}(t)$ can be computed as a function of the system matrices and the modified CAM:

$$\mathbf{K}(t) = \mathcal{F}\big(\mathbf{A}, \mathbf{B}(\mathbf{W}(t) \circ \mathbf{H}), \mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_n\big) \tag{3.7}$$

Here, $\mathcal{F}$ is a synthesis operator that depends on the selected control strategy (e.g., LQR, $\mathcal{H}_\infty$, pole placement), and $\mathbf{Q}_1, \ldots, \mathbf{Q}_n$ are the weighting matrices specific to the control design method.

The actuator input is then calculated as:

$$\mathbf{u}(t) = -\mathbf{K}(t)\mathbf{x}(t) \tag{3.8}$$

This formulation enables local stability analysis. Specifically, for any time instant t_0, the local asymptotic stability of the closed-loop system can be verified by checking the eigenvalues of the matrix $\mathbf{A} - \mathbf{B}(\mathbf{W}(t_0) \circ \mathbf{H})\mathbf{K}$. This can be done using a Lyapunov approach: if there exists a symmetric positive definite matrix $\mathbf{P}$ such that:

$$\mathbf{P}(\mathbf{A} - \mathbf{B}(\mathbf{W}(t_0) \circ \mathbf{H})\mathbf{K}) + (\mathbf{A} - \mathbf{B}(\mathbf{W}(t_0) \circ \mathbf{H})\mathbf{K})^\top \mathbf{P} = -\mathbf{L} \tag{3.9}$$

for some positive definite matrix $\mathbf{L}$, then the closed-loop system is locally asymptotically stable at time t_0 (Benner 2006).

For a given weight matrix, local stability can be guaranteed as described above, but the unpredictable nature of DRL makes global stability challenging to achieve. More precisely, global stability is challenging to achieve because the variations in control actions generated by the DRL model are not continuous and may exhibit large variations. For global stability to be established, these variations would need to be smooth and continuous.

However, it is possible to approach global stability by adding a post-processing step to the DRL model's outputs. The variations in the values of the weight matrix can be reduced, for instance, by applying low-pass filters or moving averages (Pfister 2017). This way, the outputs of the DRL model become smoother, approximating small and continuous variations, which helps mitigate the risk of large discontinuities in control actions. The specific type of post-processing employed can vary significantly based on the characteristics of the system

being considered. Different systems may exhibit distinct dynamics and sensitivities to control inputs, necessitating tailored approaches to post-processing. For instance, a highly dynamic system might benefit from more aggressive filtering techniques, while a more stable system could require less intervention.

Figure 3.5 illustrates the integration of the adjusted CAM into the computation of the controller gain at each time step.

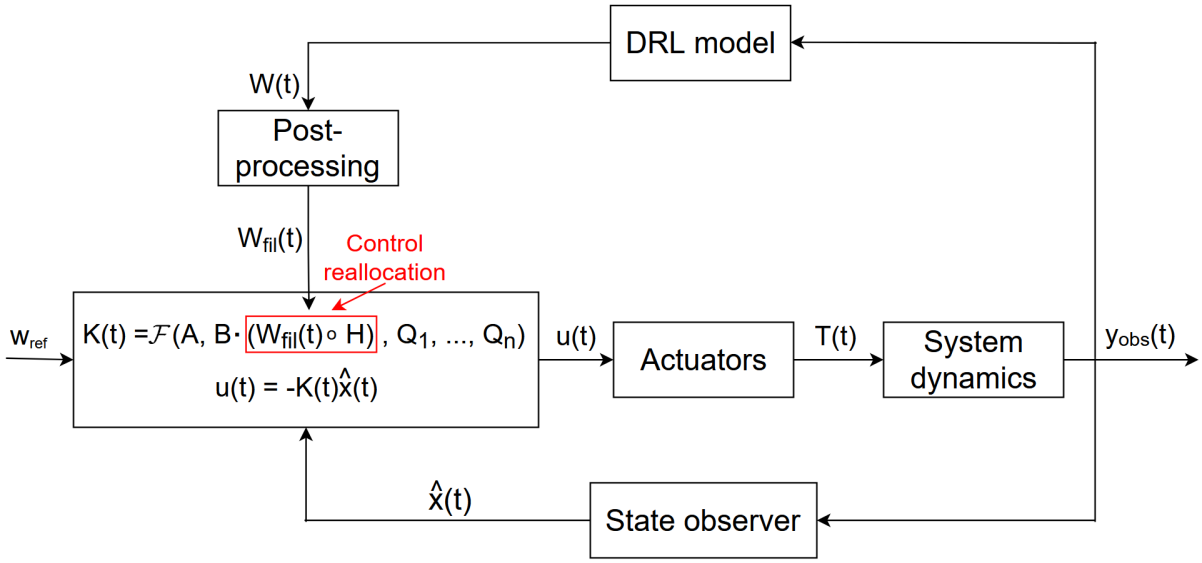


Figure 3.5: Detailed proposed hybrid controller structure.

In this figure, the variable $\hat{\mathbf{x}}$ denotes the estimated state vector and the matrix $\mathbf{W}_{\text{fil}}$ represents the weight matrix after post-processing, which is applied to reduce abrupt variations and discontinuities in the DRL outputs.

3.4 Conclusion

The core concept of the proposed approach is to perform control reallocation with DRL to overcome actuator faults in autonomous dynamic systems without the need for explicit fault diagnosis. This approach is designed to support the research statement by leveraging the adaptability and generalisability of DRL in conjunction with the stability of a model-based control framework.

The structure of the proposed approach relies on two main points that have been presented in this chapter: the training of the DRL model to overcome actuator faults, and the integration of control reallocation into the system model equations on which the model-based controller is built. This structure implies, on the one hand, that the DRL model must be pre-trained on various actuator fault scenarios, and on the other hand, that the controller directly outputs the actuator inputs.

Once trained, the DRL model is ready to be embedded within the controller. During a mission, its purpose is to adjust the CAM in real time, based on the knowledge acquired during training.

The next chapter focuses on a practical application of this methodology, specifically addressing actuator faults in an AUV. The aim of this case study is to validate the proposed approach in a scenario where actuator fault management is crucial.

Chapter 4

Implementation of the proposed approach on an Autonomous Underwater Vehicle

This chapter presents the implementation of the proposed approach in the context of AUV actuator fault scenarios. AUVs are self-propelled underwater robotic platforms capable of performing missions without direct human intervention. They are widely used for oceanographic research, underwater inspection, environmental monitoring, and military operations (He, Xie, and Ya Zhang 2025).

The case study of an AUV has been chosen because actuator faults are particularly critical in such systems: AUVs operate in environments that are not easily accessible to human operators, and an actuator fault can jeopardise the mission or even lead to the complete loss of the vehicle (Omerdic and Roberts 2004). AUV actuator faults are largely due to the harsh and unpredictable nature of the underwater environment which can lead to partial or total loss of propulsion. Examples of actuator faults and possible causes of these faults that can be found in the literature are mentioned in Appendix A.

In particular, the BlueROV2 Heavy (BlueRobotics) is considered in this work, as its architecture ensures that the system remains overactuated even in the event of a total fault of one actuator, which is consistent with the assumption stated in Chapter 1. Moreover, the complex geometry of its thrusters, which introduces coupling between multiple degrees of freedom, presents a realistic scenario to evaluate the performance of the proposed hybrid control approach as explicit fault diagnosis is challenging to obtain.

For this application case, the working hypothesis that **the state of the system is known** is made: as the focus is on the control of the system, it is assumed that the state vector of the BlueROV2 Heavy (including position, orientation, and velocity) is available and accurate. As stated in the introduction, **no external disturbances are taken into account in this work**; in particular, marine currents are not considered.

In this chapter, section 4.1 will provide an overview of BlueROV2 Heavy specifications

and simulation, followed by the application case scenario and control design described in section 4.2. Finally, section 4.3 will detail the specificities of the DRL training process in this case study.

4.1 BlueROV2 Heavy

In this section, we will provide a detailed overview of BlueROV2 Heavy, focusing on its specifications and simulation. First, we will examine the key components of the vehicle, including its thruster configuration, which play a crucial role in the impact of faults on the behaviour of the drone. Following this, we will discuss the choice of the simulation platform to model BlueROV2 Heavy, which is essential for training the DRL algorithm.

4.1.1 BlueROV2 Heavy specifications

BlueROV2 Heavy is an AUV from BlueRobotics commonly used in marine research, infrastructure inspection, and environmental monitoring. Figure 4.1 shows its design:



Figure 4.1: BlueROV2 design (source: BlueRobotics).

At its core, the vehicle is equipped with a Pixhawk autopilot (Meier et al. 2011) and a Raspberry Pi (Upton and Halfacree 2016) computer. The Pixhawk, an open-source autopilot, manages low-level control tasks such as stabilisation and sensor data processing, while the Raspberry Pi handles high-level control, communication with the ground station, and custom programming for control autonomy. The sensor suite includes an IMU, a depth sensor used for altitude control and a camera for visual inspection. Although the BlueROV2 Heavy is technically a Remotely Operated Vehicle (ROV), we will use it with its autonomous control capabilities. While the input and output data for the controller will be transmitted through a tethered connection between the robot and a computer, the drone will be operated in autonomous control mode.

BlueROV2 Heavy is propelled by eight thrusters, achieving six-degree-of-freedom manoeuvrability. Its state vector is composed of the variables associated with these six degrees of freedom: x, y, z, $roll$, $pitch$, and yaw represented in Figure 4.2.

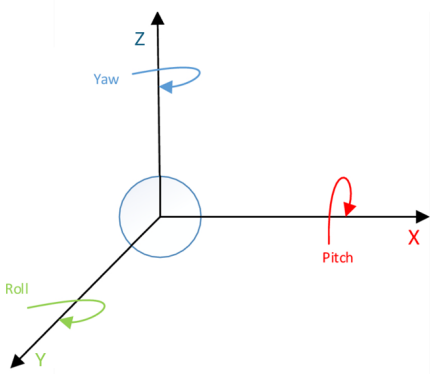


Figure 4.2: Representation of the six degrees of freedom (X. Li 2022).

The state vector $\mathbf{x}$ can be represented as:

$$\mathbf{x} = \begin{bmatrix} x & y & z & \phi & \theta & \psi & \dot{x} & \dot{y} & \dot{z} & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T \tag{4.1}$$

where ϕ, θ, and ψ represent the roll, pitch, and yaw angles, respectively, and $\dot{x}$, $\dot{y}$, $\dot{z}$, $\dot{\phi}$, $\dot{\theta}$, and $\dot{\psi}$ denote the linear and angular velocities along and around the body-fixed axes x, y, and z.

There are four horizontal thrusters $(T0, T1, T2, T3)$, controlling the x, y and *yaw* directions, and four vertical thrusters, controlling the z, *roll* and *pitch* directions. The thruster configuration is shown in Figure 4.3. The green thrusters rotate clockwise and the blue thrusters rotate counterclockwise. Since there are eight thrusters to control six degrees of freedom, redundancy is maintained even in the event of a thruster total fault, making it suitable for applying the proposed method.

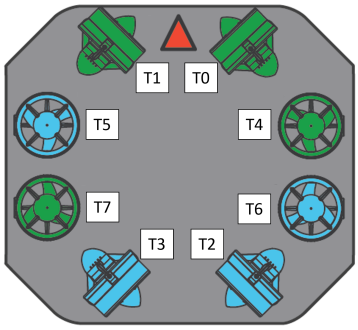


Figure 4.3: Thruster configuration (source: BlueRobotics).

As described in Equation 2.1, the CAM $\mathbf{H}$ links control inputs to the actuator inputs through the equation $\boldsymbol{\tau} = \mathbf{Hu}$, where $\boldsymbol{\tau}$ represents the desired control forces and moments, and $\mathbf{u}$ is the vector of actuator inputs.

H is expressed as:

$$\mathbf{H} = \begin{bmatrix} 0.707 & 0.707 & -0.707 & -0.707 & 0 & 0 & 0 & 0 \\ -0.707 & 0.707 & -0.707 & 0.707 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & -1 & 1 \\ 0.06 & -0.06 & 0.06 & -0.06 & -0.218 & 0.218 & 0.218 & -0.218 \\ 0.06 & 0.06 & -0.06 & -0.06 & 0.120 & 0.120 & 0.120 & -0.120 \\ -0.1888 & 0.1888 & 0.1888 & -0.1888 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.2}$$

The values of **H** are provided by the manufacturer (from BlueRobotics) and take into account the geometry and orientation of each thruster to ensure proper force distribution. This is the matrix that will be modified through the weight matrix **W**, using DRL, by applying the proposed approach.

The actuator input vector **u** is defined as:

$$\mathbf{u} = \begin{bmatrix} T_0 & T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 \end{bmatrix}^T \tag{4.3}$$

where $T_0, \ldots, T_8$ represent the inputs of the thrusters, and $\boldsymbol{\tau}$ includes the forces $F_{x,y,z}$ and moments $M_{x,y,z}$ acting on the vehicle in the body-fixed frame:

$$\boldsymbol{\tau} = \begin{bmatrix} F_x & F_y & F_z & M_x & M_y & M_z \end{bmatrix}^T \tag{4.4}$$

Having described the specifications of the BlueROV2 Heavy, the next step is to design a DRL training environment.

4.1.2 Training environment simulation

Training the DRL model directly on physical BlueROV2 Heavy is largely impractical due to the significant time, energy, and operational costs involved, as well as the potential risks of hardware damage and safety concerns associated with real-world experimentation (Zhao Wang et al. 2024). Consequently, simulation-based training has become a preferred method, followed by a transfer of the trained model to the physical drone.

Underwater simulators are available (Lechene et al. 2024), along with plugins that can be added to simulation software to replicate a submarine environment and an underwater vehicle. However, despite recent advances, the effective use of open-access tools to train DRL models in simulation for deployment on real-world AUVs remains uncommon. The key challenge, which is addressed in this thesis, is the difficulty of connecting the most

realistic possible simulation platform with a suitable and effective DRL training environment (W. Zhao, Queralta, and Westerlund 2020). The chosen simulation software is GAZEBO Simulator (Koenig and Howard 2004), an open-source simulator widely used in robotics for its realism in motion and system-environment interactions. The plugin employed to model the underwater environment is UUV Simulator (Manhães et al. 2016), as it is well-documented and his efficiency for training DRL has been shown (Chaffre, Le Chenadec, et al. 2021). UUV Simulator includes features such as realistic physics engines to model underwater movements, different simulations of AUVs, and the ability to simulate various mission scenarios (Manhães et al. 2016). Figures 4.4 and 4.5 show the BlueROV2 Heavy model and the marine environment in GAZEBO with the UUV Simulator plugin.

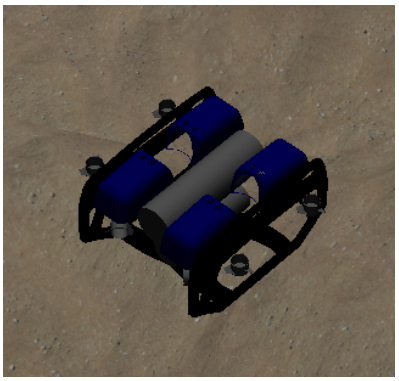


Figure 4.4: BlueROV2 Heavy model with UUV Simulator.

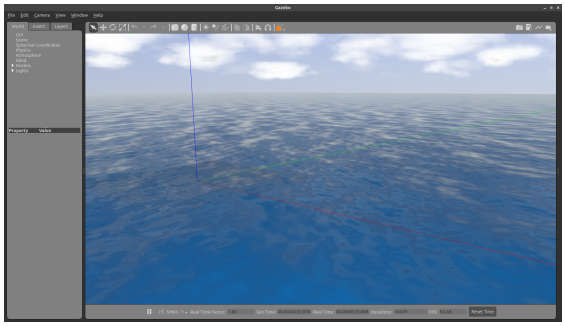


Figure 4.5: Illustration of the marine environment with UUV Simulator (Manhães et al. 2016).

In this work, the Robot Operating System (ROS) (Quigley et al. 2009) framework was used for sensors communication and actuators control. ROS is the most commonly used open-source meta operating system for robots (Macenski et al. 2022). It provides communication bridges to enable the transmission of data inside the GAZEBO simulations. The ROS tools and libraries are used for managing hardware interfaces, controlling robotic components, and handling communication between different system parts. In this framework, nodes represent individual software components responsible for various tasks within a robotic system. Nodes communicate with each other by publishing and subscribing to topics. Each topic corresponds to a specific type of data, such as sensor measurements or control commands. In this work, Python (VanRossum and Drake 2010) was the primary programming language used to implement the controller and to connect the ROS framework with the DRL training environment.

In order to train a DRL model within this simulation environment, rather than implementing the algorithms from scratch, a connection is established between a repository of DRL algorithms and the simulated DRL agent and environment. This connection is achieved through ROS topics. For the algorithm repository, we chose Stable Baselines3

(Raffin et al. 2021), which is a Python library providing a well-documented and efficient implementation of state-of-the-art DRL algorithms. Additionally, it is particularly well-suited for our use case since we control a single agent in a continuous action space, which aligns with the design of many algorithms within the library. Moreover, it allows the use of custom simulation environments, facilitating efficient experimentation and development. We chose to use OpenAI Gym (Brockman et al. 2016) as the framework for training our DRL model, which is a toolkit developed to standardise the development and benchmarking of reinforcement learning algorithms by providing a collection of environments with a common interface. To connect this framework to ROS, we used OpenAI ROS (Zamora et al. 2016) which is a tool designed to facilitate the integration of reinforcement learning with robotic simulations by providing a structured interface between OpenAI Gym environments and ROS-based robots. The connections between the simulation tools is summarised in Figure 4.6.

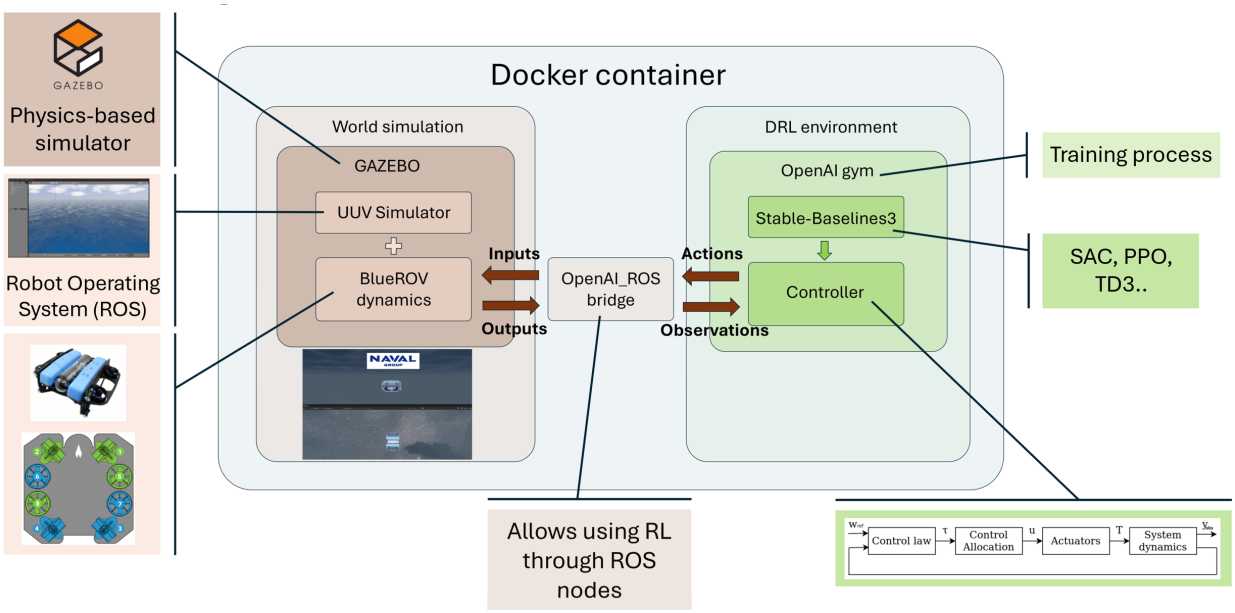


Figure 4.6: Architecture of the training environment.

To establish a connection between Stable Baselines3 and the GAZEBO simulator using OpenAI ROS, several steps were followed to ensure smooth communication and control of the simulated drone. First, the OpenAI ROS framework was used to create a custom environment that is compatible with the OpenAI Gym interface, allowing reinforcement learning algorithms from Stable Baselines3 to interact with the simulation. The drone model, simulated in GAZEBO, was controlled using ROS topics and services, where ROS topics were employed to send control commands (e.g., thruster inputs) and receive sensory feedback (e.g., depth, velocity...). The environment was defined in a custom ROS package, which included a training script that initialises a Stable Baselines3 reinforcement learning agent.

The agent communicates with the drone through an interface that subscribes to the sensors topics and publishes control actions. This implementation is given below, where the action and observation spaces are defined and the communication with ROS topics is established:

Listing 4.1: Definition of the custom Gym environment.

```
class UnderwaterEnv(gym.Env):
    def __init__(self):
        self.action_space = spaces.Box(low=-1, high=1)
        self.observation_space = spaces.Box(low=-1, high=1)

        self.cmd_pub = rospy.Publisher('/bluerov2/cmd_thrust',
        ThrustMsg)
        rospy.Subscriber('/bluerov2/odom', Odometry,
        self.odom_callback)

    def odom_callback(self, msg):
        self.current_state = self._extract_state(msg)
```

The training loop follows the standard Gym environment structure, where actions are taken, observations are received, rewards are computed, and the episode terminates based on a predefined criteria such as mission completion or time limits. The `step()` function illustrates the training loop process:

Listing 4.2: Implementation of the step function.

```
def step(self, done):
    obs = self.current_state
    thrust_msg = ThrustMsg()

    if not done:
        action = output_of_DRL_model(obs)
        thrust_msg.data = action
        self.cmd_pub.publish(thrust_msg)
        reward = compute_reward(obs)

    else:
        end episode
```

This training architecture was the topic of a conference paper published as part of this thesis for OCEANS 2025 (Brest, France). This paper, entitled "Sim-to-Real Transfer

for AUV Fault Control with Deep Reinforcement Learning", presents the simulation tools developed for training the DRL model (Lagattu, Lechene, et al. 2025). Now that we have explored the tools for simulating the underwater drone and the DRL training environment, the next section focuses on the framework of the application case for the proposed approach, providing a detailed overview of the context of application.

4.2 Scenario and control design

In this section, we will focus on the application case scenario of the proposed approach. First, the nature of the ROV mission and its objective will be outlined. Then, the selection of the model-based controller will be discussed, along with an explanation of its implementation.

4.2.1 Application case scenario

In this application case, we consider faults occurring in the four horizontal thrusters $T0$, $T1$, $T2$ and $T3$. This approach simplifies the analysis of the proposed approach, while the redundancy of these thrusters ensures that overactuation is maintained in the horizontal plane even in the event of a total thruster fault. We consider fault scenarios in which only one actuator is faulty at a time. The single actuator fault is simulated by applying a fault coefficient f to the faulty thruster, taking discrete values of:

$$f \in \{0, 0.1, 0.2, \ldots, 1\}$$

To take a first step in assessing the impact of the DRL-based control reallocation method, a simple mission is considered and is shown in Figure 4.7.

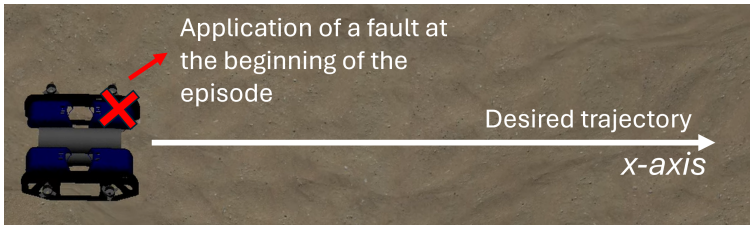


Figure 4.7: Scenario for the DRL model training.

The proposed scenario involves following a straight-line trajectory along the x-axis with a fixed desired depth and no desired rotations. A forward velocity of $0.2\,\mathrm{m/s}$ is selected, as this represents a reasonable speed for operating the drone in real-world conditions. The reference vector is then given by:

$$\mathbf{w_{ref}} = [(x + \delta), 0, z_{ref}, 0, 0, 0, 0.2, 0, 0, 0, 0, 0]^T \tag{4.5}$$

where δ represents an adjustment for trajectory tracking along the x-axis and z_{ref} is fixed and corresponds to the desired depth of the drone.

At the beginning of the mission (the drone starts at the desired depth), a fault occurs in one of the four horizontal thrusters. In simulation, the abnormal behaviour is detected immediately as the state is assumed to be known and there are no other external disturbances. Upon detection, the DRL-based control reallocation method is activated to manage the fault condition. A more accurate threshold for detection of an abnormal behaviour of the system will be determined based on an analysis of the simulation results in Chapter 5, and will be applied in real-world conditions. To make the resulting actuator inputs feasible (as discussed in 2.1.1), a truncation, as described in Eq. 2.5, is applied on $\mathbf{u}$. Regarding the saturation value defined in 2.1.1, in our simulation, $\forall i \in [1, n], s_i = 5$ (in simulation units). The next section examines the design of the hybrid controller of the ROV.

4.2.2 BlueROV2 Heavy controller

As discussed in section 2.1.2, a classic model-based controller for linear systems is the LQR controller. This is the chosen controller for this application case due to its effectiveness in optimally controlling linear systems and the straightforward establishment of its stability. To implement the LQR controller for our proposed approach, the complete model equations of the BlueROV2 Heavy are required. In this section, the model equations of the drone are first presented, followed by the implementation of the LQR controller, and finally the integration of control reallocation using DRL is described.

Modelling of the BlueROV2 Heavy

The modelling of an AUV can be summarised in the state-space representation form (Thor I. Fossen, Tor Arne Johansen, and Perez 2009) as:

$$\begin{cases} \dot{\boldsymbol{\eta}} = \mathbf{J}_{\Theta}(\boldsymbol{\eta})\boldsymbol{\nu} \\ \dot{\boldsymbol{\nu}} = \mathbf{M}^{-1}\left(\boldsymbol{\tau} - \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} - \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} - \mathbf{g}(\boldsymbol{\eta})\right) \end{cases} \tag{4.6}$$

where $\boldsymbol{\tau}$ is the control input, $\boldsymbol{\eta} = [x, y, z, \phi, \theta, \psi]^{\top}$ represents the position and orientation of the BlueROV2 Heavy in the inertial frame, and $\boldsymbol{\nu}$ denotes the linear and angular velocities in the body-fixed frame. The Jacobian matrix $\mathbf{J}_{\Theta}(\boldsymbol{\eta})$ defines the transformation between velocities in the body-fixed frame and the inertial frame, and depends on the current orientation of the vehicle. $\mathbf{M}$ represents the inertia matrix, the term $\mathbf{C}(\boldsymbol{\nu})$ encompasses the hydrodynamic Coriolis and centripetal forces matrix, $\mathbf{D}(\boldsymbol{\nu})$ describes the hydrodynamic damping forces vector and $\mathbf{g}(\boldsymbol{\eta})$ is the vector of gravity and buoyancy effect. The full system

dynamics, combining these components, are expressed as:

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{J}_\Theta(\boldsymbol{\eta})\boldsymbol{\nu} \\ \mathbf{M}^{-1}\left[\boldsymbol{\tau} - (\mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{g}(\boldsymbol{\eta}))\right] \end{bmatrix} \tag{4.7}$$

where the state vector of the BlueROV2 Heavy is defined as $\mathbf{x} = [\boldsymbol{\eta}, \boldsymbol{\nu}]^\top$.

LQR controller implementation

The system must be expressed in the linear form $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\boldsymbol{\tau}$ to allow the application of the LQR controller. The matrix $\mathbf{A}$ corresponds to the Jacobian of the system dynamics with respect to the state vector, while the matrix $\mathbf{B}$ is the Jacobian with respect to the control output vector. The $\mathbf{A}$ matrix for the BlueROV2 Heavy model is a 12×12 matrix, defined by:

$$\mathbf{A} = \begin{bmatrix} \frac{\partial \dot{\boldsymbol{\eta}}}{\partial \boldsymbol{\eta}} & \frac{\partial \dot{\boldsymbol{\eta}}}{\partial \boldsymbol{\nu}} \\ \\ \frac{\partial \dot{\boldsymbol{\nu}}}{\partial \boldsymbol{\eta}} & \frac{\partial \dot{\boldsymbol{\nu}}}{\partial \boldsymbol{\nu}} \end{bmatrix}$$

And the $\mathbf{B}$ matrix is a 12×6 matrix defined by:

$$\mathbf{B} = \begin{bmatrix} \mathbf{0}_{6\times 6} \\ \mathbf{M}^{-1} \end{bmatrix} \tag{4.8}$$

The computation of these matrices is detailed in Appendix B. In these expressions of model matrices, matrix $\mathbf{B}$ is fixed, but matrix $\mathbf{A}$ depends on the linear and angular velocities as well as the orientation of the drone when no approximation is made around an equilibrium point. Consequently, we need to place the system in the vicinity of an equilibrium state to make this matrix constant. In our case, the scenario considered is line-following along the x-axis. Thus, we choose to approximate the dynamics around this specific steady-state condition, assuming a constant forward velocity of $0.2\,\mathrm{m/s}$ and zero values for the other velocities and rotations. The fixed $\mathbf{A}$ matrix is:

$$\mathbf{A}(t) = \mathbf{A}(\dot{x} = 0.2, \dot{y} = 0, \dot{z} = 0, \dot{\phi} = 0, \dot{\theta} = 0, \dot{\psi} = 0) \tag{4.9}$$

During the transition states of the ROV, *i.e.* when it starts or stops, we maintain the matrix $\mathbf{A}$, which varies according to the state. It is only when the drone reaches a constant speed along the x-axis that the matrix $\mathbf{A}$ is fixed. This is because the dynamics of the system are influenced by changing velocities and angles during these transition phases, which

necessitates a more adaptable representation of the state equations. Once a steady speed is achieved, the system becomes more stable, allowing us to simplify the model by using a constant matrix $\mathbf{A}$ that reflects the operational conditions of the drone. This simplification enables us to approximate the system as a linear system, allowing the application of linear control techniques.

In the context of the proposed approach, the gain $\mathbf{K}$ of the LQR controller is calculated based on the matrices $\mathbf{A}$, $\mathbf{BH}$, $\mathbf{Q}$ and $\mathbf{R}$ as shown in Equation 3.7. As seen in section 2.1.2, LQR control aims to minimise the quadratic function defined in Equation 2.15, which depends on the control input $\boldsymbol{\tau}$. However, in the context of our proposed approach, as $\mathbf{B}$ is replaced by $\mathbf{BH}$ in the state equation (see Equation 3.6), the considered quadratic function depends directly on the actuator input $\mathbf{u}$ and can be expressed as:

$$J = \int_0^\infty \left(\mathbf{x}(t)^\top \mathbf{Q}\mathbf{x}(t) + \mathbf{u}(t)^\top \mathbf{R}\mathbf{u}(t)\right) dt \tag{4.10}$$

As a result, the $\mathbf{R}$ matrix imposes penalties directly on the actuator inputs. Since all thrusters are of the same type and have identical characteristics, there is no reason to prioritise the use of one actuator over another. Consequently, $\mathbf{R}$ is defined as the identity matrix $\mathbf{I_8}$. This decision guarantees that all actuators are treated equally and contribute equally to the cost function.

Regarding the $\mathbf{Q}$ matrix which penalises the state variables, as explained in section 2.1.2, it can be determined with optimisation methods such as PSO (Kennedy and Eberhart 1995). In a LQR controller, the matrix $\mathbf{Q}$ is not required to be diagonal, but it is often chosen as such for simplicity and clearer interpretation of penalties for state variables as it indicates independent penalties (Benner 2006). For simplification, we do not control the roll and pitch of the drone, as it tends to be stable in these directions without the need for control. Additionally, these axes are minimally affected by faults in the horizontal thrusters. Therefore, the coefficients of $\mathbf{Q}$ associated with roll and pitch are set to zero.

To determine the values of the $\mathbf{Q}$ matrix with a PSO technique, we begin with an initial state vector that has random small variations relative to the reference vector. We then aim to minimise the global cost, which is the square of the error between the current state and the reference vector. The state evolves according to the dynamic equation $\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{BHu}$. By adjusting the coefficients in $\mathbf{Q}$ with PSO to penalise the deviations from the desired state, we can move toward optimal values of the $\mathbf{Q}$ matrix. Figure 4.8 shows the convergence curve of the PSO method, obtained with the BlueROV2 Heavy matrices.

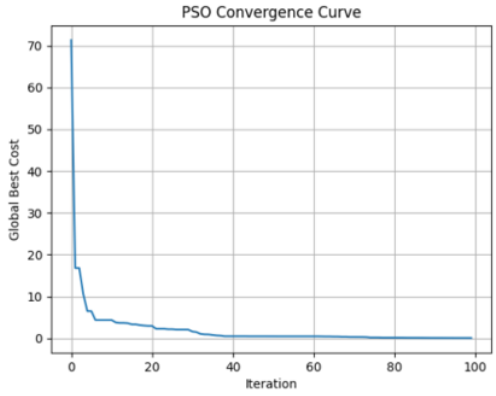


Figure 4.8: Convergence curve of PSO technique to determine the **Q** matrix.

The curve converges towards a global cost of 0, indicating that this technique has allowed to identify an effective **Q** matrix. The obtained **Q** matrix is the following:

$$\mathbf{Q} = \text{diag}(13.28, 23.05, 28.37, 0, 0, 13.27, 10.06, 16.12, 19.93, 0, 0, 9.46) \tag{4.11}$$

Now that the parameters required for the implementation of the LQR controller on the BlueROV2 Heavy have been identified, we now focus on the control reallocation process and establish the link with the computation of the LQR controller gain.

Performing control reallocation with DRL

As we are only considering the control of the BlueROV2 Heavy over the x direction and as the objective is to keep the control over each DoF, the CAM must be updated on the line associated with the direction of x. Then, the updated CAM can be reduced to:

$$\mathbf{W}(t) \circ \mathbf{H} = \begin{bmatrix} W_{11}(t)H_{11} & \cdots & W_{1r}(t)H_{1r} \\ H_{11} & \cdots & H_{1r} \\ \vdots & & \vdots \\ H_{n1} & \cdots & H_{nr} \end{bmatrix} \tag{4.12}$$

With $n = 4$ for 4 thrusters, and $r = 6$ for 6 degrees of freedom, and the learned matrix **W**, which is the output of the DRL model, can be reduced to the vector:

$$\mathbf{W}(t) = (W_{11}(t), W_{12}(t), W_{13}(t), W_{14}(t)) \tag{4.13}$$

The DRL continuous action space is defined as $[-1, 1]^4$, a choice that provides a normalised and symmetric range for learning, which helps stabilise training. To apply these actions to the physical system, this normalised space must be mapped to actuator commands.

Since the agent aims to minimise tracking errors without explicitly controlling the speed, the bounds of the weights help avoid behaviours that are too slow, or a saturation of the thrusters. In this work, these bounds were chosen by simulating a total fault of one thruster and, through trial and error, adjusting the coefficients applied to the remaining thrusters under a standard controller (without DRL) to follow the intended trajectory with minimal deviation and acceptable speed. This heuristic process led to the determination that a remapping of the action space to the physical range $[0.3, 3.5]^4$ was necessary, and this remapping was applied during training. At each CAM update, the gain $\mathbf{K}(t)$ of the LQR controller is re-computed to ensure local stability, by solving the Riccati equation which includes the adjusted CAM:

$$\mathbf{A}^\top \mathbf{P} + \mathbf{P}\mathbf{A} - \mathbf{P}\mathbf{B}(\mathbf{W}(t) \circ \mathbf{H})\mathbf{R}^{-1}(\mathbf{B}(\mathbf{W}(t) \circ \mathbf{H}))^\top \mathbf{P} + \mathbf{Q} = \mathbf{0} \tag{4.14}$$

The updated gain $\mathbf{K}(t)$ is then used to compute the actuator inputs $\mathbf{u}(t)$ (as detailed in Equation 3.8) which are applied to the system. Now that we have framed the scenario and detailed the implementation of the proposed hybrid controller of BlueROV2 Heavy, the following section focuses on the DRL training, including the selection of the DRL algorithm and the training process.

4.3 Deep Reinforcement Learning algorithm selection and training process

In this section, we begin by discussing the selection of the DRL algorithm used in the proposed FTC control strategy, highlighting its relevance for handling actuator faults in this application. We then examine the training process to apply the proposed approach to the BlueROV2 Heavy, focusing on how the learning of the DRL agent, described for the general case in Chapter 3, is optimised.

4.3.1 Choice of Deep Reinforcement Learning algorithm

As discussed in section 3.2.2, several criteria are considered to select a DRL algorithm suitable for control reallocation in dynamic systems: the need for continuous control, sample efficiency and strong exploration capabilities. Among the main DRL algorithms, SAC algorithm stands out for its effectiveness in changing environments, especially when compared to policy-based methods such as PPO algorithm (Schulman et al. 2017). As discussed in section 2.3.1, PPO algorithm is known for its stable performance and efficiency, but it has low sample efficiency, needing large amounts of interaction data to train effectively. This limitation makes it less efficient in applications where data or time is limited. In our case, where the number of fault situations is significant, an efficient algorithm for training is

required to limit the training duration. The off-policy nature of the SAC algorithm allows it to learn from previously collected data stored in a replay buffer, improving sample efficiency and enabling better reuse of experience compared to on-policy methods. SAC algorithm's design therefore allows it to learn effectively with fewer samples, making it better suited to the application case. SAC algorithm's continuous control capability is also advantageous. Compared to other continuous control methods like DDPG algorithm, SAC algorithm offers higher robustness and resilience to changes in the environment. DDPG algorithm, although built for continuous actions, is sensitive to noise and disturbances, which can decrease performance in unpredictable conditions as mentioned in section 2.3.1. SAC algorithm, however, uses entropy maximisation to enhance exploration and adapt to changes. This makes it more reliable for thruster fault management, where adaptation to various situations is needed.

Moreover, as seen in section 2.3.3, it has been demonstrated in the work of Chaffre, Le Chenadec, et al. (2021) that SAC algorithm is effective for the adaptive control of a ROV, both in simulation and in real-world applications. The results from these authors demonstrated that SAC algorithm is suitable for adjusting the gains of a PID controller to overcome external disturbances. This suggests that SAC algorithm can also be suitable for adjusting the CAM in the case of thruster faults in the BlueROV2 Heavy. For these reasons, SAC algorithm is chosen for the application case. The selected hyperparameters for the SAC algorithm are the standard ones, which have been established through extensive research and practical applications (Haarnoja et al. 2018). This choice is justified as these default settings provide a robust starting point, ensuring that the algorithm operates effectively across various scenarios. By using the default hyperparameters, we can leverage a configuration that allows efficient training and adaptation. The hyperparameters that are used in the application case are shown in Table 4.1.

Hyperparameter	**Value**
Learning Rates	3×10^{-4}
Discount Factor (γ)	0.99
Target Network Update Rate	0.005
Replay Buffer (D) Size	1×10^{6}
Batch Size	256
Number of Hidden Units	256
Number of Hidden Layers	2
Activation Function	ReLU

Table 4.1: Chosen hyperparameters for the SAC algorithm.

4.3.2 Preliminary validation of the Soft Actor-Critic algorithm

In order to experimentally validate the choice of the SAC algorithm for overcoming actuator faults in underwater vehicles, a preliminary study was conducted in simulation using the DRL model in an end-to-end control configuration. The DRL model directly produced the control input as shown in Figure 4.9, with the goal of limiting undesired deviations from the drone's trajectory in case of thruster faults. It therefore acted as a purely machine learning-based controller.

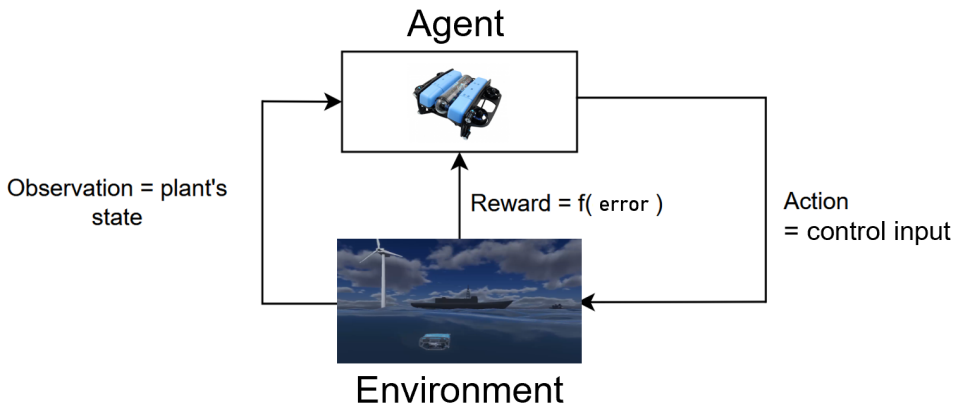


Figure 4.9: End-to-end DRL control training process.

The details of the training environment and some of the results are presented in Appendix C and can be found in the paper entitled "DRL-Based Thruster Fault Recovery for Unmanned Underwater Vehicles", published for the Australian and New Zealand Control Conference (ANZCC) 2024 (Gold Coast, Australia) as part of the work on this thesis (Lagattu, Le Chenadec, et al. 2024). The scenario used in this preliminary study was slightly different from the one described in this thesis, which is why the details are provided in the appendix.

The results show that the end-to-end DRL controller enables the vehicle to overcome thruster fault scenarios and performs more efficiently than a standard PID controller in fault conditions. However, this type of purely machine learning-based controller has limitations. In particular, the control inputs produced by the DRL model exhibit high-frequency and irregular variations (see Appendix C), unlike the smooth outputs of a standard PID controller. This behaviour arises from the lack of explicit constraints on action smoothness during training and the sensitivity of neural network policies to small changes in the observed state. While one could attempt to mitigate this by adding penalties on action variation in the reward function or applying post-processing filters to the control signals, such adjustments do not guarantee closed-loop stability as discussed 2.3.2.

This preliminary test validates the use of the SAC algorithm to learn how to overcome thruster faults on an AUV, and highlights the need for a model-based control architecture.

The following section details the training procedure used for the proposed hybrid controller combining DRL and model-based control.

4.3.3 Deep Reinforcement Learning training process

During the DRL training, an episode proceeds as follows. As the scenario involves following a straight line along the x-axis, the drone's position is set to:

$$(x = -1 \text{ m},\ y = 0 \text{ m},\ z = z_{\text{ref}},\ \phi = 0 \text{ rad},\ \theta = 0 \text{ rad},\ \psi = 0 \text{ rad})$$

with both linear and angular velocities set to zero. We select $z_{ref} = -2.5$ m, as this represents the ideal depth in the test pool for safe experiments conducted under real conditions, and corresponds to the distance at which the camera achieves optimal tracking of the ArUco marker (Chaffre, Wheare, et al. 2025). From the beginning to the end of the episode, the vehicle remains at this depth. Regarding the reference vector, we choose $\delta = 1m$ for the straight-line tracking. During the entire episode, the drone is controlled by the LQR. The drone operates under actuator fault conditions for the entire duration of the episode.

As there are numerous considered fault situations, training the DRL model on all possible efficiency levels for each thruster would significantly increase the time required to converge towards an efficient action policy. For this reason, the model is only trained on the most severe fault cases, *i.e.* where f is close to zero. The objective is to leverage the generalisation capability of the DRL model to handle less severe faults occurring during the mission.

As the aim is to learn how to overcome the faults of the four horizontal thrusters individually, each training episode proceeds as follows:

- a thruster T_i is randomly selected to experience a fault, with $i \in \{0, 1, 2, 3\}$;
- a level of efficiency is randomly chosen and applied to thruster T_i. This level can be either 0% (total fault) or 10% (partial fault).

These efficiency levels were chosen to guide the DRL agent's focus towards the most impactful single faults. Evaluation on less severe faults, with efficiency losses exceeding 10%, is then used to assess the generalisation capability of the model. The overall training process is illustrated in Figure 4.10.

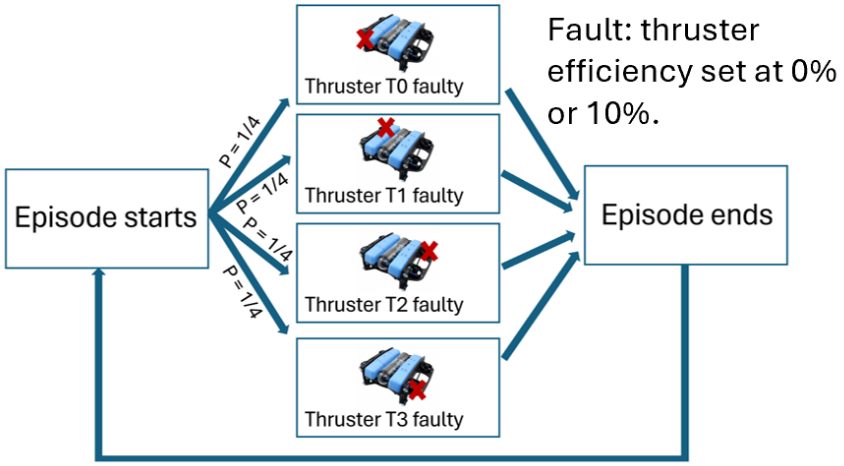


Figure 4.10: Domain randomisation process during DRL training, applied to BlueROV2 Heavy. "p" is the probability rate for each case.

The directions impacted by a fault in one of the horizontal thrusters are the y and yaw directions. This is why the reward function, whose general expression has been detailed in Chapter 3 in Algorithm 2, aims to minimise the y and yaw errors. In order to accelerate the training process, the reward was bounded in this case study. A lower limit of -10 for the reward value was imposed so that the DRL agent would not be excessively penalised, which helps to prevent unstable learning behaviour and encourages faster convergence. Similarly, an upper limit of 1 was applied to provide greater reinforcement when the tracking error is small, which allows promoting the learning of more accurate control policies. The expression of the reward function in the context of our application case scenario is shown in Algorithm 3.

For our application case, the observation vector, which serves as the input to the DRL model, includes the state variables used in the reward function:

$$\mathbf{y}_{\text{obs}}(t) = [y(t),\ \dot{y}(t),\ \psi(t),\ \dot{\psi}(t)] \tag{4.15}$$

This choice ensures that the agent receives all the necessary information to evaluate the reward and adjust its policy accordingly. In addition, the observation vector was normalised before being used both as the input to the DRL model and in the reward function, ensuring that the errors in y and yaw were taken into account in a comparable manner. At each step of the training, the DRL agent outputs the weight vector defined in Equation 4.13.

Underwater drones exhibit significant inertia due to their mass and the resistance of the water, which affects their responsiveness to control inputs (Neto et al. 2015). This is why the inertia of the ROV is taken into account in the control reallocation process via a measurement window of size t_w as shown in Figure 4.11.

Algorithm 3 Fault selection and Reward Calculation

$y_{ref} = 0, \psi_{ref} = 0$
for each episode **do**
  Randomly choose $i \in \{0, 1, 2, 3\}$
  Randomly choose and apply $f \in \{0, 0.1\}$
  for each step **do**
    if step $<$ step_max **then**
      $e_T = |y_{ref} - y(t)|$
      $e_R = |\psi_{ref} - \psi(t)|$
      $r = -\exp(e_T + e_R + \dot{e_T} + \dot{e_R})$
      if $r < -10$ **then**
        $r = -10$
      else if $r > -1$ **then**
        $r = 1$
      end if
    else
      End of episode
    end if
  end for
end for



Figure 4.11: Delayed feedback process.

For each action at time t, the environment produces a feedback at time $t + t_w$. This gives the DRL model longer-term feedback on the action taken, which increases the efficiency of the training. The CAM is updated with each action taken, *i.e.* at each time window t_w. As no action is taken before receiving the reward from the environment, this is a simplified form of delayed feedback (Howson, Pike-Burke, and Filippi 2023).

One time window t_w lasts 0.4 s and an episode ends at $step_{max} = 350$ (14 s), as this is the time needed by the hybrid controller to stabilise the behaviour of the drone subject to a fault.

The mean reward curve of the DRL model training in UUV Simulator is displayed in Figure 4.12. This figure illustrates the progression of the DRL model's mean reward over training as a function of the number of training steps. It can be observed that a minimum of $40,000$ steps was required to achieve convergence. The full training took approximately seven hours on a laptop with 32 GB of RAM, using one CPU core without parallelisation and no GPU acceleration.

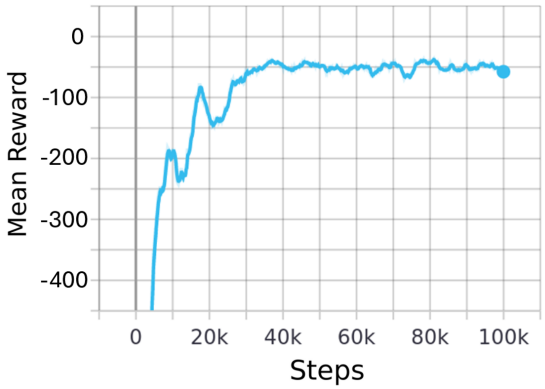


Figure 4.12: DRL model mean reward curve over number of steps.

After the training phase, a moving average low-pass filter is added to the output of the DRL model in order to attenuate variations in the CAM, with the objective of approaching global stability, as discussed in Section 3.2.2. This type of filter is chosen for its ease of tuning, as it only requires the selection of a fixed window size and does not involve parameter coupling. The transfer function of the selected moving average filter is given by:

$$H(z) = \frac{1}{N_W} \sum_{n=0}^{N_W-1} z^{-n} \tag{4.16}$$

where N_W denotes the window size of the moving average and z^{-n} represents a time shift. Low-pass filters introduce a phase delay, which can degrade closed-loop stability if not properly tuned (Franklin et al. 2002). To mitigate this issue, N_W must be carefully selected: it should be high enough to preserve the responsiveness of the control actions, while still sufficiently low to attenuate abrupt variations. Through experimental tuning, N_W was set to 15 in order to achieve a suitable compromise between smoothing the output of the DRL model and maintaining sufficient responsiveness to changes in the system.

In order to assess the impact of this low-pass filter on the DRL outputs, Figure 4.13 shows the temporal evolution of the weights $W_{11}, W_{12}, W_{13}, W_{14}$ defined in section 4.2.1 throughout a single episode after training, when $T0$'s efficiency level is at 0%.

It can be observed that the weights experience a peak of variation at the start of the episode, followed by diminishing fluctuations and a smoother curve progression after approximately 2 seconds. A possible explanation for the observed behaviour is that, as the policy is based on the state observation, there is a latency period while the DRL agent identifies the most effective action policy to adopt. This shows the benefit of using the low-pass filter, as it smooths the variations after the initial peak in the output of the DRL model.

Once the training phase has been completed, the resulting DRL model can be embedded within the controller to perform control reallocation during mission execution. At each step,

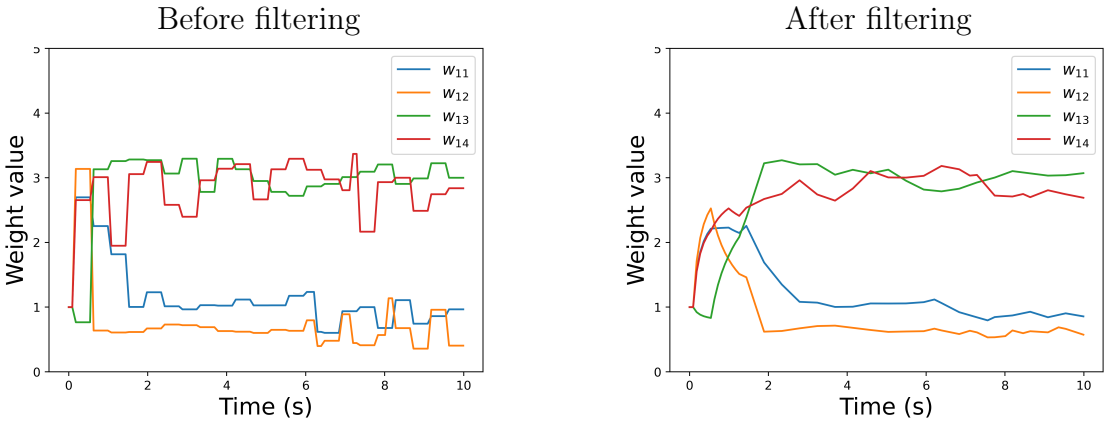


Figure 4.13: Evolution of CAM weights, where $T0$'s efficiency is at 0% during one entire episode, before (left) and after (right) application of the filter.

the DRL agent carries out the control reallocation, and the gain of the LQR controller is recomputed accordingly. Figure 4.14 illustrates the overall control architecture, after the training of the DRL model.

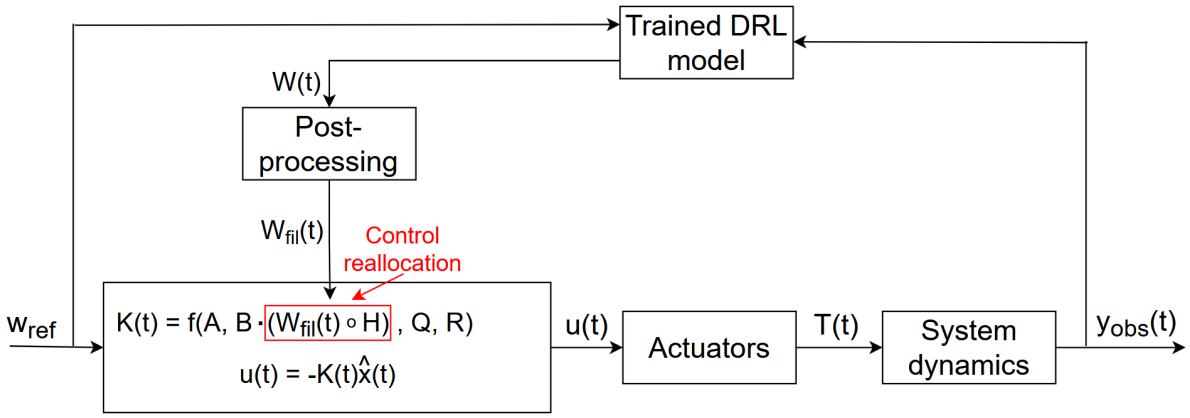


Figure 4.14: Embedding of the trained DRL model in the control architecture.

Now that the overall framework has been established, the chapter can be concluded with a summary of its main elements.

4.4 Conclusion

In this chapter, the configuration of the actuators and the specifications of the BlueROV2 Heavy have been described. The training environment for the DRL model in simulation was presented, highlighting the main challenges and the tools required for its implementation. The considered scenario was discussed, and the LQR controller was selected as the model-

based control strategy. Its implementation for the BlueROV2 Heavy was detailed, which involved modelling the vehicle and determining the LQR parameters. The integration of the DRL-based control reallocation method was also introduced, together with its embedding within the model-based controller.

The SAC algorithm was chosen for the DRL component, and preliminary tests showed that it is effective in handling thruster faults. The training process of the DRL model in our application case was described. Finally, the structure of the controller once the DRL model has been trained was presented. In this configuration, the DRL model adjusts the control allocation matrix in real time, and the gain of the LQR controller is recomputed at each step, allowing local stability of the closed-loop control. The next chapter presents the results obtained from the application of the proposed approach, applied to the actuator faults of a BlueROV2 Heavy, both in simulation and in real-world conditions.

Chapter 5

Simulation results and real-world experiments on an Autonomous Underwater Vehicle

The purpose of this chapter is to present the results obtained from the case study on thruster faults in a BlueROV2 Heavy. In section 5.1, we will discuss the evaluation methodology of the proposed hybrid controller and comparisons with other FTC approaches. Section 5.2 covers simulation results, while section 5.3 addresses results under real-world conditions.

5.1 Performance evaluation methodology

This section focuses on the evaluation process applied to thruster faults in a BlueROV2 Heavy. We will first outline the evaluation scenario for the proposed approach, followed by the considered fault cases. Then a list of evaluated controllers will be done in order to compare the proposed approach with different methods, and an identification of the steps for evaluation and comparison with other controllers will be made.

5.1.1 Evaluation scenario

The evaluation scenario involves tracking along the x-axis, starting from the initial point ($x = -1$ m, $y = 0$ m, $z = -2.5$ m, $\phi = 0$ rad, $\theta = 0$ rad, $\psi = 0$ rad), with a thruster fault introduced at $x = -0.4$ m as shown in Figure 5.1. Unlike simulation training, where the fault occurs at the start of the mission, in the evaluation the fault occurs shortly after the start of the mission, as the time required for the drone to stabilise in real conditions in the test tank must be taken into account. This position was selected because, in the

case of the physical drone, it allows sufficient time for the drone to stabilise in straight-line tracking before a fault occurs. As for the training, an episode ends at $step_{max} = 350$. The actuator fault affects a single thruster only, and occurs on one of the horizontal thrusters $(T0, T1, T2, T3)$. Regarding the reference vector, we choose $\delta = 1$ m (see Equation 4.5) to be consistent with the training of the DRL model.

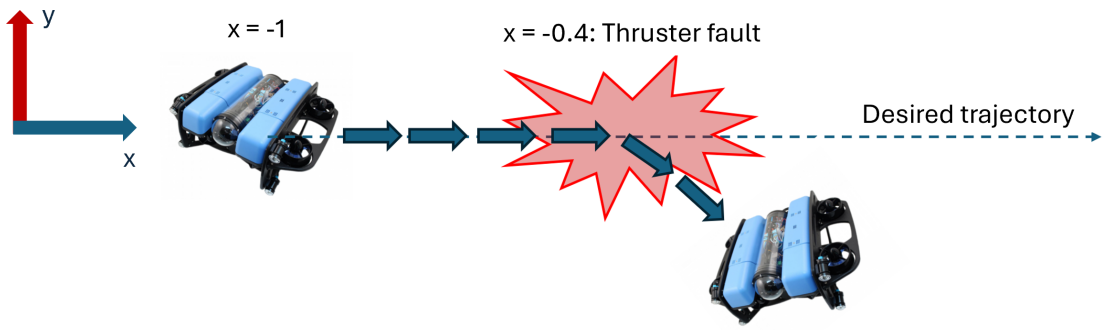


Figure 5.1: Trajectory of the BlueROV2 Heavy during the evaluation process.

The goal is to keep following a straight-line trajectory along the x-axis despite the occurence of a thruster fault. The next section focuses on the considered actuator faults occuring during this scenario.

5.1.2 Considered thruster faults

For the evaluation of recovery performance in simulation, we consider all possible horizontal thruster faults individually. The faults examined include 11 levels of efficiency loss for each horizontal thruster, defined as $f \in \{0, 0.1, 0.2, \ldots, 1\}$, where $f = 0$ corresponds to a total fault of a thruster. This exhaustive range of fault intensities allows us to assess the robustness of the controller across both known and novel fault scenarios. In particular, the values $f \in \{0, 0.1\}$ correspond to situations encountered during training, while $f \in \{0.2, \ldots, 1\}$ represent unseen levels of faults. Although the controller was not explicitly trained on these lower levels of fault severity, the resulting behaviour remains qualitatively similar. Evaluating the controller's response to these unfamiliar situations is essential for testing its generalisation capabilities, that is, its ability to extrapolate learned behaviours to fault conditions beyond those it was explicitly trained on. Such generalisation is critical for practical deployment, where the system may face a wide range of unanticipated fault levels.

The evaluation of each fault situation is carried out over a sufficiently large number of episodes to ensure statistically meaningful results. The exact number of episodes will be specified later, both for simulation and real-world experiments.

In order to evaluate the proposed approach, different controllers are used for comparison.

5.1.3 Compared controllers

To assess the performance of the proposed hybrid controller, introduced in Chapter 4 and hereafter referred to as the DRL-based LQR controller, we will not only evaluate it across various fault scenarios but also compare it with other FTC strategies.

In order to evaluate the impact of using a model-based controller framework, it is relevant to compare the DRL-based LQR controller with controllers whose gains are not calculated according to the system model. An example of this type of control is PID controllers whose gains are fixed and tuned using an experimental technique as described in section 2.1.1. To avoid lengthy terminology, in the following of this thesis, we will refer to PID controllers whose gains are not computed based on the system model simply as "PID-based controllers" to distinguish them from "LQR-based controllers."

Besides, to ensure a fair comparison with the proposed DRL-based LQR controller, which belongs to the class of adaptive controllers, we include comparisons with direct adaptive approaches for both PID-based and LQR-based controllers. Unlike the DRL-based strategy, this type of method does not rely on DRL but updates the controller gains using the MIT rule, as described in Section 2.2.2.1. The purpose of direct adaptation is to react quickly to actuator faults by increasing the weights associated with the errors between the desired and observed movements.

Direct adaptive methods are considered over indirect ones for comparison with the proposed approach because fault information is difficult to access in the considered application case. In our case, this difficulty arises from the number of actuators and their complex geometric configuration on the BlueROV2 Heavy. This is why, to compare our proposed controller, we employ existing approaches that do not depend on explicit fault diagnosis, namely direct adaptive methods.

The following is the list of controllers that will be evaluated. We distinguish between PID-based controllers and LQR-based controllers.

Evaluated PID-based controllers

- **Standard PID control:** This is a classical PID control approach as described in Equation 2.6, in which both the controller gains and the CAM remain fixed throughout the mission. This serves as a baseline for comparison with adaptive approaches. The PID gains used are the linear PID gains determined in C.-J. Wu (2018).
- **Conventional direct adaptive PID control:** This is a direct adaptive method, based on the MIT rule described in Section 2.2.2.1, applied to the gains of the PID

controller, which are adjusted without relying on a model of the system. In our case, only the proportional coefficient is updated. Regarding the tuning parameters of the MIT rule defined in section 2.2.2.1, they were determined by iteratively adjusting the tuning parameters to minimise the deviations without inducing oscillations in the system, in the worst fault case scenario that we consider, *i.e.* where one of the thusters is at 0% efficiency. We obtained $\alpha_y = 0.3$ and $\alpha_\psi = 0.02$.

- **DRL-based PID control:** In this method, the PID control law is the standard one described in Equation 2.6, with its output being the control input $\boldsymbol{\tau}$. This vector is then converted into actuator input via the CAM, which is updated by the DRL in the same way as in the DRL-based LQR controller. The difference here is that, unlike the LQR-based method, the CAM is not included in the system equations used for calculating the controller's gain at each step. Therefore, as shown in Figure 5.2, there are two control blocks: the control law, which is fixed, and the control allocation, which is adjusted by a trained DRL model.

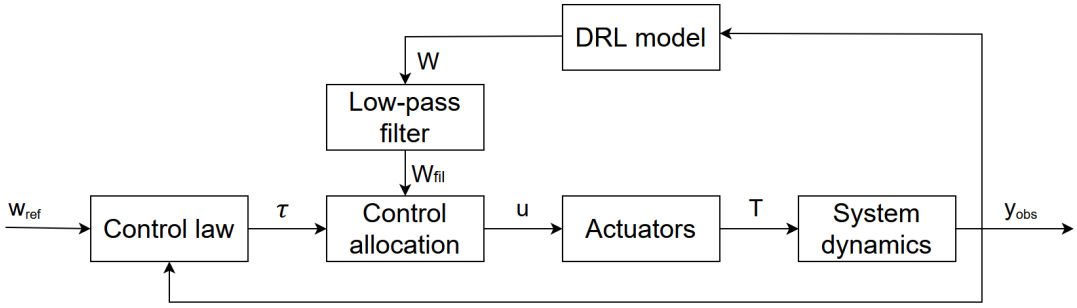


Figure 5.2: DRL-based PID control scheme.

As with the DRL-based LQR controller, a low-pass filter is applied to the output of the trained DRL model.

The training of the DRL model used for this controller is the same process as for the DRL-based LQR controller detailed in section 4.3.3, but using a PID controller with fixed gains. It is important to note that the DRL model used here differs from the one employed in the DRL-based LQR controller, as it is trained specifically for this PID control structure. In simulation the selected PID gains are the linear PID gains determined in C.-J. Wu (2018) for BlueROV2 Heavy. The main drawback compared to the proposed DRL-based LQR controller is the lack of stability guarantees, as the CAM is not considered in the computation of the control gain. Nevertheless, this approach remains relevant for comparison purposes, as it also relies on DRL.

Evaluated LQR-based controllers

- **Standard LQR control:** This involves applying the LQR controller described in Chapter 4, without the use of DRL and without any control reallocation. This serves as a baseline to highlight the benefits of incorporating DRL in case of thruster faults.
- **Conventional direct adaptive LQR control:** This method involves updating the values in the weighting matrix $\mathbf{Q}$ (described in section 4.2.2) associated with the y and yaw, which are the impacted directions in case of a fault, following the same principle as the MIT rule (see section 2.2.2.1). The higher the error in these directions, the greater the increase in the corresponding values in the matrix $\mathbf{Q}$, which in turn increases their weight in the cost function. The tuning parameters, which remain constant throughout the mission, were selected in the same way as for the direct adaptive PID control. The obtained tuning parameters are $\alpha_y = 0.5$ for the y error and $\alpha_\psi = 0.02$ for the yaw error.
- **DRL-based LQR control:** This is the proposed approach described in Chapter 4.

These controllers will be compared through two evaluation steps.

5.1.4 Evaluation steps

As shown in Figure 5.3, the evaluation process involves two steps.

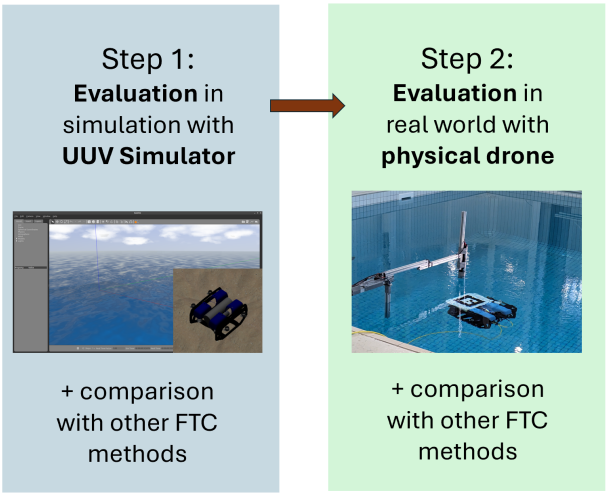


Figure 5.3: Steps of the evaluation process.

First, the trained DRL model is evaluated in the same simulation environment as for the training in order to verify whether it has properly learned the intended behaviour. The other controllers, used for comparison, are also evaluated in the same simulation environment.

Then, this trained DRL model is deployed on a physical BlueROV2 Heavy in a test tank. This step aims to evaluate the performance of the DRL model trained in simulation when directly applied (without any fine-tuning) to a real-world robotic platform. The other methods are also evaluated on the physical drone for comparison. The performance evaluation of the controllers, in simulation and real-world, is described below.

5.1.5 Performance measure

In the fault-free case, BlueROV2 Heavy reaches the target point without deviating from a straight line and with a yaw equal to zero. Thus, the performance evaluation of the hybrid controller in a fault situation is based on the observation of unwanted y error and yaw error. Let's define the performance indices for y and yaw as the integral of the error over a given number of time steps, where N represents the maximum step (end of an episode), and k corresponds to the step at which the error is detected:

$$\delta_y = \int_k^{k+N} |y(t)|\, dt, \quad \delta_{\text{yaw}} = \int_k^{k+N} |\psi(t)|\, dt \tag{5.1}$$

These indices provide a quantitative basis for comparing the controller's ability to mitigate deviations caused by actuator faults. All the mentioned controllers will be evaluated in the same manner.

For each fault scenario, each controller will be evaluated on a sample of 100 missions in simulation and 10 missions in real-world conditions. Statistical tests are also used to assess the significance of the results. Regarding the simulation results, the sample size is large enough to apply a Z-test for difference of means (Montgomery and Runger 2010). For the real-world evaluation, a Welch's t-test (Montgomery and Runger 2010) is employed, as it is appropriate for small sample sizes and unequal variances.

Now that we have outlined the evaluation process, the next section presents the results obtained through simulation.

5.2 Simulation results

The presentation of the simulation results, corresponding to step 1 of the evaluation process, is divided into the following sections: a first section evaluating the PID-based controllers; a second section evaluating the LQR-based controllers; a third section highlighting the benefits of using a DRL-based control reallocation method over a direct adaptive method, and a final section discussing the obtained results, including a comparison between the PID and LQR controllers.

5.2.1 PID-based controllers simulation results

The simulation results obtained with methods using a PID controller are presented here, including the standard PID, the conventional adaptive PID, and the DRL-based PID controllers.

For each thruster (among the four considered thruster) and each level of efficiency loss (among the eleven fault cases considered), the performance of the compared controllers is assessed over 100 episodes replaying the same fault scenario described in 5.1.1. Although the initial conditions and fault settings remain the same, the physics engine used in the UUV Simulator (based on GAZEBO) introduces slightly non-deterministic variations between episodes, due to internal noise in the simulation. As a result, repeating the same scenario multiple times is necessary to obtain statistically meaningful average performance indices. For the evaluation in simulation, the fault detection (*i.e.*, the detection of abnormal behaviour) is immediate as the state of the simulated system is known and accurate.

Fault recovery in the horizontal thrusters were evaluated separately, and the average of their results was taken for each efficiency level. Table 5.1 shows the mean and standard deviation of y and yaw performance indices for thruster faults from 0% to 100% efficiency, comparing the three evaluated PID-based controllers. This allows to evaluate severe faults (0% and 10% efficiency) encountered during training, as well as less severe faults (above 10% efficiency) that were not seen during training.

For each fault efficiency level and performance index, the results of the evaluated controllers are compared in this table, with the best performance highlighted in green and the second-best in orange. This colour scheme will be used consistently throughout this chapter.

The tabulated results indicate that at 0% thruster efficiency, the DRL-based PID controller led to a reduction in y error of 86.1% and 81.7% compared to the standard and adaptive PID controllers, respectively. For the yaw error, reductions of 57.4% and 54.2% were observed. At 10% efficiency, performance gains remained substantial, with reductions of 87.3% and 75.6% in y error, and 60.4% and 53.3% in yaw error relative to the same reference controllers. This DRL-based PID controller's superior performance can be attributed to the DRL-based method's effective control reallocation policy learned during training.

It can also be observed in Table 5.1 that, although the direct adaptive controller performs better than the standard PID controller, it still generates nearly as much unwanted rotation. This can be explained by the fact that the direct adaptive method does not redistribute the desired forces among the actuators, nor does it take the geometric configuration of the actuators into account. Instead, the conventional adaptive PID control leads certain thrusters to reach saturation.

The DRL model was also able to generalise effectively, outperforming other controllers

	0%		10%		20%		30%	
	δ_y	δ_{yaw}	δ_y	δ_{yaw}	δ_y	δ_{yaw}	δ_y	δ_{yaw}
PID	70.87 ± 0.02	48.40 ± 0.05	65.24 ± 0.10	47.47 ± 0.05	41.50 ± 0.02	43.17 ± 0.03	33.85 ± 0.01	41.47 ± 0.03
Adaptive PID	53.93 ± 0.13	45.12 ± 0.21	33.96 ± 0.77	40.26 ± 0.43	25.73 ± 0.91	38.93 ± 0.68	18.64 ± 0.18	34.62 ± 0.66
PID/DRL	9.86 ± 0.39	20.64 ± 0.18	8.28 ± 0.54	18.79 ± 0.32	7.36 ± 0.71	16.24 ± 0.53	6.25 ± 0.02	15.33 ± 0.76

	40%		50%		60%		70%	
	δ_y	δ_{yaw}	δ_y	δ_{yaw}	δ_y	δ_{yaw}	δ_y	δ_{yaw}
PID	21.46 ± 0.07	37.93 ± 0.05	13.39 ± 0.03	35.37 ± 0.09	9.72 ± 0.01	33.90 ± 0.04	3.96 ± 0.03	22.64 ± 0.03
Adaptive PID	13.04 ± 0.33	26.80 ± 0.06	8.83 ± 0.50	22.90 ± 0.21	3.46 ± 0.71	18.36 ± 0.41	2.29 ± 0.93	15.38 ± 0.60
PID/DRL	5.97 ± 0.13	14.46 ± 0.97	5.03 ± 0.32	11.30 ± 0.13	4.28 ± 0.54	9.25 ± 0.37	3.01 ± 0.73	8.74 ± 0.55

	80%		90%		100%	
	δ_y	δ_{yaw}	δ_y	δ_{yaw}	δ_y	δ_{yaw}
PID	1.73 ± 0.72	15.72 ± 0.63	1.02 ± 0.07	9.09 ± 0.05	0.00 ± 0.00	0.00 ± 0.00
Adaptive PID	0.37 ± 0.14	5.80 ± 0.08	0.07 ± 0.13	4.37 ± 0.10	0.00 ± 0.00	0.00 ± 0.00
PID/DRL	2.12 ± 0.92	6.36 ± 0.73	2.03 ± 0.11	5.03 ± 0.09	1.80 ± 0.01	2.50 ± 0.00

Table 5.1: Mean and standard deviation of performance indices for the 4 thrusters evaluated individually from 0% (total fault) to 100% efficiency with PID, conventional adaptive PID and DRL-based PID (referred to as PID/DRL in the table) in simulation.

at 20% to 50% efficiency levels as shown in Table 5.1. These levels were not applied during training, showing the generalisation capability of DRL. Beyond 60% efficiency, conventional adaptive PID control outperforms DRL-based PID control in reducing the y error, possibly because, at low fault levels, increased gains alone may be sufficient to maintain performance. Beyond 80% efficiency, the DRL-based PID controller underperforms, which may be due to a significant discrepancy from the training scenarios, where the actuator faults were more severe.

To assess whether the differences observed between two sets of results are statistically significant, the Z-test for difference of means (Montgomery and Runger 2010) is employed. This method evaluates whether the means of two independent samples differ significantly by taking into account their respective standard deviations and sample sizes. The test is performed by computing a test statistic, which measures how far the observed difference is from the null hypothesis. The null hypothesis states that the means of the two populations are equal. A p-value is derived, representing the probability of obtaining such a result if the

null hypothesis were true. If the p-value is below a predefined significance level, commonly $\alpha = 0.05$, the null hypothesis is rejected.

In this context, μ_1 and μ_2 represent the means of the two independent samples, while σ_1 and σ_2 denote their respective standard deviations. Additionally, n_1 and n_2 represent the sizes of the two independent samples being compared. The Z-test is appropriate here due to the large sample sizes. The procedure is detailed in Algorithm 4.

Algorithm 4 Z-test for Difference of Means (with p-value)

function COMPUTE_P_VALUE($\mu_1, \sigma_1, n_1, \mu_2, \sigma_2, n_2$)
  $z_0 = \frac{(\mu_1 - \mu_2)}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}$
  $p = 2 \cdot (1 - \Phi(|z_0|))$ ▷ Φ is the standard normal cumulative distribution function.
  $\alpha = 0.05$
  if $p < \alpha$ **then**
    return (p, True) ▷ Significant difference
  else
    return (p, False) ▷ No significant difference
  end if
end function

This test is applied to the results presented in Table 5.1, to provide a comparison, on the one hand, between the DRL-based PID controller and the standard PID controller, and on the other hand, between the DRL-based PID controller and the adaptive controller. All computed p-values are lower than 10^{-15}, highlighting the high statistical significance of the results.

Now that the results obtained with the PID controller have been presented, the focus shifts to those achieved using LQR controller, which include the proposed approach. This comparison will serve to highlight the differences between non–model-based controllers and model-based controllers, as it will be discussed in the simulation results discussion section.

5.2.2 LQR-based controllers simulation results

The evaluation of the trained DRL model is also performed in UUV Simulator on the standard LQR, conventional adaptive LQR, and DRL-based LQR controllers, considering thrusters faults between 0% and 100% efficiency. Table 5.2 reports the mean and standard deviation of the y and yaw performance indices, computed over 100 episodes for each horizontal thruster and each efficiency level, and then averaged across all four thrusters as for the PID-based controllers. This table allows the comparison of the three evaluated LQR-based controllers.

According to the table, it can be seen that at 0% thruster efficiency, the DRL-based LQR controller reduced the y error by approximately 73.9% and 69.3% compared to the

	0%		10%		20%		30%	
	δ_y	δ_{yaw}	δ_y	δ_{yaw}	δ_y	δ_{yaw}	δ_y	δ_{yaw}
Standard LQR	33.58 ± 0.09	39.42 ± 0.12	30.68 ± 0.05	36.12 ± 0.10	28.54 ± 0.05	33.76 ± 0.09	26.24 ± 0.06	29.82 ± 0.08
Adaptive LQR	28.56 ± 0.13	30.17 ± 0.31	26.34 ± 0.15	28.10 ± 0.26	23.61 ± 0.11	26.09 ± 0.25	20.09 ± 0.09	22.51 ± 0.19
LQR/DRL	8.77 ± 0.06	16.77 ± 0.14	7.61 ± 0.05	15.10 ± 0.13	7.02 ± 0.03	13.92 ± 0.10	6.88 ± 0.03	13.08 ± 0.11

	40%		50%		60%		70%	
	δ_y	δ_{yaw}	δ_y	δ_{yaw}	δ_y	δ_{yaw}	δ_y	δ_{yaw}
Standard LQR	25.06 ± 0.04	27.87 ± 0.07	19.33 ± 0.03	23.08 ± 0.06	14.54 ± 0.02	20.80 ± 0.04	11.60 ± 0.03	17.50 ± 0.04
Adaptive LQR	18.86 ± 0.06	21.24 ± 0.15	16.83 ± 0.02	17.21 ± 0.16	12.93 ± 0.03	14.27 ± 0.11	10.11 ± 0.02	11.90 ± 0.08
LQR/DRL	5.78 ± 0.01	12.11 ± 0.08	5.37 ± 0.01	11.56 ± 0.05	5.04 ± 0.01	11.04 ± 0.02	4.85 ± 0.01	10.54 ± 0.01

	80%		90%		100%	
	δ_y	δ_{yaw}	δ_y	δ_{yaw}	δ_y	δ_{yaw}
Standard LQR	7.37 ± 0.02	10.62 ± 0.02	4.19 ± 0.01	6.25 ± 0.01	0.00 ± 0.00	0.00 ± 0.00
Adaptive LQR	7.21 ± 0.02	8.50 ± 0.08	3.51 ± 0.02	5.06 ± 0.05	0.00 ± 0.00	0.00 ± 0.00
LQR/DRL	2.69 ± 0.01	9.19 ± 0.01	2.31 ± 0.01	6.46 ± 0.01	0.83 ± 0.00	3.17 ± 0.01

Table 5.2: Mean and standard deviation of performance indices for the 4 thrusters evaluated individually from 0% (total fault) to 100% efficiency with standard LQR, conventional adaptive LQR and DRL-based LQR controllers, evaluated over 100 episodes in simulation for each case.

standard LQR and the adaptive LQR controllers, respectively. In terms of yaw error, reductions of 57.5% and 44.4% were achieved. Similarly, at 10% efficiency, the DRL-based LQR controller achieved reductions of 75.2% and 71.1% in y error, and 58.2% and 46.3% in yaw error compared to the same benchmarks.

The enhanced performance of the DRL-based LQR controller compared to the standard and adaptive LQR controllers is likely due to the control reallocation strategy learned by the DRL agent during training. As with the adaptive PID controller, the lower performance of the conventional adaptive LQR can be attributed to its lack of consideration for the thruster geometric configuration, as it only adjusts the relative importance of the state error.

The DRL-based LQR controller also outperformed other controllers at 20% to 70% efficiency levels which were not applied during training, showing the generalisation capability of the DRL model. Beyond 70% efficiency, conventional adaptive LQR control outperforms DRL-based LQR control in reducing the yaw error, possibly because gain adjustment alone is sufficient to compensate for less severe faults. Beyond 90% efficiency, DRL-based LQR

control underperforms, which may be due to divergence from situations encountered during training. As with the results obtained for the PID controller, all computed p-values are lower than 10^{-15}, indicating a high level of statistical significance.

In addition to the performance metrics, it is relevant to analyse the behaviour of the thrusters outputs. In order to better understand the differences between using a direct adaptive controller and an indirect adaptive controller based on control reallocation using DRL, Figure 5.4 shows the time evolution of thruster forces for $T0$ at 0% efficiency, comparing a conventional direct adaptive controller and a DRL-based controller.

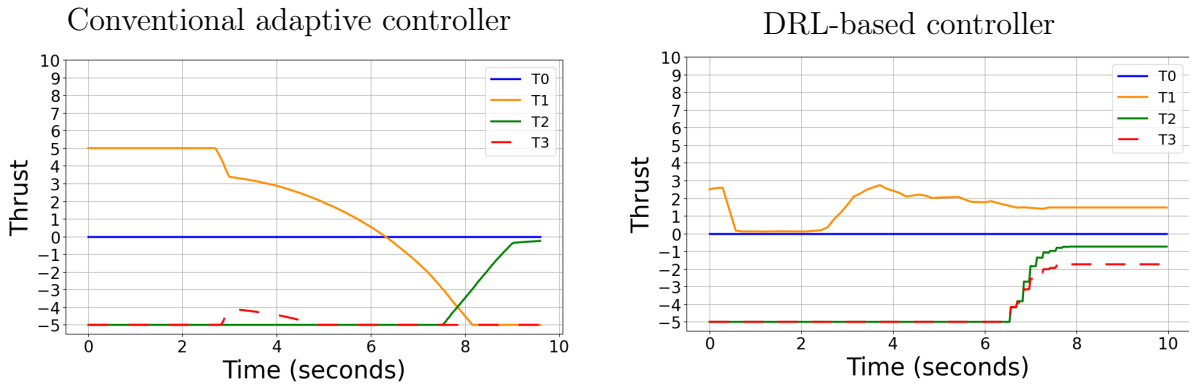


Figure 5.4: Time evolution of thruster outputs in case of $T0$ at 0% efficiency with the conventional adaptive controller (left) and the DRL-based controller (right).

While the conventional adaptive controller leads to thruster saturation for $T1$ and $T3$ which is likely due to increased gains — when attempting to compensate for actuator faults — the control reallocation mechanism employed by the DRL-based controller dynamically adjusts the distribution of control efforts among the remaining actuators. This enables the system to achieve the desired motion without overloading individual thrusters, thereby preventing saturation and ensuring more efficient use of available actuation resources.

Overall, the results confirm that the DRL-based LQR controller achieves significant reductions in both y and yaw errors compared to the standard and adaptive LQR controllers, and also demonstrates a more efficient use of the available actuation by preventing thruster saturation through dynamic control reallocation. We now turn to a discussion of the simulation results in the following section with a comparison between the PID-based and LQR-based controllers.

5.2.3 Simulation results discussion

We observed that for severe faults, close to those encountered during the DRL model training, the DRL-based controller significantly outperforms the other methods. The strong performance of the DRL-based controller can be explained by its ability to learn an effective

control reallocation mechanism during training, allowing it to adapt dynamically to fault conditions. However, there is room for improvement, as it is theoretically possible to achieve zero error in y and yaw, given that the system remains sufficiently actuated to maintain control over the forces applied on each DoF.

The percentage improvements in performance observed for the DRL-based controllers are more pronounced in the case of PID controllers than for LQR-based ones. However, the overall performance indices are globally better for all LQR-based controllers, whether standard, adaptive, or DRL-based. The optimised gains inherent to LQR design contribute to enhanced baseline performance and ensure local stability, which is not guaranteed with PID controllers.

For moderate-intensity faults, which were not seen during training but are similar to the drone's behaviour observed in training, the DRL-based controller remains the most effective due to DRL's generalisation capabilities. We also observed the relevance of control reallocation in comparison to a direct adaptive approach in the presence of severe faults, highlighting the benefit of redistributing control forces rather than directly adjusting controller gains.

However, for lower-intensity faults, which differ greatly from the behaviours seen during training, the effectiveness of the DRL-based controller decreases compared to the other controllers. The conventional adaptive PID controller outperforms the DRL-based PID controller in terms of y error at 60% thruster efficiency. For the LQR-based controllers, the adaptive version surpasses the DRL-based one in yaw error from 80% thruster efficiency. These results suggest that the DRL-based LQR controller remains effective across a wider fault range than the DRL-based PID. This may be attributed to the LQR's optimal control structure into the control input, which could make the learning process more robust to fault variation and facilitate policy generalisation within the DRL framework.

The simulation results suggest a fault detection threshold, experimentally determined at $y = \pm 0.1$ m, beyond which a DRL-based controller is preferable to a standard controller or a conventional adaptive controller. This stems from the fact that the training focused on the most severe faults, relying on the generalisation capability of DRL to handle moderate faults, while in the fault-free case the choice was made to use a standard optimal control without DRL, as explained in section 3.1.1. This fault detection threshold will be used for the experiments on physical BlueROV2 Heavy. The same fault detection threshold will be used for evaluating the direct adaptive methods to ensure comparability with the DRL-based methods.

The next section details the results obtained on the physical BlueROV2 Heavy, which corresponds to step 2 of the evaluation process.

5.3 Experiments with a physical BlueROV2 Heavy

This section focuses on the sim-to-real application process of the proposed approach which relies on the DRL model trained in the UUV Simulator, and presents the experimental results obtained using a physical BlueROV2 Heavy. The section begins with a preliminary evaluation of the proposed approach in a different simulation environment, in order to assess the ability of the DRL controller to adapt to different hydrodynamic parameters. This serves as a first step towards deployment on the physical vehicle, whose operating conditions differ even more significantly from the original training environment.

This is followed by a presentation of the experimental setup. The next parts present the results obtained using control methods based on PID and LQR control, respectively. The section concludes with a discussion of the experimental results for all the compared controllers.

5.3.1 Preliminary evaluation of the proposed approach in a different simulated environment

To assess the ability of the proposed approach to adapt to changes in the underwater vehicle model and hydrodynamic parameters, the DRL model trained on the UUV Simulator was directly transferred to the LOTUSim simulator (Lechene et al. 2024). This step helps to determine whether transferring the approach to real underwater drones is feasible, since testing on two simulators with different dynamics provides a closer representation of the variability encountered in real-world conditions.

Unlike the UUV Simulator, LOTUSim uses a different BlueROV2 configuration and updated modelling tools, resulting in differences in the vehicle dynamics. This transfer aimed to validate the DRL model's adaptability across two simulators with differing environmental dynamics and models of the BlueROV2 Heavy. LOTUSim runs in parallel with Unity (Buyuksalih et al. 2017) to visualise the trajectories of the BlueROV2 Heavy, and Figure 5.5 illustrates an example of a thruster-fault trajectory obtained using LOTUSim and Unity.

In this figure, the green flag denotes the waypoint, positioned on the straight-line trajectory along the x-axis. The red curve shows the drone's trajectory under the standard LQR controller, while the blue curve represents the trajectory under the DRL-based LQR controller. The figure also shows two views of the BlueROV2 Heavy. The considered scenario involves a fault situation in which thruster $T0$ operates at a level of efficiency of 10%, corresponding to a fault coefficient $f = 0.1$.

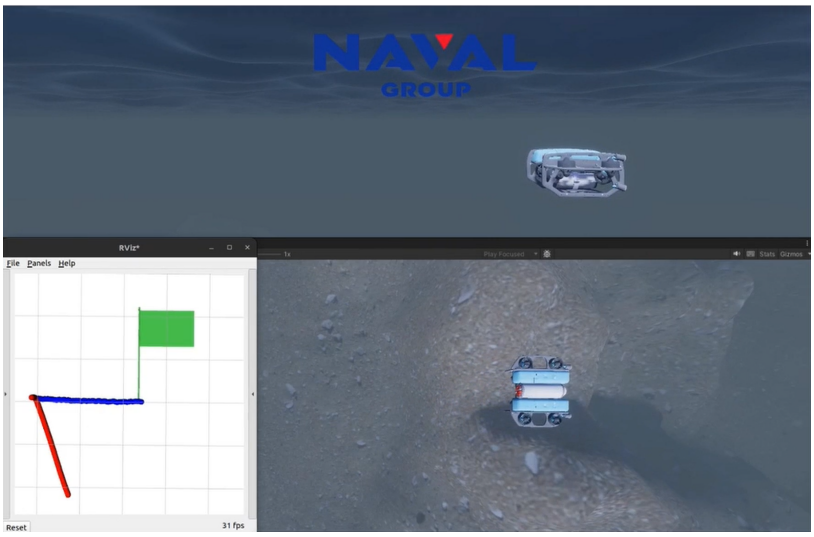


Figure 5.5: Trajectory of BlueROV2 Heavy with Unity. Thruster $T0$ is at 10% efficiency. In red: BlueROV2 Heavy trajectory using the standard LQR controller, and in blue: using the DRL-based LQR controller.

It can be observed that the trajectory of the BlueROV2 Heavy under the DRL-based LQR controller remains closely aligned with the reference straight line, whereas the trajectory under the standard LQR controller exhibits a visibly larger deviation.

This demonstrates that the force redistribution learned by the DRL component remained effective when transferred to a different simulation environment, despite discrepancies in the underlying dynamics and BlueROV2 Heavy modelling. This result highlights the adaptation capabilities of the DRL-based control strategy, and represents a first step towards assessing its potential for deployment on the real robot. The next section focuses on the experimental setup used for this deployment.

5.3.2 Experimental setup

This section presents the experimental framework used to validate the proposed approach in the real world. It first describes the tools employed for conducting the physical experiments, followed by the protocol and data collection process used during testing on the BlueROV2 Heavy.

Tools for real-world experiments

To validate the proposed approach under real-world conditions, a BlueROV2 Heavy was used in the Flinders University test pool. The localisation of the drone was determined using an ArUco marker tracker (Garrido-Jurado et al. 2014) and a depth sensor. The ArUco marker was tracked by a camera positioned slightly below the surface as shown in Figure

5.6, and its data were merged with a Kalman filter incorporating measurements from the IMU and the depth sensor.

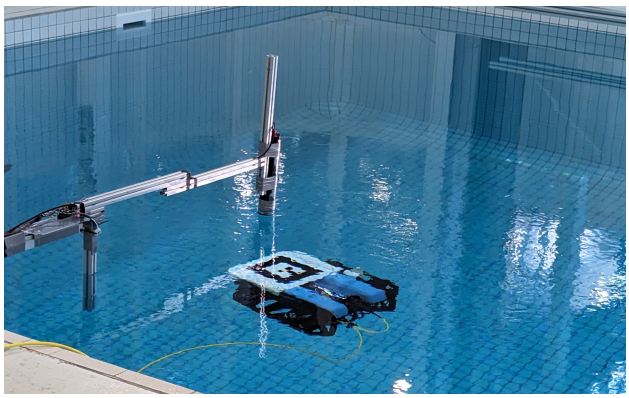


Figure 5.6: BlueROV2 Heavy in the test pool, with the localisation system.

The sensor data were transmitted via a tether to a laptop with 32 GB of RAM, where the controller outputs were also calculated. The DRL model, trained in simulation, was deployed on the real drone without any fine-tuning or additional online learning. Figure 5.7 shows the experimental setup used to apply the DRL-based fault-tolerant control strategy to the physical BlueROV2 Heavy platform.

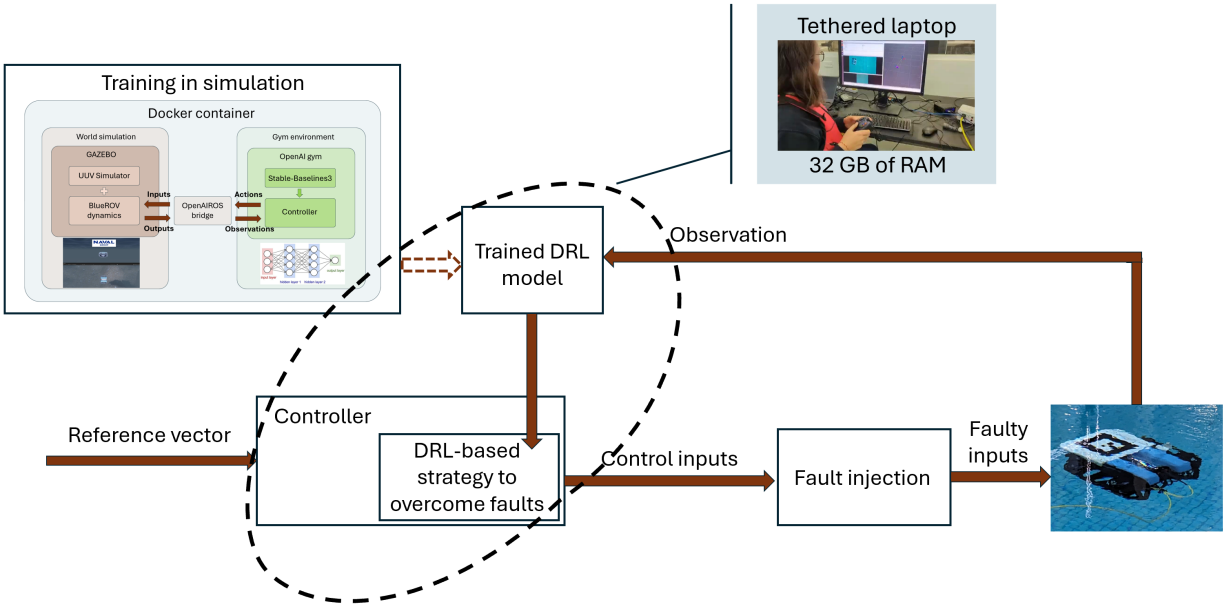


Figure 5.7: Diagram illustrating the implementation of DRL-based FTC strategies on the physical BlueROV2 Heavy.

As for the simulation part, the fault injection consists of applying a proportional coefficient of efficiency reduction to the faulty thruster, f (defined in section 3.1.2). Regarding the inputs of the physical BlueROV2 Heavy and the thruster fault injection, a technical issue

arose. The software of the BlueROV2 Heavy is not designed to accept direct actuator input commands; it is instead configured to receive a single six-dimensional vector of desired forces and torques. However, the DRL-based PID controller and the DRL-based LQR controller controllers generate actuator inputs directly, since the force allocation process is modified by the DRL method. To address this incompatibility, the controller outputs were reconverted into forces and torques using the original (unmodified) CAM as shown in Figure 5.8 where u' and τ' are the faulty actuator and control inputs as defined in section 3.1.2.

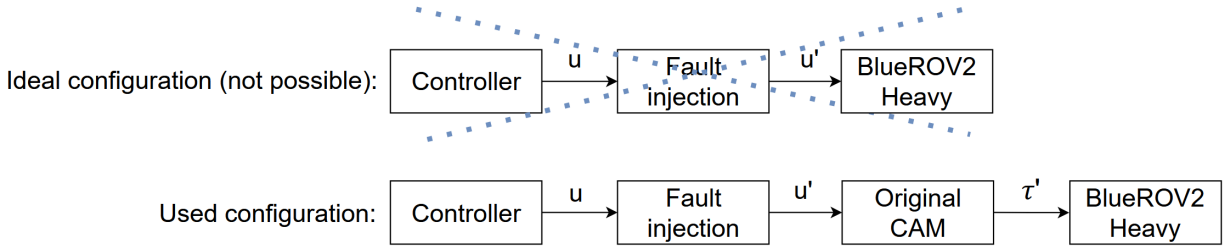


Figure 5.8: Ideal configuration and used configuration to inject the input commands to the physical drone.

While this reconversion introduces a loss of information—reducing an eight-dimensional actuator input vector to a six-dimensional wrench vector—the resulting behaviour of the vehicle remains consistent with what would have been observed, had the original actuator commands been applied directly.

Figure 5.9 shows the tools used to visualise the trajectory of the drone during the experiments. On the right side of the screenshot, the 3D position of the BlueROV2 Heavy is tracked using RViz, a 3D visualisation tool for ROS that enables real-time rendering of sensor data and robot states (Quigley et al. 2009). On the left side, the view from the external camera tracking the ArUco marker on the drone is displayed, alongside the video feed from the onboard camera of the drone. The field of view of the external camera enables observation of the drone's trajectory over approximately two metres, as illustrated in the figure.

Real-World Testing Protocol and Data Collection

Each episode consists of a scenario instance involving the tracking of a straight line and the occurrence of a thruster fault at $x = -0.4$ m, as described in section 5.1.1. The sensor data and control inputs were saved in a rosbag (Quigley et al. 2009) for each episode. A rosbag is a file format used to record and store data exchanged between different nodes in a ROS-based system. A step during an episode lasted 0.048 s, which corresponds to the reception frequency of the sensor data.

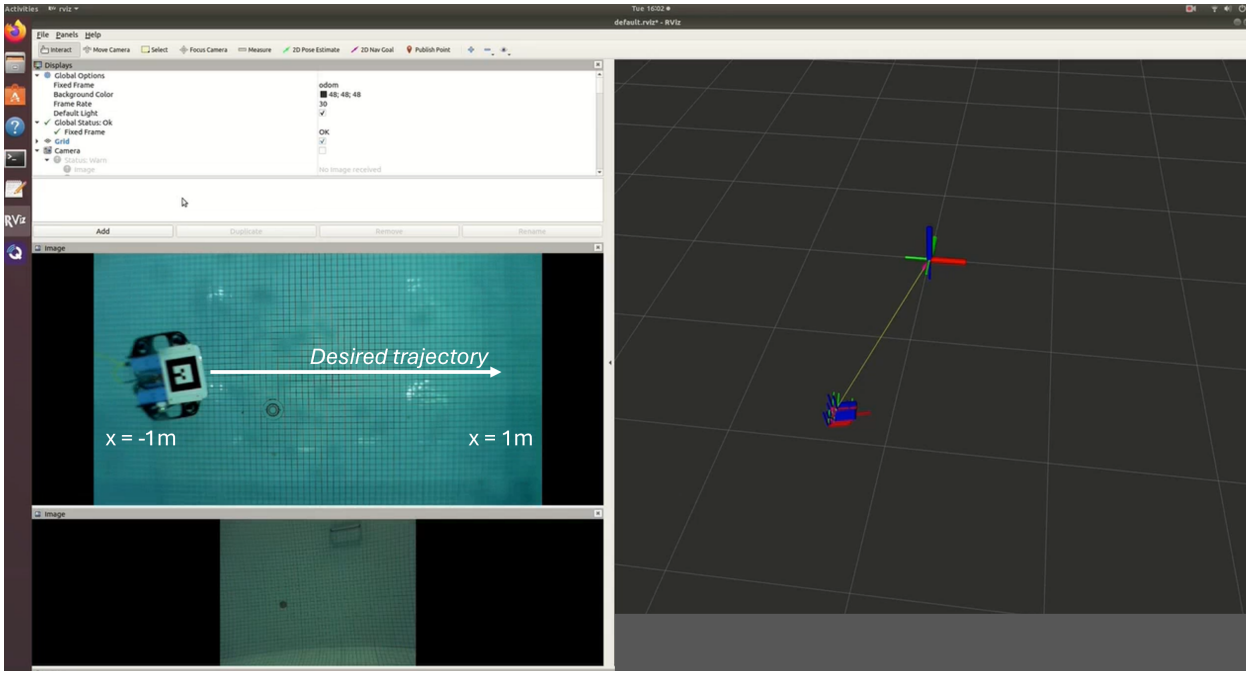


Figure 5.9: Screenshot of the tools used to visualise the drone's trajectory during the experiments.

In terms of real-world evaluation, assessing all scenarios, that is the eleven loss of efficiency cases for each of the four horizontal thrusters, would require a significant amount of time. Therefore, specific levels of thruster efficiency were selected for experiments on physical BlueROV2 Heavy: 0%, 40%, and 70%, corresponding to $f \in \{0, 0.4, 0.7\}$. This selection allows for the evaluation of a severe fault, a moderately severe fault, and a mild fault for each thruster individually, while staying within the range where the DRL model can still be efficient. The fault detection threshold was set at $y = \pm 0.1$ m, as determined in the simulation results section 5.2.3.

Before each fault scenario, a stabilisation phase (without any thruster fault) is carried out to ensure that the BlueROV2 Heavy is sufficiently close to the desired initial state ($x = -1m,\ y = 0m,\ z = -2.5m,\ \phi = 0$ rad, $\theta = 0$ rad, $\psi = 0$ rad). The pseudocode corresponding to this stabilisation step is given in Algorithm 5 where the error variable corresponds to the sum of the absolute errors on x, y, z, ϕ, θ, and ψ. The stabilisation threshold of 0.1 was determined experimentally and enabled a stable launch of the drone.

At the end of an episode, termination is triggered when the drone approaches the edge of the camera's field of view, with a 10 cm safety margin. After an episode ends, the robot is immediately repositioned at the start of the trajectory, and a new stabilisation phase is initiated before starting the next episode.

To assess the number of episodes required for an accurate estimation of the results, we used the proposed DRL-based LQR control approach over 20 episodes. During this, a

Algorithm 5 Initial stabilisation procedure

```
Set reference state: x = −1, y = 0, z = −2.5, φ = 0, θ = 0, ψ = 0
while not stabilised do
    Compute control input using current controller
    Apply control input to the BlueROV2 Heavy
    Measure current state
    if error < 0.1 then
        stabilised = True
    end if
end while
```

total fault was applied to one of the thrusters, and the mean y error, which serves as a performance measure, along with the standard deviation, were recorded over an increasing number of episodes up to 20. The results can be seen in Figure 5.10, where we can see that the performance converges after 10 episodes. Therefore, we chose to evaluate each fault case over 10 episodes on the real robot, with each set of 10 runs taking approximately 30 minutes.

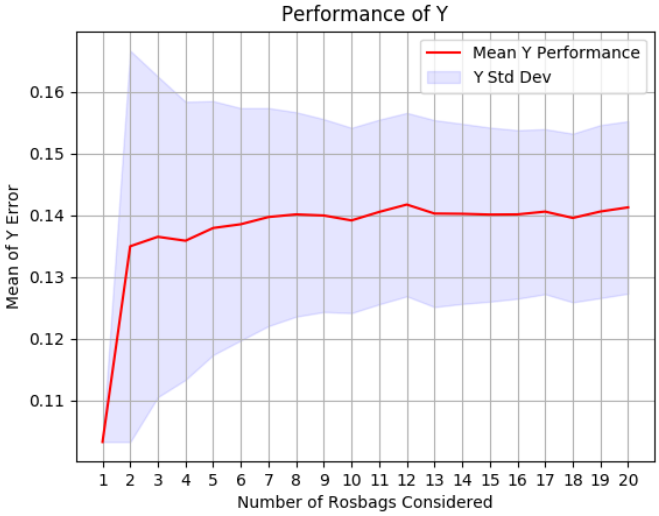


Figure 5.10: Mean of y error, with standard deviation across the number of episodes (or rosbags) considered.

The next section presents the results obtained for each fault case using the considered PID-based controllers.

5.3.3 PID-based controllers experimental results

The evaluated controllers include the standard PID, conventional adaptive PID, and DRL-based PID controllers. The conventional adaptive PID and the DRL-based PID controllers were activated upon fault detection (with a threshold set at $y = \pm 0.1$m). In

real-world conditions, it was necessary to fine-tune the PID gain related to the y direction, which was accomplished using the Ziegler-Nichols method (section 2.1.1). The new gain related to the y direction is $\mathbf{K_y} = (0.25, 2.5, 0.2)$.

As faults in $T0$ and $T2$ thrusters induced the most asymmetric behaviour, the evaluated scenario for application to the real BlueROV2 Hevay involved faults of the $T0$ or $T2$ thruster operating at 0%, 40% or 70% efficiency. The performances of the standard PID, conventional adaptive PID and DRL-based PID controllers are shown in Table 5.3.

	$T0$: 0%		T2: 0%	
	δ_y	δ_{yaw}	δ_y	δ_{yaw}
PID	70.26 ± 1.72	78.22 ± 3.26	78.18 ± 0.45	75.01 ± 1.49
Adaptive PID	31.18 ± 2.43	32.50 ± 1.86	54.28 ± 1.13	44.37 ± 1.10
PID/DRL	9.33 ± 0.63	29.10 ± 0.26	6.80 ± 0.66	38.84 ± 4.90

	$T0$: 40%		T2: 40%	
	δ_y	δ_{yaw}	δ_y	δ_{yaw}
PID	26.27 ± 0.43	82.09 ± 0.33	64.15 ± 1.15	67.11 ± 3.86
Adaptive PID	15.84 ± 0.34	26.30 ± 0.25	11.18 ± 1.06	54.37 ± 1.10
PID/DRL	4.57 ± 0.95	16.23 ± 0.28	4.46 ± 0.66	43.94 ± 1.49

	$T0$: 70%		T2: 70%	
	δ_y	δ_{yaw}	δ_y	δ_{yaw}
PID	13.26 ± 0.72	67.38 ± 1.63	14.02 ± 0.43	57.03 ± 1.54
Adaptive PID	5.68 ± 0.93	25.94 ± 0.30	5.86 ± 2.71	51.37 ± 1.13
PID/DRL	1.74 ± 0.38	17.76 ± 0.22	2.03 ± 1.11	36.66 ± 2.79

Table 5.3: Performance indices for $T0$ and $T2$ at 0%, 40% and 70% efficiency with standard PID, conventional adaptive PID, and DRL-based PID controllers on a physical BlueROV2 Heavy, over 10 evaluation tests.

It can be observed that the performance of the DRL-based PID controller is marked in green across all cases, indicating that it significantly outperforms both the standard and the conventional adaptive PID controllers in minimising the error in all the considered scenarios. At 0% $T0$ efficiency, the DRL-based PID controller achieved a reduction in y error of 86.7% compared to the standard PID controller and 70.1% compared to the adaptive PID controller. In terms of yaw error, the improvements reached 62.8% and 10.5%, respectively. At 40% $T0$ efficiency, performance gains remained significant, with reductions in y error of 82.6% and 71.1% relative to the standard and adaptive controllers, respectively. The yaw error was also reduced by 80.2% and 38.3%.

Regarding $T2$, at 0% efficiency, the DRL-based PID controller reduced the y error by 91.3% and 87.5% compared to the standard and adaptive PID controllers, respectively.

For the yaw error, improvements of 48.2% and 12.5% were achieved. At 40% efficiency, the controller continued to outperform the baselines, with reductions in y error of 93.0% and 60.1%, and in yaw error of 34.5% and 19.2% relative to the standard and adaptive controllers, respectively.

These results demonstrate that the DRL-based PID controller has been successfully applied to the real robot, as it provides a significant improvement in terms of performance compared to the other controllers. This is due to its training in simulation across a variety of fault scenarios and its strong capacity for adaptation.

Unlike simulation, the results of conventional adaptive PID controller are inferior to the results of DRL-based PID controller at 70% efficiency. Furthermore, the DRL-based PID controller shows an overall better performance in y but worse performance in yaw compared to the simulation. These differences may be due to the approximation of the drone model in simulation, to the use of a fault detection threshold, and to the fact that the thruster inputs are transformed into forces as explained in the section 5.3.2.

To assess whether the observed performance differences between controllers on the physical drone are statistically significant, Welch's t-test (Montgomery and Runger 2010), a variation of Student's t-test (Montgomery and Runger 2010), is employed. It evaluates whether the means of the two samples differ significantly without assuming homogeneity of variance. From a test statistic, described in Algorithm 6 with the same notations as for the Z-test, a p-value is obtained. In this study, the normal probability plot was examined, indicating that there is no issue with the normality assumption, which supports the use of Welch's test.

Tables 5.4 and 5.5 report the p-values obtained from the experimental results.

	DRL-based PID/Standard	DRL-based PID/Adaptive PID
$T0$: 0%	<1e-16	9.28e-11
$T0$: 40%	2.22e-16	1.12e-12
$T0$: 70%	1.33e-15	8.28e-08
$T2$: 0%	<1e-16	<1e-16
$T2$: 40%	<1e-16	3.22e-11
$T2$: 70%	3.47e-12	1.65e-03

Table 5.4: p-value computation for y error across efficiency levels from 0% to 70%, using physical drone data.

	DRL-based PID/Standard	DRL-based PID/Adaptive PID
$T0$: 0%	4.07e-12	2.8e-04
$T0$: 40%	<1e-16	<1e-16
$T0$: 70%	7.77e-15	<1e-16
$T2$: 0%	7.27e-10	6.9e-03
$T2$: 40%	1.96e-09	<1e-16
$T2$: 70%	9.31e-12	8.32e-09

Table 5.5: p-value computation for yaw error across efficiency levels from 0% to 70%, using physical drone data.

As observed in the simulation studies, the very small p-values confirm that the performance differences between the DRL-based PID controller and the other controllers are statistically significant.

Algorithm 6 Welch's t-test for Difference of Means

function WELCH_T_TEST($\mu_1, \sigma_1, n_1, \mu_2, \sigma_2, n_2, \alpha = 0.05$)
  $t_0 = \dfrac{\mu_1 - \mu_2}{\sqrt{\dfrac{\sigma_1^2}{n_1} + \dfrac{\sigma_2^2}{n_2}}}$
  $\nu = \dfrac{\left(\dfrac{\sigma_1^2}{n_1} + \dfrac{\sigma_2^2}{n_2}\right)^2}{\dfrac{\left(\frac{\sigma_1^2}{n_1}\right)^2}{n_1 - 1} + \dfrac{\left(\frac{\sigma_2^2}{n_2}\right)^2}{n_2 - 1}}$
  $p = 2 \cdot \left(1 - T_{\lfloor \nu \rfloor}(|t_0|)\right)$ ▷ T_ν is the cumulative distribution function
  if $p < \alpha$ **then**
    return (p, True) ▷ Significant difference
  else
    return (p, False) ▷ No significant difference
  end if
end function

To better visualise the results, figures 5.11 and 5.12 show the trajectories of the faulty BlueROV2 Heavy controlled by the different controllers in the case where $T0$ is at 0% and 40% over 10 evaluation tests.

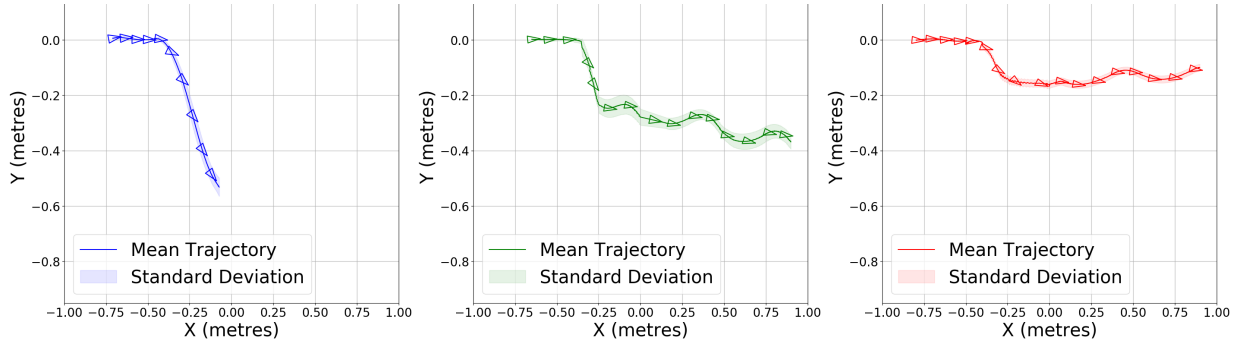


Figure 5.11: Mean and Standard Deviation of the 2D Trajectory of the BlueROV2 Heavy controlled by the standard PID (left), conventional adaptive PID (middle) and DRL-based PID (right) controllers at 0% efficiency of $T0$, over 10 evaluation tests.

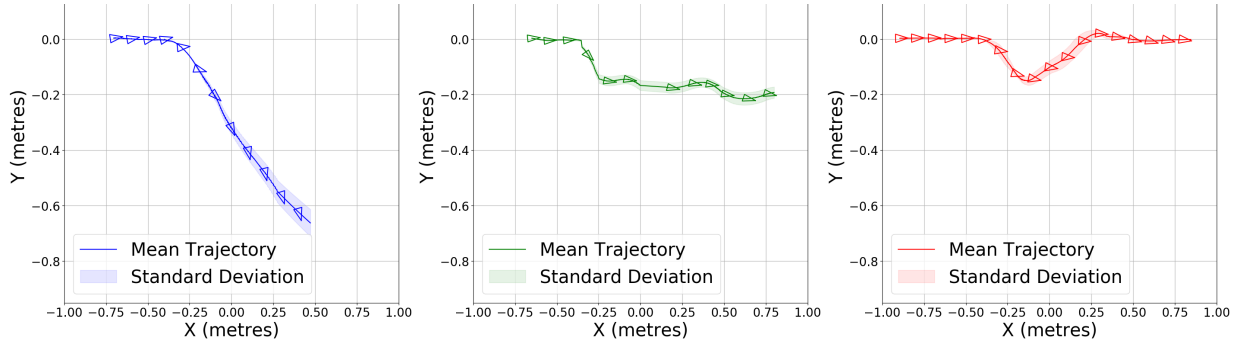


Figure 5.12: Mean and Standard Deviation of the 2D Trajectory of the BlueROV2 Heavy controlled by the standard PID (left), conventional adaptive PID (middle) and DRL-based PID (right) controllers at 40% efficiency of $T0$, over 10 evaluation tests.

We can observe in these figures that the trajectory of the faulty drone controlled by a standard PID showed significant deviation along the y-axis. The conventional adaptive PID controller does reduce the trajectory error compared to the standard controller, but it is still largely outperformed by the DRL-based approach. This is likely because its response is limited to gain adjustment, whereas the DRL-based controller learns a more efficient control reallocation strategy through training, clearly reducing the undesired errors.

The results obtained using PID controllers, both in simulation and from experimental trials, are part of the work published under the title "Control Reallocation Using Deep Reinforcement Learning for Actuator Fault Recovery of an Autonomous Underwater Vehicle", presented at the International Conference on Robotics and Automation (ICRA) in May 2025 in Atlanta. The novelty of the paper is performing control reallocation using DRL within a PID control framework for actuator fault scenarios in AUVs (Lagattu, Artusi, et al. 2025).

The next section focuses on the experimental results obtained with the LQR-based controllers.

5.3.4 LQR-based controllers experimental results

The controllers evaluated include the standard LQR, conventional adaptive LQR, and DRL-based LQR controllers. As for the PID-based controllers, the fault detection threshold was set at $y = \pm 0.1$m. The evaluation was performed on each horizontal thruster individually and the results are displayed in Table 5.6.

	0%		10%		40%		70%	
	δ_y	δ_{yaw}	δ_y	δ_{yaw}	δ_y	δ_{yaw}	δ_y	δ_{yaw}
Standard LQR	25.30 ± 2.35	39.78 ± 4.21	22.37 ± 2.47	28.13 ± 2.82	16.20 ± 1.38	24.38 ± 6.09	10.79 ± 1.16	22.47 ± 5.44
Adaptive LQR	16.49 ± 3.09	65.30 ± 6.62	14.53 ± 0.66	49.30 ± 5.25	12.64 ± 1.91	21.07 ± 3.15	5.07 ± 1.00	17.65 ± 2.00
LQR/DRL	11.95 ± 1.09	28.32 ± 1.89	8.40 ± 1.13	23.69 ± 4.87	5.96 ± 0.17	17.56 ± 3.60	3.87 ± 0.35	13.95 ± 2.34

Table 5.6: Mean and standard deviation of performance indices for the 4 thrusters evaluated individually, with 0%, 10%, 40%, and 70% efficiency with standard LQR, conventional adaptive LQR and DRL-based LQR controllers, evaluated over 10 episodes on a physical BlueROV2 Heavy for each case.

From the table results, it can be observed that at 0% thruster efficiency, the DRL-based LQR controller achieved approximately 52.4% and 28.6% reductions in y error compared to the standard LQR and the conventional adaptive LQR controllers, respectively. In terms of yaw error, reductions of 16.7% and 56.5% were observed. Similarly, at 10% efficiency, it improved y error by 62.45% and 41.2%, and yaw error by 16.7% and 52.4% relative to the same benchmarks. Unlike in simulation, the standard LQR controller outperforms the conventional adaptive controller at both 0% and 10% efficiency.

The discrepancies between simulation and real-world results may be due to the approximation of the drone model in simulation and to the presence of a cable attached to the drone, which may cause drifting. At 40% and 70% efficiency, the DRL-based LQR controller outperforms the compared controller, as observed in simulation.

Tables 5.7 and 5.8 present the p-values obtained from the results in Table 5.6.

	DRL-based LQR/Standard	DRL-based LQR/Adaptive LQR
0%	4.80e-13	6.03e-06
10%	1.89e-13	1.28e-10
40%	3.14e-11	4.39e-08
70%	1.63e-10	5.63e-4

Table 5.7: p-value computation for y error across efficiency levels from 0% to 70%, using physical drone data.

	DRL-based LQR/Standard	DRL-based LQR/Adaptive LQR
0%	7.60e-10	1.64e-13
10%	1.15e-4	5.77e-15
40%	1.21e-05	1.81e-04
70%	1.63e-06	1.24e-08

Table 5.8: p-value computation for yaw error across efficiency levels from 0% to 70%, using physical drone data.

As for the previous results, the p-values are extremely small, indicating the statistical significance of the performance differences between DRL-based LQR controller and the compared controllers.

To better visualise the results, Figures 5.13, 5.14, 5.15, 5.16 illustrate the mean and standard deviation of the 2D trajectory of the BlueROV2 Heavy, controlled by three different controllers: the standard LQR (left), conventional adaptive LQR (middle), and DRL-based LQR (right) controllers. Each set of images shows the trajectory of the AUV using these controllers, under two different thruster efficiencies.

Figure 5.13 displays the results at 0% efficiency of $T1$, with each controller tested over 10 evaluation runs. Figure 5.14 then presents the results at 40% efficiency of $T1$. Similarly, Figure 5.15 shows the trajectories at 0% efficiency of $T2$, and finally, Figure 5.16 illustrates the outcomes at 40% efficiency of $T2$.

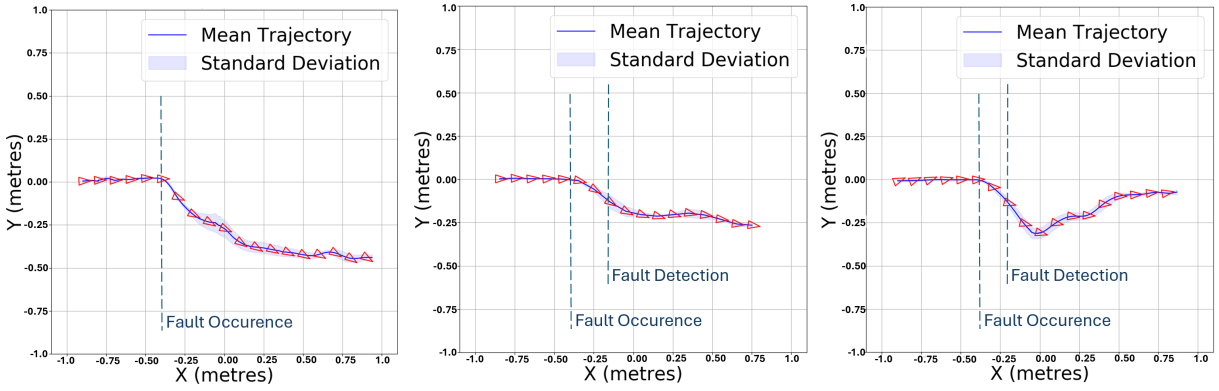


Figure 5.13: Mean and standard deviation of the 2D trajectory of the BlueROV2 Heavy controlled by the standard LQR (left), conventional adaptive LQR (middle) and DRL-based LQR (right) controllers at 0% efficiency of $T1$, over 10 evaluation tests.

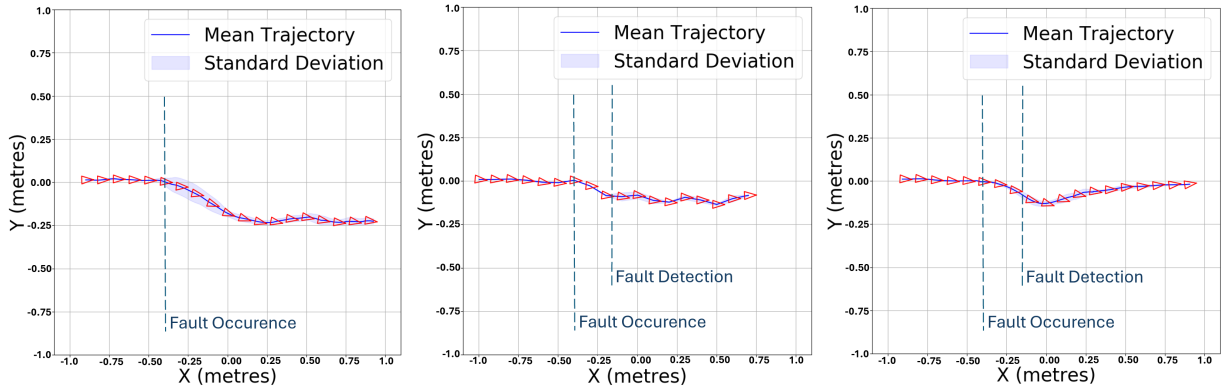


Figure 5.14: Mean and standard deviation of the 2D trajectory of the BlueROV2 Heavy controlled by the standard LQR (left), conventional adaptive LQR (middle) and DRL-based LQR (right) controllers at 40% efficiency of $T1$, over 10 evaluation tests.

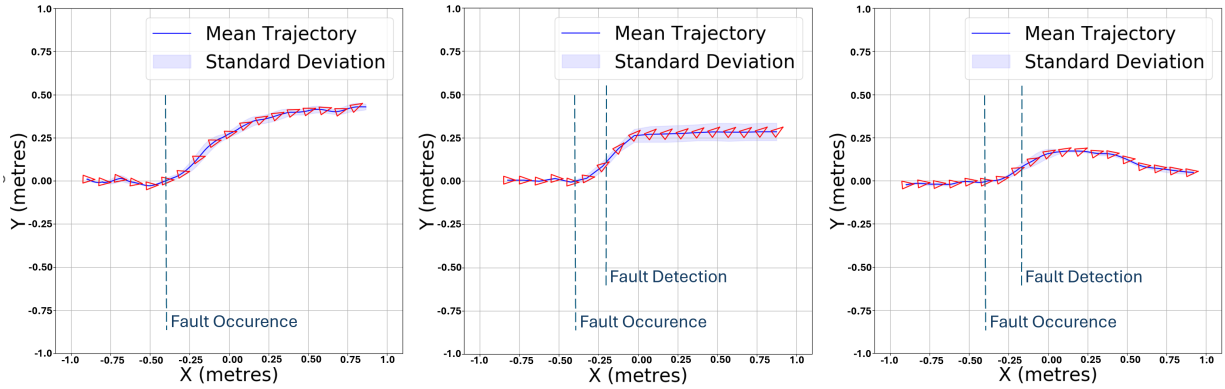


Figure 5.15: Mean and standard deviation of the 2D trajectory of the BlueROV2 Heavy controlled by the standard LQR (left), conventional adaptive LQR (middle) and DRL-based LQR (right) controllers at 0% efficiency of $T2$, over 10 evaluation tests.

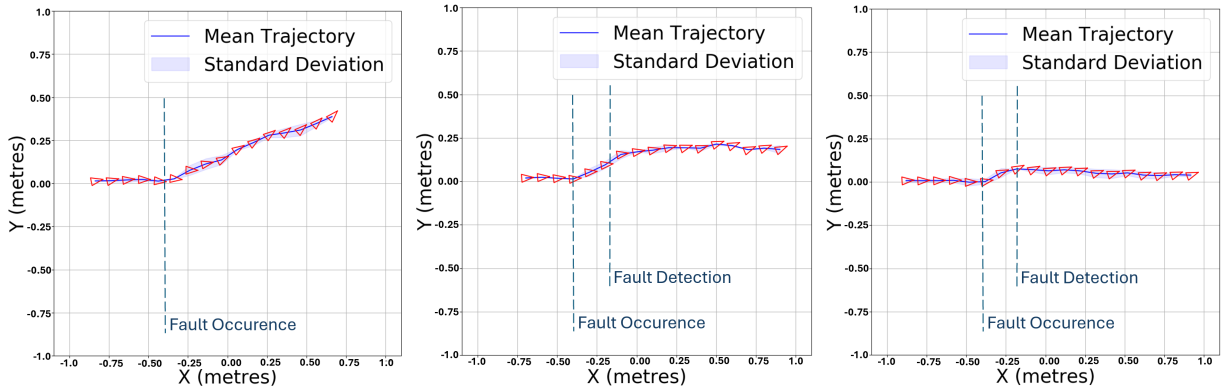


Figure 5.16: Mean and standard deviation of the 2D trajectory of the BlueROV2 Heavy controlled by the standard LQR (left), conventional adaptive LQR (middle) and DRL-based LQR (right) controllers at 40% efficiency of $T2$, over 10 evaluation tests.

From these figures, we observe that in the case of the standard LQR, there is a significant deviation from the straight-line path, with BlueROV2 Heavy deviates entirely, either towards negative y values for $T1$ or positive y values for $T2$. Regarding the conventional adaptive LQR, an improvement over the standard LQR can be seen, as the trajectory of BlueROV2 Heavy appears to self-correct slightly and deviate less than with the standard LQR controller. We can also observe that the DRL-based LQR controller significantly outperforms both the standard LQR and the conventional adaptive LQR, as the drone's trajectory, after the fault is detected, aligns more closely with the straight line.

These observations suggest that direct adaptation of the LQR helps limit deviation by increasing the value of the $\mathbf{Q}$ matrix associated with y-axis errors, which increases the importance of correcting these deviations in the LQR cost function. However, this is insufficient; notably in Figure 5.15, we observe a yaw orientation error that grows significantly. This is due to the fact that as in simulation, the direct adaptive approach does not redistribute forces among the actuators and does not account for the geometric configuration of the BlueROV2 Heavy's thrusters.

The observed performance of the DRL-based LQR controller indicates that the CAM update by DRL allows for an overall better trajectory correction, as it enables a redistribution of forces. It suggests that the DRL model implicitly recognises the faults, relying on experience from training to identify the faulty behaviour of the BlueROV2 Heavy and learn how to redistribute forces among the actuators correctly despite the fault.

The results obtained using LQR controllers, both in simulation and through physical experiments, are part of the work submitted to The International Journal of Robotics Research (IJRR) on 30 April 2025, under the title "Actuator Fault Recovery with Deep Reinforcement Learning in a Model-Based Control Framework: Application to a Physical AUV". This contribution explores control reallocation using DRL in a model-based control framework, with a focus on linear quadratic regulation for actuator fault scenarios in AUVs (Lagattu, Alham, et al. 2025). The next section draws a conclusion from these experimental results.

5.3.5 Experimental results discussion

The experimental results on the physical AUV confirm the trends observed in simulation. Despite the differences between the simulation environment and real-world deployment—such as the presence of the tether, the fact that thruster inputs are not injected directly into the system, and discrepancies in hydrodynamic effects—the relative performance of the controllers remains consistent. Since no fine-tuning of the DRL model was performed on the actual drone, this shows that the simulation tools used were realistic enough for the DRL to adapt to the physical drone.

The DRL-based controllers demonstrate superior performances compared to the standard and conventional adaptive ones. The standard controllers cause the drone to deviate significantly from the trajectory, and the direct adaptive methods manage to reduce this deviation but tend to result in large yaw angles, without being able to recover the trajectory of the BlueROV2 Heavy. This is due to the fact that direct adaptive controllers do not explicitly account for the geometry of the thrusters when faults occur. By contrast, the use of DRL makes it possible to adapt to the new thruster configuration, enabling finer control through force redistribution. DRL capabilities also allows the trained policy to generalise to faults of lower severity than those encountered during training.

The statistical analysis based on p-value computations confirms that, with a set of ten experimental episodes, the performance differences between the controllers are statistically significant.

It is also worth noting that the DRL-based LQR controller performs better than the DRL-based PID controller in terms of yaw correction, while being slightly less accurate in the y-axis. This may be explained by the fact that the PID gains were specifically tuned for the physical drone, whereas the LQR gains used in the experiments were those obtained from simulation-based optimisation. The optimisation procedure used for the LQR parameters described in section 4.2.2 relies on iterative adjustments and evaluation of performance in simulation, which makes it difficult to apply directly to the physical drone due to the time and safety constraints.

The deployment of the proposed approach on the physical platform validates the method under real-world conditions and demonstrates the adaptability of the solution. Based on the findings discussed, the next chapter will provide a summary and conclusion of this thesis, highlighting key insights and future directions.

Chapter 6

Conclusion

6.1 Thesis outcomes

The objective of this thesis was to develop a control structure improving the overall performance of an autonomous dynamic system subject to an actuator fault compared to existing methods. This objective has been addressed by developing a general method combining DRL and model-based control. The choice of a hybrid controller stems from the desire to combine the advantages of model-based control, which provides a stability framework for the closed-loop system, with those of DRL, which enables adaptation to actuator faults.

Existing fault-tolerant control methods

In the first part of this thesis, we reviewed various FTC strategies developed to address actuator faults in autonomous dynamic systems. The key characteristics of these FTC approaches include the ability to maintain system stability and performance despite the presence of faults. However, they all present certain limitations. The main challenge of indirect adaptive methods, which are the type of methods used in this thesis since they take into account the geometry of the actuators, is the need for an explicit fault diagnosis — a requirement that can be particularly difficult to satisfy in complex systems with many actuators as the number of possible fault scenarios increases rapidly. To overcome this limitation while preserving the key characteristics of FTC, the method proposed in this thesis performs control reallocation without relying on explicit fault diagnosis, by integrating DRL within a model-based control framework.

Hybrid approach for actuator fault recovery

The proposed approach (Lagattu, Alham, et al. 2025) is intended for systems in which explicit actuator fault diagnosis is particularly challenging, and where the actuator

configuration is complex, involving a large number of actuators. The methodology combines the strengths of model-based control and DRL within a hybrid controller structure, addressing three research questions defined in the introduction. The first research question asked: *How to integrate a DRL method in a model-based controller framework in order to minimise the effects of the actuator faults on the behaviour of the vehicle?* This thesis has shown that integration can be achieved by employing DRL to reallocate control effort. In this design, DRL is used when an abnormal behaviour is detected, while the model-based component preserves a stability framework during fault-free conditions. In a fault situation, the DRL model outputs weights to modify the CAM of a model-based controller, allowing to take into account the actuator geometric configuration of the system. This hybrid controller combines the learning strengths of DRL with the guarantees of model-based control, addressing the five key characteristics of FTC: optimal recovery performance through efficient force reallocation, adaptability to diverse fault situations, generalisability to previously unseen faults, stability ensured by the model-based framework, and resource-efficiency through low-cost deployment once DRL training is complete.

The second research question asked: *How a DRL model can be trained to allow the hybrid controller to overcome a wide range of actuator faults, without explicit fault diagnosis?* The proposed solution is to pre-train the DRL model in simulation to implicitly recognise fault patterns from system behaviour, using the observation vector as input. During operation, the DRL model relies on the deviations between the reference state and the observed state, without requiring explicit fault diagnosis. At the beginning of each training episode, a fault scenario is randomly selected to enable domain randomisation. Besides, the reward function is designed to minimise trajectory deviations. The algorithm is trained until the reward converges, and after training the DRL model is able to reallocate the control effort accordingly, enabling the controller to handle both known and previously unseen faults thanks to its generalisation ability.

The third research question asked: *How can system stability be guaranteed while integrating a DRL method in the control loop, in a context where dynamic behaviours are unpredictable?* First, this challenge is addressed by restricting the use of DRL to fault conditions only, ensuring that a stable model-based control is used in nominal conditions. Second, the DRL-based control reallocation is integrated into the model-based control framework: the controller gain is recalculated in real time at each time step according to the allocation matrix adjusted by the DRL, which ensures that the controller gain provides local stability of the closed-loop control. Finally, to come closer to global stability, a post-processing block has been added to the system. This block helps ensure that the outputs of the DRL model are subject to small and continuous variations, aiming to approximate the conditions for global stability. The proposed approach is particularly suited to overactuated systems

with complex geometries, such as the BlueROV2 Heavy, which served as the application case in this thesis.

Application case: BlueROV2 Heavy

The proposed approach has been applied to the BlueROV2 Heavy, an AUV equipped with eight thrusters. It was first necessary to develop a simulation environment to train the DRL model and to facilitate its deployment on the physical drone (Lagattu, Lechene, et al. 2025). For this application case, the SAC algorithm was chosen for the DRL component, and an LQR controller was selected for the model-based control. In order to implement the LQR controller, the model matrices of the BlueROV2 Heavy have been identified. The training scenario consisted in following a straight line along the x-axis, with the occurrence of a fault introduced during the mission. To reduce training time, in the application case, the DRL model is trained only on the most severe faults, relying on the generalisation capacity of DRL to handle less severe faults. At each episode, the fault affected one of the four horizontal thrusters selected at random, and the fault severity was also randomly chosen: either no thrust (0% efficiency, i.e., total fault) or a severe fault with only 10% remaining efficiency. To facilitate transfer to the real drone, a time window is introduced to better account for the inertia of the BlueROV2 Heavy. Once the DRL model training was complete, the proposed method was applied to the drone in simulation and real-world conditions.

Evaluation results

The evaluation relied on performance indices based on trajectory tracking errors induced by actuator faults. The evaluation scenario was identical to the one used during training but a broader range of fault levels was introduced in order to assess the generalisation capability of the proposed method. The last research question was *how to evaluate the impact of combining DRL and model-based control on fault recovery efficiency compared to existing methods*. This was addressed by benchmarking the hybrid controller against several alternatives on the BlueROV2 Heavy in simulation and on a physical drone. As part of preliminary investigations, and to validate the choice of the SAC algorithm, an end-to-end controller was evaluated in simulation (Lagattu, Le Chenadec, et al. 2024). While this controller was able to complete the mission, it exhibited more abrupt variations in thruster commands compared to the proposed approach. This confirmed the benefit of embedding DRL into a model-based framework rather than relying exclusively on purely learning-based control strategies. The proposed controller was then compared, for the same straight-line tracking mission, to standard and direct adaptive controllers, in order to benchmark against a reference controller and an adaptive method not relying on DRL. It was also compared to a non-model-based PID controller (Lagattu, Artusi, et al. 2025), to assess the contribution

of using a model-based controller such as LQR. Results showed that integrating DRL within a model-based control framework provides advantages. In particular, the hybrid controller significantly reduces trajectory tracking errors in faulty scenarios, while maintaining smooth control signals, achieving up to a 62.45% reduction in trajectory error on the physical BlueROV2 Heavy compared to the other methods. This improvement was most pronounced in the presence of severe actuator faults, where conventional methods struggled to preserve performance.

6.2 Research perspectives

In this thesis, we have demonstrated the benefits of using DRL into a model-based FTC framework. We can identify two major research avenues: (1) increasing scenario realism, and (2) addressing inherent challenges of the proposed approach.

Towards realistic and challenging fault-tolerant scenarios

In the current application case, the proposed control strategy has been validated under simplified conditions, where trajectory tracking is restricted to the x-axis over a distance of a few metres. This was motivated by the need to facilitate an initial validation of the method and the application to a physical drone while reducing training time. However, a possible extension of this work is to evaluate the approach in longer, more complex and realistic scenarios that better reflect the operational conditions of underwater vehicles, such as covering areas or AUV docking. A first direction involves extending the controller to full six DoF trajectory tracking. This requires training the DRL agent on a broader range of motion patterns including simultaneous variations in yaw, pitch, and roll, as well as variability in the starting and ending points. Such an extension would lead to significantly longer training times, as the controller would need to learn a more diverse set of behaviours and fault responses. In addition, the neural network architecture used to adapt the CAM would need to predict a complete set of weight adjustments for each actuator, as faults could impact any direction.

Secondly, more complex fault types should be explored. These include multiple simultaneous faults affecting several actuators at once, time-varying partial faults, or nonlinear faults. Moreover, the condition of the system upon fault occurrence should be taken into account. For instance, a collision might induce both a sudden actuator fault and a strong perturbation in velocity and orientation. This could result in an abrupt and destabilising transient that the controller must handle. To address such cases, robust learning strategies could be investigated, such as LSTM-based agents (Y. Yu et al. 2019) that better capture temporal dependencies and abrupt transitions. These approaches would also increase DRL

model training complexity. In addition, our analysis has been restricted to linear or linearised systems around an equilibrium point, which facilitated the controller design and the analysis of local stability. A possible extension would be to consider nonlinear dynamics, which would necessitate to adapt the control architecture in order to maintain performance of the DRL-based control reallocation and the stability of the closed-loop control.

Another major challenge is the presence of external disturbances, such as underwater currents, which were previously assumed to be absent. These can induce deviations in the system's behaviour that may resemble those caused by actuator faults. However, modifying the CAM in response to such deviations could be counterproductive, as the issue is not related to actuation capability. Adapting the control allocation to compensate for an apparent anomaly could degrade performance if the anomaly is actually due to an environmental factor rather than a fault, as the controller may unnecessarily reassign forces in a way that conflicts with the optimal response to external disturbances, thereby introducing instability or inefficiency. This raises an important limitation of the current approach, which aims to avoid explicit fault diagnosis and instead relies on observing deviations from expected behaviour. One potential solution could be to enrich the observation space with estimations of external forces (e.g., using observers or current estimation algorithms (K. M. B. Lee et al. 2019)), allowing the DRL agent to differentiate between internal faults and external perturbations.

Finally, several improvements can be made regarding the transfer to the physical drone. For better performance of the DRL model, fine-tuning could be carried out directly on the physical drone. In addition, instead of using a camera for localisation, the sensors of the drone should be employed, which necessarily introduce noise. To handle this, it would be needed to model the noise in simulation and take it into account during the training of the DRL model. Moreover, a more accurate fault simulation could be implemented on the physical drone, since, as described in section 5.3.2, the fault simulation performed was not ideal. Reconfiguring the software of the BlueROV2 Heavy would be required to allow such an ideal fault simulation. It would also be interesting to transfer the DRL model to other systems, for example by using a pre-trained DRL model and fine-tuning it on a different drone, or by training a new model to handle faults affecting different types of actuators, such as the fins of a torpedo-shaped AUV. These extensions aim to move the proposed approach closer to deployment in real-world scenarios, where uncertainty, variability, and lack of precise fault information are common. They also highlight the trade-off between robustness and training feasibility, which future work will need to balance carefully.

Challenges of DRL-based fault adaptation

The proposed methodology relies on the assumption that different faults produce

distinguishable patterns in the vehicle's behaviour, enabling the DRL model to implicitly identify and handle them. However, actuator faults may not always yield unique effects: if two faults result in the same abnormal motion—due to thruster geometry and control law coupling—the policy will struggle to apply different corrective forces, leading to ambiguous updates and suboptimal adaptation. This phenomenon highlights a key limitation: the mapping from observed state deviation to actuator adjustment must be injective; if it is not, learning may converge to a compromise that partially mitigates both faults simultaneously, but optimally addresses neither. As a future direction, it could be valuable to identify in advance the configurations where such ambiguity arises. This could help delimit the operating conditions where the proposed approach is applicable or guide the design of additional sensor feedback to disambiguate fault signatures when needed. Moreover, the current method does not provide any diagnosis to the operator. While the DRL agent redistributes forces effectively, the human operator has no insight into the nature of the fault. Future work could explore reverse-engineering the adjustments proposed by the DRL model — analysing the redistribution pattern across actuators — to infer and communicate fault characteristics. For instance, a persistent reduction on a particular thruster could reveal information about the condition of this thruster.

Another perspective is to train and use DRL not only in cases of severe actuator faults (0% and 10% efficiency) but across the entire range of faults where DRL is intended to be applied. In this thesis, the DRL model was trained only on severe faults, and its generalisation capability was evaluated for less severe faults. It would be interesting to train DRL on a wider range of faults in order to improve its effectiveness. However, with the algorithm and training method used here, the DRL model did not effectively learn fault situations over an extended range. It would therefore also be necessary to further develop the training process and the DRL algorithm so that training becomes sufficiently effective to handle all fault cases. In line with these perspectives, ongoing work including an internship and a forthcoming PhD will focus on comparing DRL algorithms and improving training efficiency, addressing the identified need for more effective learning strategies. Finally, to enhance performance — particularly for the BlueROV2 Heavy scenario — modern tools like MarineGym (S. Chu et al. 2025), a recent DRL platform for underwater vehicles, should be employed and compared in future work to improve training efficiency and facilitate application on a real drone. Besides, incorporating elements present in real-world conditions—such as sensor noise, actuation latency, and the effect of the tether—into the simulation would provide a more realistic training environment and help bridge the gap between simulation and real-world deployment.

These challenges underline the need for complementary strategies that enhance the reliability and interpretability of the proposed approach across a broader range of fault scenarios.

Contributions/papers

The main contributions of this thesis are summarised below, with the associated papers:

1. **Development of a novel hybrid controller using DRL to overcome thruster faults in an LQR framework, with real-world testing on an AUV.** *Actuator fault recovery with deep reinforcement learning in a model-based control framework: Application to a physical AUV*, Katell Lagattu, Ramy Alham, Thomas Chaffre, Eva Artusi, Gilles Le Chenadec, Karl Sammut, Paulo E. Santos, Benoit Clement, **IJRR** (submitted April 2025, first reviews received, under revision).

2. **Development of a novel hybrid controller using DRL to overcome thruster faults in a PID framework, with real-world testing on an AUV.** *Control Reallocation using Deep Reinforcement Learning for Actuator Fault Recovery of an Autonomous Underwater Vehicle*, Katell Lagattu, Eva Artusi, Gilles Le Chenadec, Karl Sammut, Paulo E. Santos, Benoit Clement, **ICRA 2025**, Atlanta, US (published article, oral + poster).

3. **Methodology using open-access tools for transferring a DRL model to a real AUV, focusing on thruster fault scenarios.** *Sim-to-real transfer for AUV fault control with Deep Reinforcement Learning*, Katell Lagattu, Hélène Lechêne, Eva Artusi, Gilles Le Chenadec, Karl Sammut, Paulo E. Santos, Benoit Clement, **OCEANS 2025**, Brest, France (published article, oral).

4. **Learning method for end-to-end DRL to overcome thruster faults in an AUV.** *DRL-Based Thruster Fault Recovery for Unmanned Underwater Vehicles*, Katell Lagattu, Eva Artusi, Gilles Le Chenadec, Karl Sammut, Paulo E. Santos, Benoit Clement, **ANZCC 2024**, Gold Coast, Australia (published article, oral).

Bibliography

Abbaspour, Alireza et al. (2020). "A survey on active fault-tolerant control systems". In: *Electronics* 9.9, p. 1513.

Abci, Boussad et al. (2020). "An informational approach for sensor and actuator fault diagnosis for autonomous mobile robots". In: *Journal of Intelligent & Robotic Systems* 99.2, pp. 387–406.

Abd Latip, Siti Fadilah et al. (2019). "Adaptive PID actuator fault tolerant control of single-link flexible manipulator". In: *Transactions of the Institute of Measurement and Control* 41.4, pp. 1019–1031.

Abdelaziz, THS and M Valášek (2004). "Pole-placement for SISO linear systems by state-derivative feedback". In: *IEE Proceedings-Control Theory and Applications* 151.4, pp. 377–385.

Abed, W. et al. (2017). "An unmanned marine vehicle thruster fault diagnosis scheme based on OFNDA". In: *Journal of Marine Engineering & Technology*.

Ahmadzadeh, Seyed Reza, Petar Kormushev, and Darwin G. Caldwell (Dec. 2014). "Multi-objective reinforcement learning for AUV thruster failure recovery". en. In: *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. Orlando, FL, USA: IEEE, pp. 1–8. ISBN: 978-1-4799-4552-8. DOI: 10.1109/ADPRL.2014.7010621. URL: http://ieeexplore.ieee.org/document/7010621/ (visited on 11/17/2022).

Ahmed, Ibrahim, Marcos Quinones-Grueiro, and Gautam Biswas (2022). "A high-fidelity simulation test-bed for fault-tolerant octo-rotor control using reinforcement learning". In: *2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC)*, pp. 1–10. DOI: 10.1109/DASC55683.2022.9925862.

Ahmed Ali, Sofiane, Mohammed Guermouche, and Nicolas Langlois (2015). "Fault-tolerant control based Super-Twisting algorithm for the diesel engine air path subject to loss-of-effectiveness and additive actuator faults". In: *Applied Mathematical Modelling* 39.15, pp. 4309–4329. ISSN: 0307-904X. DOI: https://doi.org/10.1016/j.apm.2014.12.047. URL: https://www.sciencedirect.com/science/article/pii/S0307904X14007227.

Alessandri, A., M. Caccia, and G. Veruggio (1999). "Fault detection of actuator faults in unmanned underwater vehicles". In: *Control Engineering Practice* 7.3, pp. 357–368. ISSN: 0967-0661. DOI: https://doi.org/10.1016/S0967-0661(98)00169-5. URL: https://www.sciencedirect.com/science/article/pii/S0967066198001695.

Amari, Shun-ichi (1993). "Backpropagation and stochastic gradient descent method". In: *Neurocomputing* 5.4-5, pp. 185–196.

Amin, Arslan Ahmed and Khalid Mahmood Hasan (2019). "A review of fault tolerant control systems: advancements and applications". In: *Measurement* 143, pp. 58–68.

Anderson, Brian DO et al. (2001). "Multiple model adaptive control with safe switching". In: *International journal of adaptive control and signal processing* 15.5, pp. 445–470.

Andrychowicz, Marcin et al. (2016). "Learning to learn by gradient descent by gradient descent". In: *Advances in neural information processing systems* 29.

Ardema, Mark D (2004). *Newton-Euler Dynamics*. Springer Science & Business Media.

Arıcı, Mehmet and Tolgay Kara (2020). "Robust adaptive fault tolerant control for a process with actuator faults". In: *Journal of Process Control* 92, pp. 169–184.

Artusi, Eva, Fabien Chaillan, and Aldo Napoli (2021). "Path planning for a maritime suface ship based on Deep Reinforcement Learning and weather data". In: *OCEANS 2021: San Diego – Porto*, pp. 1–8.

Asghari, Mohammad et al. (2022). "Transformation and linearization techniques in optimization: A state-of-the-art survey". In: *Mathematics* 10.2, p. 283.

Åström, Karl J and Tore Hägglund (2006). "PID control". In: *IEEE Control Systems Magazine* 1066, pp. 30–31.

Åström, Karl J and Björn Wittenmark (2013). *Adaptive control*. Courier Corporation.

Åström, Karl Johan and Richard Murray (2021). *Feedback systems: an introduction for scientists and engineers*. Princeton university press.

Atassi, Ahmad N and Hassan K Khalil (1999). "A separation principle for the stabilization of a class of nonlinear systems". In: *IEEE Transactions on Automatic Control* 44.9, pp. 1672–1687.

Attia, Mohamed et al. (2018). "Towards trusted autonomous surgical robots". In: *2018 IEEE international conference on systems, man, and cybernetics (SMC)*. IEEE, pp. 4083–4088.

Ávila Okada, Kenji Fabiano et al. (2021). "A Survey on Fault Detection and Diagnosis Methods". In: *2021 14th IEEE International Conference on Industry Applications (INDUSCON)*, pp. 1422–1429. DOI: 10.1109/INDUSCON51756.2021.9529495.

Barata, João Carlos Alves and Mahir Saleh Hussein (2012). "The Moore–Penrose pseudoinverse: A tutorial review of the theory". In: *Brazilian Journal of Physics* 42, pp. 146–165.

Barron, EN and H Ishii (1989). "The Bellman equation for minimizing the maximum cost." In: *NONLINEAR ANAL. THEORY METHODS APPLIC.* 13.9, pp. 1067–1090.

Batlle, Carles (2005). "Passive control theory I". In: *EURON/GEOPLEX Summer School on Modeling and Control of Complex Dynamical Systems, Bertinoro, Italy*, p. 72.

Benesty, Jacob et al. (2011). "Recursive least-squares algorithms". In: *A perspective on stereophonic acoustic echo cancellation*, pp. 63–69.

Benner, Peter (2006). "Control theory". In: *Handbook of Linear Algebra*, pp. 57–1.

Benosman, M (2011). "Passive fault tolerant control". In: *Robust Control Theory Appl* 9.

Bhan, Luke, Marcos Quinones-Grueiro, and Gautam Biswas (2021). "Fault tolerant control combining reinforcement learning and model-based control". In: *2021 5th International Conference on Control and Fault-Tolerant Systems (SysTol)*. IEEE, pp. 31–36.

Bishop, Richard C Dorf Robert H (2011). *Modern control systems*. Pearson.

Blanchini, Franco (1995). "Nonquadratic Lyapunov functions for robust control". In: *Automatica* 31.3, pp. 451–461.

Blanke, Mogens (2005). "Diagnosis and fault-tolerant control for ship station keeping". In: *Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation Intelligent Control, 2005.* IEEE, pp. 1379–1384.

Blanke, Mogens, Marcel Staroswiecki, and N Eva Wu (2001). "Concepts and methods in fault-tolerant control". In: *Proceedings of the 2001 American control conference.(Cat. No. 01CH37148)*. Vol. 4. IEEE, pp. 2606–2620.

Bosch, Paul PJ van den and Alexander C van der Klauw (2020). *Modeling, identification and simulation of dynamical systems*. crc Press.

Bradtke, Steven (1992). "Reinforcement learning applied to linear quadratic regulation". In: *Advances in neural information processing systems* 5.

Brockman, Greg et al. (2016). "Openai gym". In: *arXiv preprint arXiv:1606.01540*.

Brosilow, Coleman and Babu Joseph (2002). *Techniques of model-based control*. Prentice Hall Professional.

Buhmann, Martin Dietrich (2000). "Radial basis functions". In: *Acta numerica* 9, pp. 1–38.

Buyuksalih, Ismail et al. (2017). "3D modelling and visualization based on the unity game engine–advantages and challenges". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 4, pp. 161–166.

Caccia, M et al. (2001). "Experiences on actuator fault detection, diagnosis and accomodation for ROVs". In: *International Symposiyum of Unmanned Untethered Sub-mersible Technol*, pp. 3–21.

Chaffre, Thomas, Gilles Le Chenadec, et al. (2021). "Direct adaptive pole-placement controller using deep reinforcement learning: Application to AUV control". In: *IFAC-PapersOnLine* 54.16, pp. 333–340.

Chaffre, Thomas, Jonathan Wheare, et al. (2025). "Sim-to-real transfer of adaptive control parameters for AUV stabilisation under current disturbance". In: *The International Journal of Robotics Research* 44.3, pp. 407–430. DOI: 10.1177/02783649241272115. eprint: https://doi.org/10.1177/02783649241272115. URL: https://doi.org/10.1177/02783649241272115.

Choi, Jin-Kyu and Hayato Kondo (2010). "On fault-tolerant control of a hovering AUV with four horizontal and two vertical thrusters". In: *OCEANS'10 IEEE SYDNEY*, pp. 1–6. DOI: 10.1109/OCEANSSYD.2010.5603565.

Chopra, Rohan and Sanjiban Sekhar Roy (2020). "End-to-end reinforcement learning for self-driving car". In: *Advanced Computing and Intelligent Engineering: Proceedings of ICACIE 2018, Volume 1*. Springer, pp. 53–61.

Christensen, Anders Lyhne (2008). "Fault detection in autonomous robots". In: *Phd, Université Libre de Bruxelles*.

Chu, Shuguang et al. (2025). "MarineGym: A High-Performance Reinforcement Learning Platform for Underwater Robotics". In: *arXiv preprint arXiv:2503.09203*.

Chu, Zhenzhong et al. (2020). "Fault reconstruction using a terminal sliding mode observer for a class of second-order MIMO uncertain nonlinear systems". In: *ISA Transactions* 97, pp. 67–75. ISSN: 0019-0578. DOI: https://doi.org/10.1016/j.isatra.2019.07.024. URL: https://www.sciencedirect.com/science/article/pii/S0019057819303131.

Cimpoesu, Elena M, Bogdan D Ciubotaru, and Dan Stefanoiu (2013). "Fault detection and diagnosis using parameter estimation with recursive least squares". In: *2013 19th International Conference on Control Systems and Computer Science*. IEEE, pp. 18–23.

Dankwa, Stephen and Wenfeng Zheng (2019). "Twin-delayed ddpg: A deep reinforcement learning technique to model a continuous movement of an intelligent robot agent". In: *Proceedings of the 3rd international conference on vision, image and signal processing*, pp. 1–5.

Daum, Frederick E (2021). "Extended kalman filters". In: *Encyclopedia of Systems and Control*. Springer, pp. 751–753.

Davies, Jessica et al. (2009). "Active versus passive fault tolerant control of a high redundancy actuator". In: *2009 European control conference (ECC)*. IEEE, pp. 3671–3676.

De Kleer, Johan and Brian C Williams (1987). "Diagnosing multiple faults". In: *Artificial intelligence* 32.1, pp. 97–130.

Debnath, Lokenath (2002). "Recent developments in the Wigner-Ville distribution and time-frequency signal analysis". In: *Proc. Indian Natl. Sci. Acad., A*. Vol. 68, pp. 35–56.

Ding, Steven X (2021). *Advanced methods for fault diagnosis and fault-tolerant control*. Springer.

Dorato, Peter (1987). "A historical review of robust control". In: *IEEE Control Systems Magazine* 7.2, pp. 44–47.

Doukhi, Oualid and Deok-Jin Lee (2021). "Deep reinforcement learning for end-to-end local motion planning of autonomous aerial robots in unknown outdoor environments: Real-time flight experiments". In: *Sensors* 21.7, p. 2534.

Dulac-Arnold, Gabriel, Daniel Mankowitz, and Todd Hester (2019). "Challenges of real-world reinforcement learning". In: *arXiv preprint arXiv:1904.12901*.

Elliott, Howard, Roberto Cristi, and Manohar Das (1985). "Global stability of adaptive pole placement algorithms". In: *IEEE transactions on automatic control* 30.4, pp. 348–356.

Ernits, Juhan, Richard Dearden, and Miles Pebody (Sept. 2010). "Automatic fault detection and execution monitoring for AUV missions". In: *2010 IEEE/OES Autonomous Underwater Vehicles*. ISSN: 2377-6536, pp. 1–10. DOI: 10.1109/AUV.2010.5779660.

Escobet, Teresa et al. (2019). *Fault diagnosis of dynamic systems*. Springer.

Falkenberg, Thomas, René Tavs Gregersen, and Mogens Blanke (2014). "Navigation system fault diagnosis for underwater vehicle". In: *IFAC Proceedings Volumes* 47.3, pp. 9654–9660.

Farivar, Faezeh and Majid Nili Ahmadabadi (2015). "Continuous reinforcement learning to robust fault tolerant control for a class of unknown nonlinear systems". In: *Applied Soft Computing* 37, pp. 702–714.

Fei, Fan et al. (2020). "Learn-to-Recover: Retrofitting UAVs with Reinforcement Learning-Assisted Flight Control Under Cyber-Physical Attacks". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7358–7364. DOI: 10.1109/ICRA40945.2020.9196611.

Fossen, Thor I (1999). "Guidance and control of ocean vehicles". In: *University of Trondheim, Norway, Printed by John Wiley & Sons, Chichester, England, ISBN: 0 471 94113 1, Doctors Thesis*.

Fossen, Thor I., Tor Arne Johansen, and Tristan Perez (2009). "A Survey of Control Allocation Methods for Underwater Vehicles". In: *Underwater Vehicles*. Ed. by Alexander V. Inzartsev. Rijeka: IntechOpen. Chap. 7. DOI: 10.5772/6699. URL: https://doi.org/10.5772/6699.

François-Lavet, V. et al. (2018). "An introduction to deep reinforcement learning". In: *Foundations and Trends® in Machine Learning*.

Franklin, Gene F et al. (2002). *Feedback control of dynamic systems*. Vol. 4. Prentice hall Upper Saddle River.

Gao, Zhiwei, Carlo Cecati, and Steven X. Ding (2015). "A Survey of Fault Diagnosis and Fault-Tolerant Techniques—Part I: Fault Diagnosis With Model-Based and Signal-Based Approaches". In: *IEEE Transactions on Industrial Electronics* 62.6, pp. 3757–3767.

Garrido-Jurado, S. et al. (2014). "Automatic generation and detection of highly reliable fiducial markers under occlusion". In: *Pattern Recognition* 47.6, pp. 2280–2292. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2014.01.005.

Ghasemi, Anahita, Ali Abootalebi, and Mohammad Mesbah (2014). "Reliable Adaptive Controller for Uncertain System: An LMI Approach". In: *Indian Journal of Science and Technology* 7.10, pp. 1663–1673.

Goel, Ankur and Akhilesh Swarup (2015). "High order super twisting sliding mode control of robotic manipulator". In: *ICIC express letters. Part B, Applications: an international journal of research and surveys* 6.11, pp. 3095–3101.

Goodrich, Michael A, Alan C Schultz, et al. (2008). "Human–robot interaction: a survey". In: *Foundations and Trends® in Human–Computer Interaction* 1.3, pp. 203–275.

Goodwin, Graham Clifford, Stefan F Graebe, Mario E Salgado, et al. (2001). *Control system design*. Vol. 240. Prentice Hall Upper Saddle River.

Görges, Daniel (2019). "Distributed adaptive linear quadratic control using distributed reinforcement learning". In: *IFAC-PapersOnLine* 52.11, pp. 218–223.

Grigorescu, Sorin et al. (2020). "A survey of deep learning techniques for autonomous driving". In: *Journal of field robotics* 37.3, pp. 362–386.

Haarnoja, T. et al. (2018). "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR.

Hadi, Behnaz, Alireza Khosravi, and Pouria Sarhadi (2022). "Deep reinforcement learning for adaptive path planning and control of an autonomous underwater vehicle". In: *Applied Ocean Research* 129, p. 103326. ISSN: 0141-1187. DOI: https://doi.org/10.1016/j.apor.2022.103326. URL: https://www.sciencedirect.com/science/article/pii/S0141118722002589.

He, Long, Mengting Xie, and Ya Zhang (2025). "A Review of Path Following, Trajectory Tracking, and Formation Control for Autonomous Underwater Vehicles". In: *Drones* 9.4. ISSN: 2504-446X. DOI: 10.3390/drones9040286. URL: https://www.mdpi.com/2504-446X/9/4/286.

Hespanha, Joao et al. (2001). "Multiple model adaptive control. Part 2: switching". In: *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal* 11.5, pp. 479–496.

Horn, Roger A (1990). "The hadamard product". In: *Proc. Symp. Appl. Math.* Vol. 40, pp. 87–169.

Howson, Benjamin, Ciara Pike-Burke, and Sarah Filippi (2023). "Optimism and delays in episodic reinforcement learning". In: *International Conference on Artificial Intelligence and Statistics*.

Hua, Changsheng, Linlin Li, and Steven X. Ding (2022). "Reinforcement Learning-Aided Performance-Driven Fault-Tolerant Control of Feedback Control Systems". In: *IEEE Transactions on Automatic Control* 67.6, pp. 3013–3020. DOI: 10.1109/TAC.2021.3088397.

Huang, Norden Eh (2014). *Hilbert-Huang transform and its applications*. Vol. 16. World Scientific.

Ioannou, PA and J Sun (1988). "Theory and design of robust direct and indirect adaptive-control schemes". In: *International Journal of Control* 47.3, pp. 775–813.

Ioannou, Petros A and Jing Sun (2012). *Robust adaptive control*. Courier Corporation.

Jakubczyk, Bronislaw and Witold Respondek (1980). "On linearization of control systems". In: *Bulletin de l'Académie Polonaise des Sciences, Série des Sciences Mathématiques* 28.9-10, pp. 517–522.

Jiang, Jin and Xiang Yu (2012). "Fault-tolerant control systems: A comparative study between active and passive approaches". In: *Annual Reviews in Control* 36.1, pp. 60–72. ISSN: 1367-5788. DOI: https://doi.org/10.1016/j.arcontrol.2012.03.005. URL: https://www.sciencedirect.com/science/article/pii/S1367578812000065.

Johansen, Tor A and Thor I Fossen (2013). "Control allocation—A survey". In: *Automatica* 49.5, pp. 1087–1103.

Kaniewski, Piotr (2020). "Extended Kalman filter with reduced computational demands for systems with non-linear measurement models". In: *Sensors* 20.6, p. 1584.

Katayama, Tohru et al. (2005). *Subspace methods for system identification*. Vol. 1. Springer.

Katipamula, Srinivas and Michael Brambley (Apr. 2005). "Methods for Fault Detection, Diagnostics and Prognostics for Building Systems - A Review Part I". In: *HVAC and R Research* 11. DOI: 10.1080/10789669.2005.10391133.

Keesman, Karel J (2011). *System identification: an introduction*. Springer Science & Business Media.

Kennedy, James and Russell Eberhart (1995). "Particle swarm optimization". In: *Proceedings of ICNN'95-international conference on neural networks*. Vol. 4. ieee, pp. 1942–1948.

Khalil, Hassan K (2002). *Nonlinear systems; 3rd ed.* Prentice-Hall.

Kim, Deokhyung and Hyunjun Jung (2021). "Comparison of Reinforcement Learning Algorithms used in Game AI". In: *Proceedings of the Korean Institute of Information and Commucation Sciences Conference*. The Korea Institute of Information and Commucation Engineering, pp. 693–696.

Kirk, Donald E (2004). *Optimal control theory: an introduction*. Courier Corporation.

Koenig, Nathan and Andrew Howard (2004). "Design and use paradigms for gazebo, an open-source multi-robot simulator". In: *2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566)*. Vol. 3. Ieee, pp. 2149–2154.

Kohl, Nate and Risto Miikkulainen (2009). "Evolving neural networks for strategic decision-making problems". In: *Neural Networks* 22.3, pp. 326–337.

Kouvaritakis, Basil and Mark Cannon (2016). "Model predictive control". In: *Switzerland: Springer International Publishing* 38, pp. 13–56.

Kwakernaak, Huibert (1993). "Robust control and H infinite-optimization—tutorial paper". In: *automatica* 29.2, pp. 255–273.

Lagattu, Katell, Ramy Alham, et al. (2025). "Actuator Fault Recovery with Deep Reinforcement Learning in a Model-Based Control Framework: Application to a Physical AUV". In: *The International Journal of Robotics Research (IJRR)*. Manuscript submitted for review on April 30, 2025.

Lagattu, Katell, Eva Artusi, et al. (2025). "Control Reallocation Using Deep Reinforcement Learning for Actuator Fault Recovery of an Autonomous Underwater Vehicle". In: *Proceedings of the 2025 International Conference on Robotics and Automation (ICRA)*. IEEE.

Lagattu, Katell, Gilles Le Chenadec, et al. (2024). "Drl-based thruster fault recovery for unmanned underwater vehicles". In: *Australian & New Zealand Control Conference (ANZCC)*, pp. 25–30.

Lagattu, Katell, Helene Lechene, et al. (2025). "Sim-to-Real Transfer for AUV Fault Control with Deep Reinforcement Learning". In: *Proceedings of the IEEE OCEANS 2025 Conference*. Brest, France.

Lechene, Helene et al. (2024). "LOTUS: Learning from Operational Teaming with Unmanned Systems". In: *OCEANS 2024 - Singapore*, pp. 1–5. DOI: 10.1109/OCEANS51537.2024.10682309.

Lee, Ki Myung Brian et al. (2019). "Online estimation of ocean current from sparse GPS data for underwater vehicles". In: *2019 International conference on robotics and automation (ICRA)*. IEEE, pp. 3443–3449.

Li, Jiashuai et al. (2024). "Fault-tolerant visual servo control for a robotic arm with actuator faults". In: *Neural Computing and Applications*, pp. 1–14.

Li, Junhao, Shuhan Mou, and Dehua Zhang (2021). "A Novel Adaptive Robust Control Algorithm for Quadrotor UAV". In: *2021 6th International Conference on Robotics and Automation Engineering (ICRAE)*, pp. 50–54. DOI: 10.1109/ICRAE53653.2021.9657806.

Li, Linlin (2016). *Fault detection and fault-tolerant control for nonlinear systems*. Springer.

Li, Qin et al. (2008). "Robust passive control for uncertain time-delay singular systems". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 56.3, pp. 653–663.

Li, Xiaomeng (Nov. 2022). "A visual ergonomic assessment approach using Kinect and OWAS in real workplace environments". In: *Multiscale and Multidisciplinary Modeling, Experiments and Design* 6. DOI: 10.1007/s41939-022-00133-w.

Li, Yuxi (2017). "Deep reinforcement learning: An overview". In: *arXiv preprint arXiv:1701.07274*.

Lillicrap, Timothy P et al. (2015). "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971*.

Lin, Feng (2007). *Robust control design: an optimal control approach*. John Wiley & Sons.

Ling-Ling, Fan and Song Yong-Duan (2010). "On fault-tolerant control of dynamic systems with actuator failures and external disturbances". In: *Acta Automatica Sinica* 36.11, pp. 1620–1625.

Liu, Fuqiang, Yan Long, et al. (2021). "Active Fault Localization of Actuators on Torpedo-Shaped Autonomous Underwater Vehicles". In: *Sensors* 21.2. ISSN: 1424-8220. DOI: 10.3390/s21020476. URL: https://www.mdpi.com/1424-8220/21/2/476.

Liu, Fuqiang, Hao Tang, et al. (2022). "Review on fault diagnosis of unmanned underwater vehicles". In: *Ocean Engineering* 243, p. 110290. ISSN: 0029-8018. DOI: https://doi.org/10.1016/j.oceaneng.2021.110290. URL: https://www.sciencedirect.com/science/article/pii/S0029801821015961.

Liu, Lei, Zhanshan Wang, and Huaguang Zhang (2016). "Adaptive fault-tolerant tracking control for MIMO discrete-time systems via reinforcement learning algorithm with less learning parameters". In: *IEEE Transactions on Automation Science and Engineering* 14.1, pp. 299–313.

Ljung, Lennart (1998). "System identification". In: *Signal analysis and prediction*. Springer, pp. 163–173.

Lu, Yukun et al. (2023). "Gain-adaptive Skyhook-LQR: a coordinated controller for improving truck cabin dynamics". In: *Control Engineering Practice* 130, p. 105365. ISSN: 0967-0661. DOI: https://doi.org/10.1016/j.conengprac.2022.105365. URL: https://www.sciencedirect.com/science/article/pii/S0967066122001964.

Lussier, Benjamin et al. (2005). "Fault tolerance in autonomous systems: how and how much?" In: *4th IARP-IEEE/RAS-EURON Joint Workshop on Technical Challenges for Dependable Robots in Human Environments (DRHE)*.

Ma, Jianjun and Chi Peng (2021). "Adaptive model-free fault-tolerant control based on integral reinforcement learning for a highly flexible aircraft with actuator faults". In: *Aerospace Science and Technology* 119, p. 107204. ISSN: 1270-9638. DOI: https://doi.org/10.1016/j.ast.2021.107204. URL: https://www.sciencedirect.com/science/article/pii/S1270963821007148.

Macenski, Steven et al. (2022). "Robot operating system 2: Design, architecture, and uses in the wild". In: *Science robotics* 7.66, eabm6074.

Manhães, Musa Morena Marcusso et al. (2016). "UUV Simulator: A Gazebo-based package for underwater intervention and multi-robot simulation". In: *OCEANS 2016 MTS/IEEE Monterey*, pp. 1–8. DOI: 10.1109/OCEANS.2016.7761080.

Mareels, Iven MY et al. (1987). "Revisiting the MIT rule for adaptive control". In: *Adaptive Systems in Control and Signal Processing 1986*. Elsevier, pp. 161–166.

Marzat, J et al. (2012). "Model-based fault diagnosis for aerospace systems: a survey". In: *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* 226.10, pp. 1329–1360. DOI: 10.1177/0954410011421717. eprint: https://doi.org/10.1177/0954410011421717. URL: https://doi.org/10.1177/0954410011421717.

Maybeck, Peter S and Richard D Stevens (1990). "Reconfigurable flight control via multiple model adaptive control methods". In: *29th IEEE Conference on Decision and Control*. IEEE, pp. 3351–3356.

Meier, Lorenz et al. (2011). "Pixhawk: A system for autonomous flight using onboard computer vision". In: *2011 ieee international conference on robotics and automation*. IEEE, pp. 2992–2997.

Milhim, Aladdin, Youmin Zhang, and Camille-Alain Rabbath (Apr. 2010). "Gain Scheduling Based PID Controller for Fault Tolerant Control of Quad-Rotor UAV". In: *AIAA infotech@ aerospace 2010*, p. 3530. ISBN: 978-1-60086-963-1. DOI: 10.2514/6.2010-3530.

Mirjalili, Seyedali and Seyedali Mirjalili (2019). "Genetic algorithm". In: *Evolutionary algorithms and neural networks: theory and applications*, pp. 43–55.

Mnih, Volodymyr et al. (2015). "Human-level control through deep reinforcement learning". In: *nature* 518.7540, pp. 529–533.

Mohammadpour, Javad and Carsten W Scherer (2012). *Control of linear parameter varying systems with applications*. Springer Science & Business Media.

Montgomery, Douglas C and George C Runger (2010). *Applied statistics and probability for engineers*. John wiley & sons.

Nachum, Ofir et al. (2017). "Bridging the gap between value and policy based reinforcement learning". In: *Advances in neural information processing systems* 30.

Nagendra, Savinay et al. (2017). "Comparison of reinforcement learning algorithms applied to the cart-pole problem". In: *2017 international conference on advances in computing, communications and informatics (ICACCI)*. IEEE, pp. 26–32.

Narendra, Kumpati S and Jeyendran Balakrishnan (1997). "Adaptive control using multiple models". In: *IEEE transactions on automatic control* 42.2, pp. 171–187.

Neto, Armando Alves et al. (2015). "Attitude control for an Hybrid Unmanned Aerial Underwater Vehicle: A robust switched strategy with global stability". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 395–400. DOI: 10.1109/ICRA.2015.7139029.

Nguyen, Nhan T and Nhan T Nguyen (2018). *Model-reference adaptive control*. Springer.

Nioras, Andreas (Sept. 2020). "Survey of fault diagnosis and accommodation of unmanned underwater vehicles". In: *29th International Workshop on Principles of Diagnosis, Warsaw, Poland*.

Nise, Norman S (2020). *Control systems engineering*. John Wiley & Sons.

Noura, Hassan et al. (2009). *Fault-tolerant control systems: Design and practical applications*. Springer Science & Business Media.

Omerdic, Edin and Geoff Roberts (2004). "Thruster fault diagnosis and accommodation for open-frame underwater vehicles". In: *Control Engineering Practice* 12.12. Guidance and control of underwater vehicles, pp. 1575–1598. ISSN: 0967-0661. DOI: https://doi.org/10.1016/j.conengprac.2003.12.014. URL: https://www.sciencedirect.com/science/article/pii/S0967066103002892.

Osinenko, Pavel, Dmitrii Dobriborsci, and Wolfgang Aumer (Jan. 2022). "Reinforcement learning with guarantees: a review". In: *IFAC-PapersOnLine* 55, pp. 123–128. DOI: 10.1016/j.ifacol.2022.07.619.

Park, Jung-Hyeun et al. (2024). "Research on Clustering-Based Fault Diagnosis during ROV Hovering Control". In: *Applied Sciences* 14.12. ISSN: 2076-3417. DOI: 10.3390/app14125235. URL: https://www.mdpi.com/2076-3417/14/12/5235.

Patel, Vishakha Vijay (2020). "Ziegler-nichols tuning method: Understanding the pid controller". In: *Resonance* 25.10, pp. 1385–1397.

Patton, Ron J (1994). "Robust model-based fault diagnosis: the state of the art". In: *IFAC Proceedings Volumes* 27.5, pp. 1–24.

— (2021). "Fault-tolerant control". In: *Encyclopedia of systems and control*. Springer.

Pfister, Henry (2017). "Discrete-Time Signal Processing". In: *Lecture Note, pfister. ee. duke. edu/courses/ece485/dtsp. pdf*.

Pluymers, Bert et al. (2005). "Constrained linear MPC with time-varying terminal cost using convex combinations". In: *Automatica* 41.5, pp. 831–837.

Podder, Tarun Kanti and Nilanjan Sarkar (2001). "Fault-tolerant control of an autonomous underwater vehicle under thruster redundancy". In: *Robotics and Autonomous Systems* 34.1, pp. 39–52. ISSN: 0921-8890. DOI: https://doi.org/10.1016/S0921-8890(00)00100-7. URL: https://www.sciencedirect.com/science/article/pii/S0921889000001007.

Poli, Riccardo, James Kennedy, and Tim Blackwell (Oct. 2007). "Particle Swarm Optimization: An Overview". In: *Swarm Intelligence* 1.

Puterman, Martin L (1990). "Markov decision processes". In: *Handbooks in operations research and management science* 2, pp. 331–434.

Qu, Xingru et al. (2023). "A Deep Reinforcement Learning-Based Path-Following Control Scheme for an Uncertain Under-Actuated Autonomous Marine Vehicle". In: *Journal of*

Marine Science and Engineering 11.9. ISSN: 2077-1312. DOI: 10.3390/jmse11091762. URL: https://www.mdpi.com/2077-1312/11/9/1762.

Quamar, Md Muzakkir and Ali Nasir (2024). "Review on fault diagnosis and fault-tolerant control scheme for robotic manipulators: Recent advances in AI, machine learning, and digital twin". In: *arXiv preprint arXiv:2402.02980*.

Quigley, Morgan et al. (Jan. 2009). "ROS: an open-source Robot Operating System". In: *ICRA Workshop on Open Source Software*. Vol. 3.

Quintana, Guillem and Joaquim Ciurana (2011). "Chatter in machining processes: A review". In: *International Journal of Machine Tools and Manufacture* 51.5, pp. 363–376.

Quirynen, Rien and Stefano Di Cairano (2021). "Sequential quadratic programming algorithm for real-time mixed-integer nonlinear MPC". In: *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE, pp. 993–999.

Raffin, Antonin et al. (2021). "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268, pp. 1–8.

Ribeiro, Maria Isabel (2004). "Kalman and extended kalman filters: Concept, derivation and properties". In: *Institute for Systems and Robotics* 43.46, pp. 3736–3741.

Rizk, Hanan, Ahmed Chaibet, and Ali Kribèche (2023). "Model-based control and model-free control techniques for autonomous vehicles: A technical survey". In: *Applied Sciences* 13.11, p. 6700.

Rizvi, Syed Ali Asad and Zongli Lin (2019). "Reinforcement learning-based linear quadratic regulation of continuous-time systems using dynamic output feedback". In: *IEEE transactions on cybernetics* 50.11, pp. 4670–4679.

— (2023). *Output Feedback Reinforcement Learning Control for Linear Systems*. Springer.

Robotics, Bluefin (2006). *BPAUV during a US Navy exercise from the HSV Swift*. https://commons.wikimedia.org/wiki/File:BPAUV-MP_from_HSV-.jpg. Image via Wikimedia Commons. Licensed for reuse with attribution.

Rotondo, Damiano (2017). *Advances in gain-scheduling and fault tolerant control techniques*. Springer.

Ruder, Sebastian (2016). "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747*.

Rugh, Wilson J and Jeff S Shamma (2000). "Research on gain scheduling". In: *Automatica* 36.10, pp. 1401–1425.

Sahoo, Avilash, Santosha K Dwivedy, and PS Robi (2019). "Advancements in the field of autonomous underwater vehicle". In: *Ocean Engineering* 181, pp. 145–160.

Saied, Majd et al. (2020). "Active versus passive fault-tolerant control of a redundant multirotor UAV". In: *The Aeronautical Journal* 124.1273, pp. 385–408.

Sanner, Robert M and Jean-Jacques E Slotine (1991). "Gaussian networks for direct adaptive control". In: *1991 American control conference*. IEEE, pp. 2153–2159.

Schulman, John et al. (2017). "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347*.

Schwarting, Wilko, Javier Alonso-Mora, and Daniela Rus (2018). "Planning and decision-making for autonomous vehicles". In: *Annual Review of Control, Robotics, and Autonomous Systems* 1.1, pp. 187–210.

Sedighizadeh, Mostafa and Alireza Rezazadeh (2008). "Adaptive PID controller based on reinforcement learning for wind turbine control". In: *Proceedings of world academy of science, engineering and technology*. Vol. 27. Citeseer, pp. 257–262.

Shashoa, Nasar Aldian Ambark, Adel Saad Emhemmed, and Sulaiman Khalifa Yakhlef (2013). "Fault detection based on two-stage recursive least squares parameter estimation". In: *14th International Conference on Sciences and Techniques of Automatic Control & Computer Engineering-STA'2013*. IEEE, pp. 382–387.

Shin, Jong-Yeob and Irene Gregory (2007). "Robust gain-scheduled fault tolerant control for a transport aircraft". In: *2007 IEEE International Conference on Control Applications*. IEEE, pp. 1209–1214.

Shuprajhaa, T., Shiva Kanth Sujit, and K. Srinivasan (2022). "Reinforcement learning based adaptive PID controller design for control of linear/nonlinear unstable processes". In: *Applied Soft Computing* 128, p. 109450. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2022.109450. URL: https://www.sciencedirect.com/science/article/pii/S156849462200566X.

Siegwart, Roland, Illah Reza Nourbakhsh, and Davide Scaramuzza (2011). *Introduction to autonomous mobile robots*. MIT press.

Singh, Uttam Kumar, Jay Prakash Tripathi, and Kishore Khanna (2024). "Analysis of convergence and stability of PSO for accurate estimation of the incipient faults of hydraulic pump". In: *Journal of the Brazilian Society of Mechanical Sciences and Engineering* 46.3, p. 158.

Snedecor, G. and W. Cochran (1989). *Statistical Methods*. Iowa state University Press, Ames, Iowa.

Stengel, Robert F (1994). *Optimal control and estimation*. Courier Corporation.

Sutton, Richard S (1998). "Sutton & barto book: Reinforcement learning: An introduction". In: *A Bradford Book*. MIT Press Cambridge, MA.

Tan, Haining (2021). "Reinforcement learning with deep deterministic policy gradient". In: *2021 International Conference on Artificial Intelligence, Big Data and Algorithms (CAIBDA)*. IEEE, pp. 82–85.

Tao, Gang et al. (2013). *Adaptive control of systems with actuator failures*. Springer Science & Business Media.

Trentelman, Harry L et al. (2002). "Control theory for linear systems". In: *Appl. Mech. Rev.* 55.5, B87–B87.

Tsai, C. et al. (2021). "Multi-Sensor Fault Diagnosis of Underwater Thruster Propeller Based on Deep Learning". In: *Sensors* 21.

Umar, Abdul Aris et al. (2024). "Stability Margin of Data-Driven LQR and Its Application to Consensus Problem". In: *Mathematics* 12.2. ISSN: 2227-7390. DOI: 10.3390/math12020199. URL: https://www.mdpi.com/2227-7390/12/2/199.

Upton, Eben and Gareth Halfacree (2016). *Raspberry Pi user guide*. John Wiley & Sons.

Utkin, Vadim I (2004). "Sliding mode control". In: *Variable structure systems: from principles to implementation* 66, p. 1.

VanRossum, Guido and Fred L Drake (2010). *The python language reference*. Vol. 561. Python Software Foundation Amsterdam, The Netherlands.

Wang, Huijiao et al. (2014). "H∞ fault-tolerant control of networked control systems with actuator failures". In: *IET Control Theory & Applications* 8.12, pp. 1127–1136.

Wang, Xue-Song, Yu-Hu Cheng, and Sun Wei (2007). "A proposal of adaptive PID controller based on reinforcement learning". In: *Journal of China University of Mining and Technology* 17.1, pp. 40–44.

Wang, Zhanshan et al. (2015). "Fault-tolerant controller design for a class of nonlinear MIMO discrete-time systems via online reinforcement learning algorithm". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 46.5, pp. 611–622.

Wang, Zhao et al. (2024). "Adversarial deep reinforcement learning based robust depth tracking control for underactuated autonomous underwater vehicle". In: *Engineering Applications of Artificial Intelligence* 130, p. 107728. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2023.107728. URL: https://www.sciencedirect.com/science/article/pii/S0952197623019127.

Watson, David P and David H Scheidt (2005). "Autonomous systems". In: *Johns Hopkins APL technical digest* 26.4, pp. 368–376.

Welch, Greg and Gary Bishop (1994). "Welch & Bishop , An Introduction to the Kalman Filter 2 1 The Discrete Kalman Filter In 1960". In: *International Conference on Computer Graphics and Interactive Techniques*. URL: https://api.semanticscholar.org/CorpusID:14848980.

Wen, Fushuan and Zhenxiang Han (1995). "Fault section estimation in power systems using a genetic algorithm". In: *Electric Power Systems Research* 34.3, pp. 165–172.

Wiener, Norbert (1948). *Cybernetics or Control and Communication in the Animal and the Machine*. MIT press.

Willsky, Alan S. (1976). "A survey of design methods for failure detection in dynamic systems". In: *Automatica* 12.6, pp. 601–611. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(76)90041-8. URL: https://www.sciencedirect.com/science/article/pii/0005109876900418.

Wu, Chu-Jou (2018). "6-dof modelling and control of a remotely operated vehicle". PhD thesis. Flinders University, College of Science and Engineering.

Yaghmaie, Farnaz Adib, Fredrik Gustafsson, and Lennart Ljung (2022). "Linear quadratic control using model-free reinforcement learning". In: *IEEE Transactions on Automatic Control* 68.2, pp. 737–752.

Yahmed, Ahmed Haj et al. (2023). *Deploying Deep Reinforcement Learning Systems: A Taxonomy of Challenges*. arXiv: 2308.12438 [cs.LG]. URL: https://arxiv.org/abs/2308.12438.

Yang, Liping et al. (2008). "Study on simultaneous Fault Tolerant Control of AUV thrusters". In: *2008 IEEE International Conference on Automation and Logistics*, pp. 105–110. DOI: 10.1109/ICAL.2008.4636129.

Yin, Baoji et al. (2019). "Fault degree identification method for thruster of autonomous underwater vehicle using homomorphic membership function and low frequency trend prediction". In: *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science* 233.4, pp. 1426–1440. DOI: 10.1177/0954406218768830. eprint: https://doi.org/10.1177/0954406218768830. URL: https://doi.org/10.1177/0954406218768830.

Yu, Yong et al. (2019). "A review of recurrent neural networks: LSTM cells and network architectures". In: *Neural computation* 31.7, pp. 1235–1270.

Zadeh, S and Reza Ghasemi (2020). "Fault tolerant controller design for a class of affine nonlinear systems based on adaptive virtual actuator". In: *arXiv preprint arXiv:2003.13335*.

Zamora, Iker et al. (2016). "Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo". In: *arXiv preprint arXiv:1608.05742*.

Zhang, Dan and Bin Wei (2017). "A review on model reference adaptive control of robotic manipulators". In: *Annual Reviews in Control* 43, pp. 188–198.

Zhang, Dengsheng (2019). "Wavelet transform". In: *Fundamentals of image data mining: Analysis, Features, Classification and Retrieval*, pp. 35–44.

Zhang, Wenjie et al. (2021). "An adaptive fault-tolerant control method for robot manipulators". In: *International Journal of Control, Automation and Systems* 19.12, pp. 3983–3995.

Zhang, Youmin and Jin Jiang (2008). "Bibliographical review on reconfigurable fault-tolerant control systems". In: *Annual reviews in control* 32.2, pp. 229–252.

Zhao, Jun, Wei Wang, and Chunyang Sheng (2018). *Data-driven prediction for industrial processes and their applications*. Springer.

Zhao, Wenshuai, Jorge Peña Queralta, and Tomi Westerlund (2020). "Sim-to-real transfer in deep reinforcement learning for robotics: a survey". In: *2020 IEEE symposium series on computational intelligence (SSCI)*. IEEE, pp. 737–744.

Zhou, Kemin and John Comstock Doyle (1998). *Essentials of robust control*. Vol. 104. Prentice hall Upper Saddle River, NJ.

Zhu, Daqi, Qian Liu, and Yongsheng Yang (Nov. 2008). "An Active Fault-Tolerant Control Method Ofunmanned Underwater Vehicles with Continuous and Uncertain Faults". In: *International Journal of Advanced Robotic Systems* 5. DOI: 10.5772/6230.

ZullyC3P (2015). *Fox-C8 XT drone in flight*. Photo licensed under Creative Commons Attribution-Share Alike 4.0 International (CC BY-SA 4.0). URL: https://commons.wikimedia.org/wiki/File:Drone_Fox-C8_XT_en_vol.jpg.

Appendix A

UUV Thruster faults

The purpose of this appendix is to specify the conditions under which thruster faults can occur.

Thruster faults in AUVs can stem from various external factors. Examples of the origins of thruster faults in AUVs are provided in Table A.1.

Causes	**Associated faults and consequences**	**References**
Block of ice between the propeller blades during a mission in Antarctic.	Thruster blocking led to servo-amplifier reaching current limit mode, requiring thruster disablement.	(Caccia et al. 2001)
Flooding of a thruster.	Saltwater induced electrical dispersion, diminishing motor revolution rate feedback, resulting in blades' accelerated rotation with a time-varying impact.	(Caccia et al. 2001)
Contact with hard surfaces.	Partially or fully damaged blades can disrupt thruster motor operation, potentially causing severe internal damage.	(Abed et al. 2017)
Cavitation phenomenon.	Severe corrosion causing gradual propeller damage.	(Tsai et al. 2021)

Table A.1: Examples of natural events that caused AUV thruster faults.

AUV thruster faults induced by this variety of cases or other unexpected events from external factors are identified and grouped into different types in the literature. Table A.2

shows a list of thruster types of faults that can be found in literature, with their associated references.

Fault location	Detailed types of faults	References
Thrusters	**Fault:**	
	Blocking	(Caccia et al. 2001)
	Constant output	(Z. Chu et al. 2020)
	Partial fault:	
	Propeller damage/deformation	(Tsai et al. 2021; F. Liu, Tang, et al. 2022; F. Liu, Long, et al. 2021)
	Constant bias	(Omerdic and Roberts 2004; Falkenberg, Gregersen, and Blanke 2014)
	Time-varying bias	(Caccia et al. 2001)

Table A.2: Detailed types of AUV thruster faults due to environmental factors.

Appendix B

Linearisation of BlueROV2 Heavy model

The purpose of this appendix is to detail the method used to linearise the BlueROV2 Heavy model.

In this work, the motion of the AUV is described using two primary frames of reference: the body-fixed frame (x_B, y_B, z_B), which is rigidly attached to the vehicle, and the North-East-Down (NED) frame (x_N, y_N, z_N), which serves as a global reference with its origin (O_N) fixed at a point on the Earth's surface.

Figure B.1 illustrates the six degrees of freedom of the AUV and the transformation between the body-fixed frame and the NED frame for the BlueROV2 Heavy.

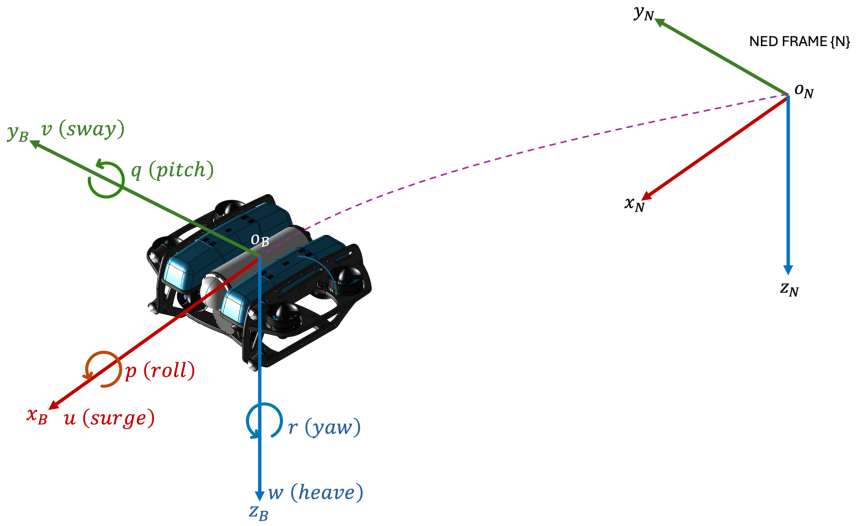


Figure B.1: BlueROV2 Heavy: Six degrees of freedom and transformation between body-fixed and NED frames.

To implement the LQR controller for our proposed approach, a complete dynamic model of the BlueROV2 Heavy is required. This model includes both kinematic and kinetic equations, which together provide a comprehensive description of the BlueROV2 Heavy's motion (Thor I Fossen 1999).

Kinematics describes the geometric aspects of motion, specifically the relationship between the ROV's velocities in the body-fixed frame and the rate of change of its positions and orientations in the NED frame. This relationship is given by:

$$\dot{\boldsymbol{\eta}} = \mathbf{J}_{\boldsymbol{\Theta}}(\boldsymbol{\eta})\boldsymbol{\nu} \tag{B.1}$$

where $\boldsymbol{\eta} = [x, y, z, \phi, \theta, \psi]^\top$ represents the position and orientation of the BlueROV2 Heavy in the NED frame, and $\boldsymbol{\nu} = [u, v, w, p, q, r]^\top$ denotes the linear and angular velocities in the body-fixed frame. The Jacobian matrix $\mathbf{J}_{\boldsymbol{\Theta}}(\boldsymbol{\eta})$ relates body-fixed velocities to the rates in the NED frame and depends on the orientation angles ϕ, θ, and ψ. In this work, we assume a block-diagonal form of $\mathbf{J}_{\boldsymbol{\Theta}}(\boldsymbol{\eta})$, which implies that linear and angular velocities are decoupled. This simplification is commonly used in marine vehicle kinematics (Thor I Fossen 1999) but may not fully capture complex hydrodynamic interactions.

The Jacobian matrix $\mathbf{J}_{\boldsymbol{\Theta}}(\boldsymbol{\eta})$ is defined as:

$$\mathbf{J}_{\boldsymbol{\Theta}}(\boldsymbol{\eta}) = \begin{bmatrix} \mathbf{R}_b^n(\boldsymbol{\Theta}_{nb}) & \mathbf{0} \\ \mathbf{0} & \mathbf{T}_{\boldsymbol{\Theta}}(\boldsymbol{\Theta}_{nb}) \end{bmatrix} \tag{B.2}$$

where $\mathbf{R}_b^n(\boldsymbol{\Theta}_{nb})$ is the standard rotation matrix converting linear velocities from the body-fixed frame to the NED frame, and $\mathbf{T}_{\boldsymbol{\Theta}}(\boldsymbol{\Theta}_{nb})$ is the angular velocity transformation matrix (Thor I Fossen 1999).

Kinetics focuses on the forces and moments responsible for producing motion, capturing their interactions with the AUV's dynamic environment. These interactions include hydrodynamic effects such as added mass, damping, and restoring forces, all of which are critical for accurately modelling the vehicle's motion. The equation of motion, which sums up all the forces and moments acting on the AUV, is formulated as follows (Thor I Fossen 1999):

$$\dot{\boldsymbol{\nu}} = \mathbf{M}^{-1}\left(\boldsymbol{\tau} - \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} - \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} - \mathbf{g}(\boldsymbol{\eta})\right) \tag{B.3}$$

where the control output vector is defined by $\boldsymbol{\tau} = [\boldsymbol{\tau}_u, \boldsymbol{\tau}_v, \boldsymbol{\tau}_w, \boldsymbol{\tau}_p, \boldsymbol{\tau}_q, \boldsymbol{\tau}_r]^\top$, $\boldsymbol{\tau}_u, \boldsymbol{\tau}_v, \boldsymbol{\tau}_w$ represent the forces in the surge, sway, and heave directions, respectively, and $\boldsymbol{\tau}_p, \boldsymbol{\tau}_q, \boldsymbol{\tau}_r$ correspond to the torques about the roll, pitch, and yaw axes, $\mathbf{M}$ is the mass matrix, $\mathbf{C}(\boldsymbol{\nu})$ is the Coriolis and centripetal matrix, $\mathbf{D}(\boldsymbol{\nu})$ is the damping matrix, and $\mathbf{g}(\boldsymbol{\eta})$ is the restoring forces and moments vector.

To fully describe the kinetics of the BlueROV2 Heavy, these matrices must be defined.

The mass matrix $\mathbf{M}$ of the BlueROV2 Heavy is determined from (Thor I Fossen 1999):

$$\mathbf{M} = \mathbf{M}_{\text{RB}} + \mathbf{M}_{\text{A}} \tag{B.4}$$

where $\mathbf{M}_{\text{RB}}$ is the rigid-body mass matrix that accounts for the vehicle's rigid-body dynamics and is given by:

$$\mathbf{M}_{\text{RB}} = \begin{bmatrix} m\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_g \end{bmatrix} \tag{B.5}$$

This expression assumes that the vehicle has a symmetric mass distribution. Additionally, the total inertia is considered as the sum of the rigid-body inertia and the added mass effects, neglecting any potential coupling terms between them.

The added mass matrix $\mathbf{M}_{\text{A}}$, which accounts for the inertia effects of the fluid surrounding the vehicle, is given by:

$$\mathbf{M}_A = -\text{diag}\left(X_{\dot{u}}, Y_{\dot{v}}, Z_{\dot{w}}, K_{\dot{p}}, M_{\dot{q}}, N_{\dot{r}}\right). \tag{B.6}$$

Here, we assume that the added mass matrix is diagonal, meaning that no coupling exists between different degrees of freedom.

The rigid-body and added-mass parameters used for the BlueROV2 Heavy are detailed in Tables B.1 and B.2.

Table B.1: Rigid-Body parameters (C.-J. Wu 2018).

Parameter	**Value**
m	11.5 kg
I_x	0.16 kg m^2
I_y	0.16 kg m^2
I_z	0.16 kg m^2

The Coriolis and centripetal matrix $\mathbf{C}(\boldsymbol{\nu})$ is expressed as (Thor I Fossen 1999):

$$\mathbf{C}(\boldsymbol{\nu}) = \mathbf{C}_{\text{RB}}(\boldsymbol{\nu}) + \mathbf{C}_{\text{A}}(\boldsymbol{\nu}) \tag{B.7}$$

where $\mathbf{C}_{\text{RB}}(\boldsymbol{\nu})$ is the Coriolis-centripetal matrix associated with the rigid-body dynamics and $\mathbf{C}_{\text{A}}(\boldsymbol{\nu})$ accounts for the Coriolis and centripetal forces arising from the added mass effects.

Table B.2: Added-Mass Parameters (C.-J. Wu 2018).

DoF	**Symbol**	**Value**
Surge	$X_{\dot{u}}$	5.5 kg
Sway	$Y_{\dot{v}}$	12.7 kg
Heave	$Z_{\dot{w}}$	14.57 kg
Roll	$K_{\dot{p}}$	0.12 kg m^2
Pitch	$M_{\dot{q}}$	0.12 kg m^2
Yaw	$N_{\dot{r}}$	0.12 kg m^2

Assuming that the body-fixed frame origin is located at the center of mass $(x_g, y_g, z_g) = (0, 0, 0)$, and that the vehicle is symmetric around its three principal axes ($I_{xy} = I_{xz} = I_{yz} = 0$), the rigid-body Coriolis and centripetal matrix becomes:

$$\mathbf{C}_{\mathrm{RB}}(\boldsymbol{\nu}) = \begin{bmatrix} \mathbf{0}_{3\times3} & -m\mathbf{S}(\boldsymbol{\nu}_1) \\ -m\mathbf{S}(\boldsymbol{\nu}_1) & -\mathbf{S}(\mathbf{I}\boldsymbol{\nu}_2) \end{bmatrix} \tag{B.8}$$

Here, $\boldsymbol{\nu}_1 = [u\ v\ w]^\top$ are the linear velocities in the body-fixed frame, and $\boldsymbol{\nu}_2 = [p\ q\ r]^\top$ are the angular velocities. The function $\mathbf{S}(\cdot)$ denotes the skew-symmetric matrix operator. The inertia matrix is denoted $\mathbf{I} = \mathrm{diag}(I_x, I_y, I_z)$, simplified under the assumption of symmetry.

Assuming a diagonal added mass matrix and relative velocity $\boldsymbol{\nu}_r = \boldsymbol{\nu} - \boldsymbol{\nu}_{\mathrm{current}}$, where $\boldsymbol{\nu}_{\mathrm{current}}$ is the velocity of the ocean current expressed in the vehicle's body-fixed frame, the added-mass Coriolis matrix is:

$$\mathbf{C}_{\mathrm{A}}(\boldsymbol{\nu}_r) = \begin{bmatrix} \mathbf{0}_{3\times3} & -\mathbf{S}(\mathbf{M}_A\boldsymbol{\nu}_{1r}) \\ -\mathbf{S}(\mathbf{M}_A\boldsymbol{\nu}_{1r}) & -\mathbf{S}(\mathbf{I}_A\boldsymbol{\nu}_{2r}) \end{bmatrix} \tag{B.9}$$

In this expression, $\boldsymbol{\nu}_{1r} = [u_r\ v_r\ w_r]^\top$ is the linear relative velocity, and $\boldsymbol{\nu}_{2r} = [p\ q\ r]^\top$ is the angular velocity. The matrix $\mathbf{M}_A = \mathrm{diag}(X_{\dot{u}}, Y_{\dot{v}}, Z_{\dot{w}})$ contains the added mass coefficients for the linear motions, while $\mathbf{I}_A = \mathrm{diag}(K_{\dot{p}}, M_{\dot{q}}, N_{\dot{r}})$ includes the rotational added inertia

terms.

The hydrodynamic damping forces acting on an AUV are modelled as a combination of linear damping $\mathbf{D}_L$, due to skin friction, and quadratic damping $\mathbf{D}_{\text{NL}}(\boldsymbol{\nu})$, resulting from vortex shedding. The total damping is given by (Thor I Fossen 1999):

$$\mathbf{D}(\boldsymbol{\nu}) = \mathbf{D}_L + \mathbf{D}_{\text{NL}}(\boldsymbol{\nu}) \tag{B.10}$$

Assuming diagonal damping and symmetry about the principal axes, the matrices are defined as:

$$\mathbf{D}_L = -\text{diag}(X_u, Y_v, Z_w, K_p, M_q, N_r) \tag{B.11}$$

$$\begin{aligned}\mathbf{D}_{\text{NL}}(\boldsymbol{\nu}) = -\text{diag}(&X_{|u|u}|u|u,\ Y_{|v|v}|v|v,\ Z_{|w|w}|w|w,\\ &K_{|p|p}|p|p,\ M_{|q|q}|q|q,\ N_{|r|r}|r|r)\end{aligned} \tag{B.12}$$

The damping coefficients used in this study were obtained from (C.-J. Wu 2018) and are presented in Table B.3.

Table B.3: Linear and Quadratic Damping Parameters (C.-J. Wu 2018).

DoF	**Linear Damping**		**Quadratic Damping**	
	Symbol	**Value**	**Symbol**	**Value**
Surge	X_u	4.03 N s/m	$X_{u\|u\|}$	18.18 N s^2/m^2
Sway	Y_v	6.22 N s/m	$Y_{v\|v\|}$	21.66 N s^2/m^2
Heave	Z_w	5.18 N s/m	$Z_{w\|w\|}$	36.99 N s^2/m^2
Roll	K_p	0.07 N s/rad	$K_{p\|p\|}$	1.55 N s^2/rad^2
Pitch	M_q	0.07 N s/rad	$M_{q\|q\|}$	1.55 N s^2/rad^2
Yaw	N_r	0.07 N s/rad	$N_{r\|r\|}$	1.55 N s^2/rad^2

The restoring force and moment vector, expressed at the centre of gravity (CG), is given by (Thor I Fossen 1999):

$$\mathbf{g}(\boldsymbol{\eta}) = \begin{bmatrix} (W_{\text{grav}} - B_{\text{buoy}})\sin(\theta) \\ -(W_{\text{grav}} - B_{\text{buoy}})\cos(\theta)\sin(\phi) \\ -(W_{\text{grav}} - B_{\text{buoy}})\cos(\theta)\cos(\phi) \\ y_b B_{\text{buoy}}\cos(\theta)\cos(\phi) - z_b B_{\text{buoy}}\cos(\theta)\sin(\phi) \\ -z_b B_{\text{buoy}}\sin(\theta) - x_b B_{\text{buoy}}\cos(\theta)\cos(\phi) \\ x_b B_{\text{buoy}}\cos(\theta)\sin(\phi) + y_b B_{\text{buoy}}\sin(\theta) \end{bmatrix} \tag{B.13}$$

The weight W_{grav} acts at the CG, while the buoyancy B_{buoy} acts at the centre of buoyancy (CB). The CB position relative to the CG in the body frame is denoted by the vector $\mathbf{r}_b = [x_b \;\; y_b \;\; z_b]^\top$. For the BlueROV2 Heavy, the CG and CB are nearly aligned, which makes roll and pitch restoring moments negligible.

The full system dynamics, combining these components, are expressed as:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \mathbf{J}_{\boldsymbol{\Theta}}(\boldsymbol{\eta})\boldsymbol{\nu} \\ \mathbf{M}^{-1}\left[\boldsymbol{\tau} - (\mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{g}(\boldsymbol{\eta}))\right] \end{bmatrix} \tag{B.14}$$

where the state vector of the BlueROV2 Heavy is defined as $\boldsymbol{x} = [\boldsymbol{\eta}, \boldsymbol{\nu}]^\top$.

Now that the nonlinear dynamic model of the system has been fully described, we can derive the expression of the state-space matrices $\mathbf{A}$ and $\mathbf{B}$ used in the LQR design. These matrices are obtained by linearising the system around an operating point corresponding to a constant reference trajectory. As described in the next section, this trajectory consists of straight-line motion along the x-axis. The matrix $\mathbf{A}$ corresponds to the Jacobian of the system dynamics with respect to the state vector, while the matrix $\mathbf{B}$ is the Jacobian with respect to the control output vector. The $\mathbf{A}$ matrix for the BlueROV2 Heavy model is a 12×12 matrix, defined by:

$$\mathbf{A} = \begin{bmatrix} \frac{\partial \dot{\boldsymbol{\eta}}}{\partial \boldsymbol{\eta}} & \frac{\partial \dot{\boldsymbol{\eta}}}{\partial \boldsymbol{\nu}} \\ \\ \frac{\partial \dot{\boldsymbol{\nu}}}{\partial \boldsymbol{\eta}} & \frac{\partial \dot{\boldsymbol{\nu}}}{\partial \boldsymbol{\nu}} \end{bmatrix}$$

And the $\mathbf{B}$ matrix is a 12×6 matrix defined by:

$$\mathbf{B} = \begin{bmatrix} \mathbf{0}_{6\times 6} \\ \\ \mathbf{M}^{-1} \end{bmatrix} \tag{B.15}$$

These expressions give us the linearised model of bluerov2 Heavy.

Appendix C

End-to-end DRL control method

In this appendix, the training environment for testing the end-to-end DRL controller is detailed, and the evaluation of this method is shown.

C.1 Environment setup

The chosen drone's mission involves reaching a designated waypoint. For the evaluation of the proposed method, the drone's initial position is set at coordinates (0, 0, -9) and the target waypoint is positioned at (1, 1, -10). Non-diagnosable thruster faults, such as propeller damage or blocking, result in a reduction of thruster efficiency. This loss of efficiency can take various forms, but is considered in this paper for simplicity as a constant multiplicative bias in the thruster's output.

While all thrusters are susceptible to faults, the initial focus is directed towards the two thrusters with the most significant impact on the mission, as has been shown experimentally, namely $T0$ and T_5. The idea is to train the agent on partial faults of these two thrusters, encompassing the cases outlined by Eq. C.1:

$$\begin{aligned} T0_f &= \alpha T0, \quad 0 \leq \alpha \leq 0.5 \\ T5_f &= \alpha T5, \quad 0 \leq \alpha \leq 0.5 \end{aligned} \tag{C.1}$$

The agent is trained with a series of episodes, with a maximum number of steps denoted as nb_step_max. At each step, a partial thruster fault is applied as indicated in Algorithm 7. An episode ends either when the waypoint is reached or when the number of steps reaches nb_step_max.

At each step, the agent chooses an action and receives a new state vector and a reward. As the AUV's goal is to reach the designated waypoint, the reward takes into account the position x, y and z of the drone. With the aim of maintaining a low rotation rate for the

Algorithm 7 Partial Thruster Fault Choice

for each episode **do**
    Randomly choose a thruster, $T0$ or $T5$
    Randomly choose α between 0 and 0.5
    Apply partial thruster fault using the chosen thruster and α
end for

drone, a constraint is imposed on the rotation speed ($\dot{\phi}$, $\dot{\theta}$, $\dot{\psi}$) to be kept close to zero. The AUV's rotation angles ϕ, θ and ψ are not considered, enabling the agent to freely choose the appropriate orientation considering the faults. The AUV's linear speed is implicitly maximised within the reward, as a longer time taken to reach the waypoint results in a lower reward. The drone's linear speed is also not explicitly taken into account. As a result, the state parameters selected are the drone's position and its rotation speed. The state vector $\mathbf{s}$ is defined in Eq. C.2. For simplicity, it is supposed that the position and rotation speed are measured accurately during the mission.

$$\mathbf{s} = \begin{bmatrix} x & y & z & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^\intercal \tag{C.2}$$

The reward function for each step is described in Algorithm 8, with x_d, y_d and z_d the desired position, *i.e.* the waypoint position, and $\dot{\phi}_d$, $\dot{\theta}_d$, $\dot{\psi}_d$ the desired rotation speeds, *i.e.* zero. The error limit below which the waypoint is considered to have been reached is denoted by ϵ.

Algorithm 8 Reward function

for each step **do**
    if nb_step > nb_step_max **then**
        End of episode
    end if
    if waypoint reached, *i.e.* $e \leq \epsilon$ **then**
        $r = 100$
        End of episode
    else
        $e = |x_d - x| + |y_d - y| + |z_d - z| + |\dot{\phi}_d - \dot{\phi}| + |\dot{\theta}_d - \dot{\theta}| + |\dot{\psi}_d - \dot{\psi}|$
        $r = -\exp(e)$
    end if
end for

The action vector of the agent is the control input $\boldsymbol{\tau}$, consisting of the forces and torques applied to the drone:

$$\boldsymbol{\tau} = \begin{bmatrix} F_x & F_y & F_z & M_x & M_y & M_z \end{bmatrix}^T \tag{C.3}$$

Action values are continuous and bounded according to the values acceptable to the drone. To measure the performance of the proposed method, a performance index γ is introduced. This index is defined as the integral of the error e over an episode, which starts at $t = 0$ and ends at $t = t_{\text{end}}$ (end of episode):

$$\gamma = \int_0^{t_{\text{end}}} e(\boldsymbol{\tau})\, d\boldsymbol{\tau} \tag{C.4}$$

C.2 Simulation Results

To assess the effectiveness of the proposed method, a comparison between a conventional PID-type algorithm and a DRL-based algorithm during fault situation is conducted through simulations, and described in this section. The evaluation of the two controllers is conducted on episodes where the fault is present throughout the entire episode. Each type of fault is evaluated over 100 episodes. The PID controller is the most widely used control structure, and it is expressed as:

$$\boldsymbol{\tau} = \mathbf{K_p} \cdot e(t) + \mathbf{K_i} \cdot \int_0^t e(\boldsymbol{\tau})\, d\boldsymbol{\tau} + \mathbf{K_d} \cdot \frac{de(t)}{dt}, \tag{C.5}$$

where $e(t)$ is the output error at time t and $\mathbf{K_p}$, $\mathbf{K_i}$, $\mathbf{K_d}$ are respectively the proportional, integral, and derivative gains. The selected gains are the default gains of the AUV Simulator package. The results of the PID controller in the presence of partial faults of thrusters $T0$ and $T5$ are presented in Table C.1. This table shows the mean of the performance index over 100 episodes as a function of the considered faulty thruster and its efficiency rate. In green are the cases where the success rate is 100%, in orange where the success rate is between 60% and 70%, and in red where the success rate is 0%. It can then be observed that, in the presence of fault, the controller mostly fails.

The agent was trained over 2400 episodes, each consisting of a maximum of 50 steps. The reward curve associated with the training is shown in Fig. C.2, and the performance results during the evaluation are presented in Table C.3.

In order to evaluate the significance of the result, a method known as the Two-Sample Mean Comparison Test is used (Snedecor and Cochran 1989). This method assesses whether two sample means differ significantly, considering their standard deviations and sample sizes. It is performed calculating a statistic test and comparing it to a critical value to determine significance (Algorithm 9).

	Thruster $T0$	Thruster $T5$
Efficiency rate: 0.5	2.69 ±0.47	2.54 ±0.37
Efficiency rate: 0.4	3.26 ±0.42	2.65 ±0.43
Efficiency rate: 0.3	4.03 ±0.25	2.83 ±0.35
Efficiency rate: 0.2	4.30 ±0.43	3.20 ±0.46
Efficiency rate: 0.1	4.54 ±0.44	3.38 ±0.43
Efficiency rate: 0.0	4.82 ±0.41	3.82 ±0.48

Figure C.1: Mean of the PID-method performance index γ over 100 episodes with the associated standard deviation.

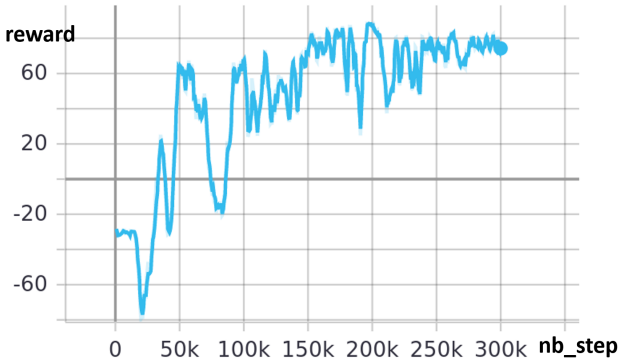


Figure C.2: Mean episodic training reward (averaged over 100 episodes) of the SAC method. The x-axis represents the number of steps, and the y-axis represents the reward value.

	Thruster T0	Thruster T5
Efficiency rate: 0.5	2.03 ±0.31	1.75 ±0.27
Efficiency rate: 0.4	2.15 ±0.34	1.82 ±0.36
Efficiency rate: 0.3	2.28 ±0.35	1.87 ±0.34
Efficiency rate: 0.2	2.56 ±0.36	2.02 ±0.32
Efficiency rate: 0.1	2.91 ±0.43	2.18 ±0.33
Efficiency rate: 0.0	4.12 ±0.52	2.43 ±0.33

Figure C.3: Mean of the SAC-method performance index γ over 100 episodes with the associated standard deviation.

Algorithm 9 Test Difference of Means

function TEST_MEANS($\mu_1, \sigma_1, n_1, \mu_2, \sigma_2, n_2, \alpha = 0.05$)
  $z_0 = \frac{(\mu_1 - \mu_2)}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}$
  $z_{\frac{\alpha}{2}} \cong 1.96$
  if $z_0 > z_{\frac{\alpha}{2}}$ or $z_0 < -z_{\frac{\alpha}{2}}$ **then**
    return True ▷ Significant difference
  else
    return False ▷ No significant difference
  end if
end function

This algorithm is applied to the results shown in Tables C.1 and C.3 to compare the PID controller and the DRL-based controller. It demonstrates that the DRL-based controller leads to a significant improvement for all the fault cases considered. Table C.1 shows the result of $z_0 - z_{\frac{\alpha}{2}}$ for all fault cases.

	Thruster T0	Thruster T5
Efficiency rate: 0.5	9.76	15.29
Efficiency rate: 0.4	18.58	12.85
Efficiency rate: 0.3	38.72	17.70
Efficiency rate: 0.2	29.06	19.09
Efficiency rate: 0.1	24.52	20.18
Efficiency rate: 0.0	8.61	21.90

Table C.1: Calculation of the Two-Sample Mean Comparison Test $z_0 - z_{\frac{\alpha}{2}}$ for all fault cases, to compare the PID controller and the DRL-based controller. Positive values demonstrate the significance of the result.

To better understand the results, the AUV trajectories were presented in three different scenarios: the first involves no fault and is controlled by the PID (Fig. C.4), the second simulates a fault in thruster $T5$, reducing its efficiency to 0.2, while still using the PID control (Fig. C.5), and finally, the third scenario features the same thruster fault but uses the DRL-based control method (Fig. C.6).

The Fig. C.5 shows that when the AUV is under PID control and experiences the

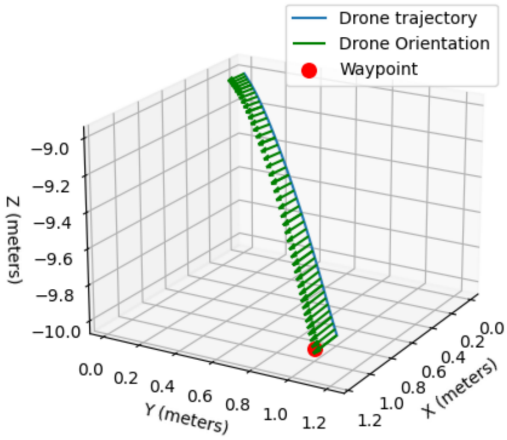


Figure C.4: AUV trajectory controlled by PID in the absence of fault.

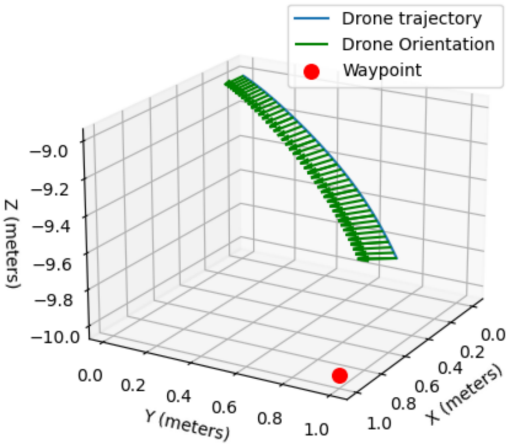


Figure C.5: AUV trajectory controlled by PID, where $T5$ efficiency rate is 0.2%.

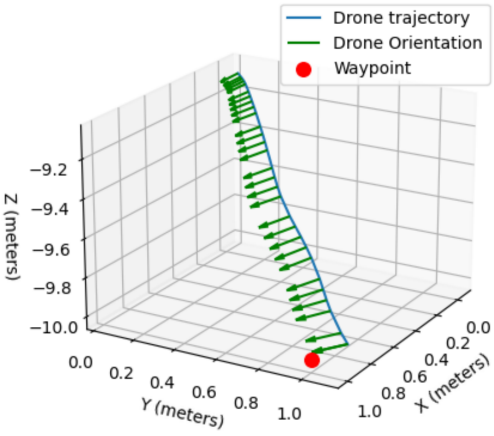


Figure C.6: AUV trajectory controlled by the DRL-based controller, where $T5$ efficiency rate is 0.2%.

considered thruster fault, it fails to reach the designated waypoint: it can be observed that the drone is unable to descend sufficiently along the z-axis to reach the waypoint, and that the trajectory deviates along the x and y axis. This is attributed to the fact that thruster $T5$ is a vertical thruster, involved in both the movement along the z-axis and the rotation of the drone.

In contrast, the DRL-based method enables it to successfully navigate to the waypoint, as shown in Fig. C.6, even if the trajectory does not follow a straight line. This can be interpreted by the fact that the DRL-based controller employs dynamic strategies to maintain control and compensate for the effects of the faulty thruster.

These results can be found in the paper published for the ANZCC 2024 conference as part of the work on this thesis (Lagattu, Le Chenadec, et al. 2024).

However, a purely machine learning-based controller has limitations. Figures C.7

and C.8 show the evolution of the control input forces over time, using a classical PID controller and an end-to-end DRL controller, respectively. It can be observed that, unlike the smooth control inputs produced by the PID law, the control inputs generated by the DRL model exhibit very large and chaotic variations. This behaviour is due to the fact that the DRL agent optimises its actions solely based on the reward signal, without any built-in consideration for physical constraints such as actuator dynamics or energy consumption. As a result, applying an end-to-end DRL controller directly on a physical drone is hardly feasible, since the thrusters are not designed to respond to such rapid and aggressive changes, and the resulting control commands also lead to excessive energy consumption. It would be possible to include constraints on the variation of actions within the reward function, or to apply post-processing to smooth the control inputs, in order to make the deployment of the end-to-end DRL controller on a physical drone more realistic. However, such approaches still offer no guarantee of closed-loop stability.

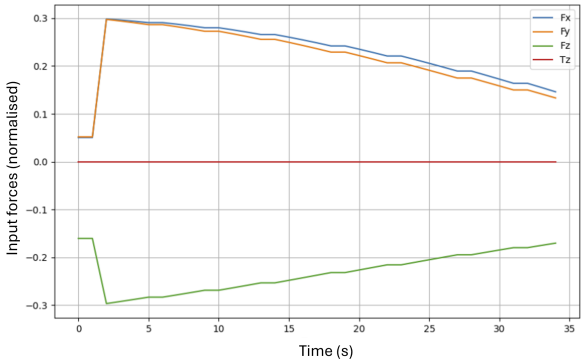


Figure C.7: Evolution of control input forces with a PID controller, without fault.

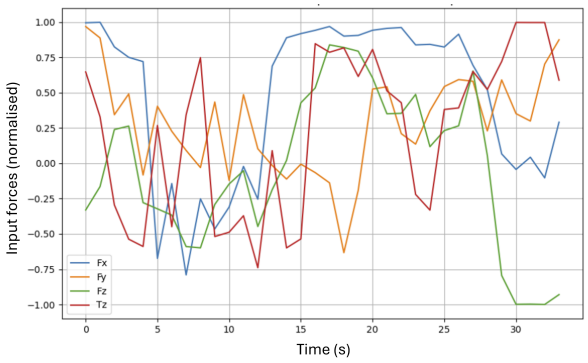


Figure C.8: Evolution of control input forces with an end-to-end DRL controller, without fault.

COLLEGES SCIENCES
BRETAGNE POUR L'INGENIEUR
LOIRE ET LE NUMERIQUE

Titre : Stratégie de contrôle pour la gestion de pannes d'actionneurs par apprentissage par renforcement profond – Application aux véhicules sous-marins autonomes

Mot clés : Contrôle tolérant aux pannes, panne d'actionneur, réallocation des forces, apprentissage par renforcement profond, véhicule sous-marin autonome, contrôle basé sur un modèle

Résumé : Les systèmes dynamiques autonomes, tels que les drones d'exploration, sont vulnérables aux pannes d'actionneurs, qui peuvent compromettre leur fonctionnement ou entraîner une perte totale du système. Il est donc essentiel de mettre en place une stratégie de contrôle tolérant aux pannes (FTC) afin de maintenir les performances et la stabilité du système en cas de panne d'actionneur.

Les approches FTC classiques reposent généralement sur la réallocation des forces de contrôle, consistant à redistribuer les efforts désirés entre les actionneurs restants. Cette réallocation nécessite un diagnostic explicite de la panne, c'est-à-dire l'identification en temps réel de son origine et de son intensité. Toutefois, ce diagnostic devient difficilement réalisable en présence de pannes multiples ou lorsque les ressources de calcul sont limitées. Ces limites ont conduit à l'exploration de l'apprentissage par renforcement profond (DRL) pour le contrôle des systèmes. Les stratégies FTC basées sur le DRL proposées dans la littérature visent à générer directement les commandes des actionneurs, permettant ainsi au système de s'adapter de manière autonome aux pannes. Le DRL présente une forte capacité d'adaptation aux pannes d'actionneurs, en s'affranchissant de tout diagnostic explicite. Cependant, un contrôle basé uniquement sur du DRL ne garantit pas la stabilité du système, en raison de l'imprévisibilité des sorties du modèle de DRL, ce qui constitue un défi majeur.

Cette thèse s'attaque à cette problématique en cherchant à démontrer que la combinaison du DRL avec un contrôle basé sur le modèle du système améliore à la fois les performances et la stabilité d'un système dynamique autonome soumis à une panne d'actionneur, en comparaison avec les méthodes existantes. Pour ce faire, la méthodologie adoptée consiste à effectuer la réallocation des forces à l'aide d'une méthode de DRL dans le cadre d'un contrôleur basé sur le modèle du système, en associant l'adaptabilité du DRL aux garanties de stabilité offertes par les approches classiques basées sur le modèle du système. Le DRL permet de redistribuer les efforts en adaptant dynamiquement la matrice d'allocation, tandis que la stabilité locale est assurée par le recalcul des gains de contrôle à partir du modèle du système modifié. Entraîné en simulation sur un ensemble de scénarios de pannes variés, le modèle DRL assure une réallocation des forces en temps réel lorsqu'une panne survient.

La méthode est appliquée à un BlueROV2 Heavy, un véhicule sous-marin autonome doté de plusieurs propulseurs, dans des expériences menées à la fois en simulation et en conditions réelles. Les résultats mettent en évidence la pertinence de la réallocation du contrôle basée sur le DRL pour faire face aux pannes de propulseurs, surpassant les méthodes adaptatives qui ne s'appuient pas sur le modèle du système ou qui ne s'appuient pas sur le DRL. Par ailleurs, ils soulignent la capacité du modèle à généraliser, en parvenant à gérer des scénarios de panne qui n'ont pas été rencontrés lors de l'apprentissage.

Ces résultats soutiennent une approche

prometteuse du FTC pour la tolérance aux pannes d'actionneurs, combinant l'adaptabilité du DRL à la stabilité du contrôle basé sur un modèle, sans besoin de diagnostic explicite des pannes.

Title: Actuator fault recovery control strategy with Deep Reinforcement Learning - Application to Autonomous Underwater Vehicles

Keywords: Fault-tolerant control, actuator fault, control reallocation, deep reinforcement learning, autonomous underwater vehicle, model-based control

Abstract: Autonomous dynamic systems, such as exploration drones, are at risk from actuator faults, which can compromise the operation or lead to a system loss. This is why it is necessary to establish a Fault-Tolerant Control (FTC) strategy to maintain system performance and stability under actuator faults. Traditional FTC methods commonly use control reallocation, redistributing forces among remaining actuators. Control reallocation requires explicit fault diagnosis, *i.e.* identifying the fault's origin and intensity in real time. Explicit fault diagnosis becomes impractical with multiple faults or limited computational capacity. These limitations have led to the exploration of Deep Reinforcement Learning (DRL) for system control. The idea of existing DRL-based FTC strategies in the literature is to leverage DRL to directly generate control inputs for the actuators, allowing the system to autonomously adapt to faults. DRL offers high adaptability for actuator fault recovery, enabling the control strategy to operate without explicit fault diagnosis. However, DRL-based control lacks stability guarantees due to the unpredictability of DRL model outputs, which poses a challenge in guaranteeing stability of the system. This thesis addresses this challenge by aiming to demonstrate that the combination of DRL and model-based control improves the performance and stability of an autonomous dynamic system subject to an actuator fault compared to existing methods. To support this statement, the chosen methodology involves performing control reallocation using DRL within a model-based control framework, integrating DRL's adaptability with the stability benefits of model-based methods. DRL redistributes forces by adjusting the Control Allocation Matrix, with local stability ensured through recalculated control gains in a model-based framework. Trained in simulation to address varied fault scenarios, the DRL model provides real-time control reallocation in the case of faults. This method is applied to a BlueROV2 Heavy, which is an autonomous underwater vehicle equipped with multiple thrusters, in both simulation and real-world experiments. Results demonstrate the effectiveness of DRL-based control reallocation for thruster faults in the BlueROV2 Heavy, outperforming non-model-based and non-DRL adaptive methods. Additionally, the results highlight the DRL model's generalisation capacity, as it successfully manages fault scenarios not encountered during training. These findings support a promising FTC approach to actuator fault tolerance that combines DRL's adaptability with the stability of model-based control, without any need for explicit fault diagnosis.