MODELING OF A MACHINE LEARNING-BASED VIRTUAL COPILOT FOR HELICOPTERS

Stefano Cecchi,
TXT E-TECH, Cologno Monzese, Italy
Elisa Capello,
Politecnico di Torino Department of Mechanical and Aerospace Engineering, CNR-IEIIT, Torino, Italy
Gianluca Parnisari,
TXT E-TECH, Cologno Monzese, Italy

Abstract

Key challenges in the aviation industry are mainly related to increasing safety and efficiency. With the rise of Artificial Intelligence (AI), new technologies enable on-board systems to adapt to new tasks and to perform them without programming for different scenarios. This research focuses on the design of a virtual copilot for helicopters, to assist the pilot and to reduce workload. Three critical use cases are investigated: predicting Vortex Ring State (VRS), detecting engine malfunctions, and aiding pilots during autorotation scenarios. Each of these presents unique challenges, so different machine learning techniques are analyzed to choose the best fit for each use cases: a Supervised Learning algorithm is developed to predict VRS, providing timely warnings for proactive mitigation; engine malfunctions are addressed using an Unsupervised Learning assistant to detect subtle performance deviations; Reinforcement Learning and Imitation Learning algorithms are explored to design a virtual assistant for supporting during autorotative descent.

1. NOTATION

Symbols

V_d	Descent velocity
V_h	Rotor induced velocity at hover
V_x	Horiziontal velocity
V_z	Vertical velocity
T	Thrust
C_T	Mean thrust coefficient
$C_{T,h}$	Thrust coefficient at hover
NG or N1	Gas Generator velocity
NR	Rotor speed
NF or N2	Turbine speed

Acronyms:

AI	Artificial Intelligence
AFCS	Automatic Flight Control System
AGL	Above Ground Level
APU	Auxiliary Power Unit
CFD	Computational Fluid Dynamics
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DDP	Differential Dynamic Programming
DDPG	Deep Deterministic Policy Gradient
EASA	European Union Aviation Safety Agency
FDR	False Discovery Rate
FOD	Foreign Object Damage
GADGHT	Ground/Air Display for Geographical/Hydrographic Tracking
HUMS	Health and Usage Monitoring System
kNN	K-Nearest Neighbors
LQR	Linear Quadratic Regulator
MDP	Markov Decision Process
ML	Machine Learning
MRMR	Minimum Redundancy Maximum Relevance
MSE	Mean Squared Error
NMPC	Nonlinear Model Predictive Control
NN	Neural Networks
OEI	One Engine Inoperative
RBF	Radial Basis Function
ReLU	Rectified Linear Unit
RFM	Rotorcraft Flight Manual
RL	Reinforcement Learning
ROD	Rate of Descent

RVM	Ring Vortex Model
SL	Supervised Learning
SVM	Support Vector Machine
TAS	True Airspeed
TCAS	Traffic and Collisions Avoidance System
UL	Unsupervised Learning
VRS	Vortex Ring State

2. INTRODUCTION

Controlling a helicopter requires significant effort and constant attention. Some tasks' workload is high, and performance could decline, leading to unpleasant consequences. According to the study performed by the *United States Army Aeromedical Research Laboratory* (Ref.1), most pilots have established that responding to emergencies is the task with the higher cognitive workload. Thus, increasing awareness of workload-related errors is a major influence on cockpit design. Several analogical and digital assistants, such as the *Ground Proximity Warning System* or the *Traffic and Collisions Avoidance System* (TCAS) respectively, are already implemented on board to support the pilot in the decision-making process reducing workload and stress.
In this sense, AI can provide solutions by operating through understanding data, identifying hidden correlations in this data, anomaly detection, and discovery of any unusual evolutions.

Although AI is often intended as a system itself, it is a set of technologies implemented in a system to enable it to reason, learn, and act to solve a complex problem. Traditional AI systems rely on rule-based approaches where explicit rules and logic are programmed to guide decision-making. The subject of this work deals with an AI subfield called *Machine Learning* (ML), which provides the learning part of AI. It consists of the building of algorithms for optimizing a performance criterion, using example data or experience. In other words, it is the process of solving a practical problem by gathering a dataset and algorithmically building a statistical model based on that dataset. Machine learning can be divided into three main groups: *Supervised Learning* (SL), *Unsupervised Learning* (UL), and *Reinforcement Learning* (RL).

ML is a solution to deal with real-time data flows and enables real-time risk management. Its applications, also for other fields, go from natural language processing to computer vision and beyond.
The role of an ideal machine learning-based virtual assistant will be similar to the one of a second pilot: it may be capable of reevaluating the current situation constantly as this is dynamically evolving, keeping the pilot updated only with the precise information he needs to manage the situation, allowing him to focus on fly the aircraft safely and handle other critical tasks; it could suggest typical or exceptional procedures, following the *Rotorcraft Flight Manual* (RFM), and could aid the pilot during the flight if any deviations occur, proposing alternative solutions; it could also anticipate some critical situations so that the pilot can act accordingly with anticipation.
This research intends to show different examples of human cognitive assistance in decision-making and action selection, and Human-AI teaming (first two of the EASA levels in Ref.2), dealing with three significant use cases (predicting *Vortex Ring State* (VRS), detecting engine malfunctions, and aiding pilots during autorotation scenarios) and applying the different machine learning paradigms. The use cases choice was made based on the content of the AW189 RFM. Even though the reliability of helicopter technology has significantly improved, systems and component failures or malfunctions still occur, thus primary concern was given to seek possible applications between emergencies and systems' malfunctions.
The virtual copilot is trained in Matlab-Simulink, and some simulations will be performed to show the effectiveness of the proposed solution.

The starting points of the project were Ref.3, Ref.4, and Ref.5, which report three different ways of using artificial intelligence in the aeronautical sector, in particular for fixed-wing aircraft.
Ref.3 reports the design of an intelligent tutoring system, used in simulation training to identify mistakes or suboptimal maneuvers of trainee pilots and suggests corrective actions. The algorithm is trained with a Supervised Learning technique named *Behavioral Cloning*, which consists of learning the machine to mimic qualified flight instructors' behavior based on a large dataset of example maneuvers. Ref.4 has roughly the same goal: using artificial intelligence to provide real-time feedback on the quality of each flight maneuver to student pilots for early-stage learning. The paper reports the different phases encountered in designing a *Maneuver Identification system* using Supervised Learning too.
Ref.5 is a project about an intelligent assistant (*Harvis*) composed of two parts: a *Non-Stabilized approach assistant* and an *Aircraft Dynamic re-routing assistant*. The former uses an eye-tracking-based algorithm to announce parameter deviations in the approach stabilization phase and perform decision-making support by giving a second opinion to the pilot on the situation. The latter is an automated re-routering planner activated in some high-workload situations to reduce the pilot workload.

The paper is organized into nine sections. After Section 2 has introduced the topic, Sections 3 to 8 focus on the development of the machine learning algorithm for specific use cases. Each of these sections also includes a brief review of relevant literature. Finally,

Section 9 concludes the paper by summarizing the findings and outlining potential areas for future research.

3. VORTEX RING STATE

During vertical descent, the vertical velocity (V_d) is opposite to the rotor induced velocity. The meeting of flow lines and induced velocity leads to the creation of vortices: for low rates of descent ($-1 \leq {}^{V_d}/_{V_h} \leq 0$), where V_h is the induced velocity in hover), recirculation and turbulence are contained and isolated in the downstream tube (Figure 1).

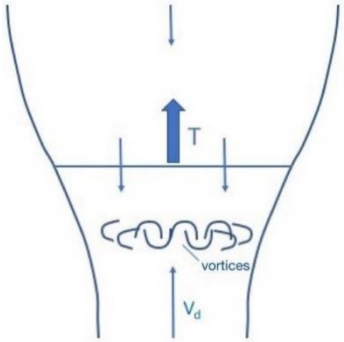


Figure 1: Normal working state (Ref.25).

As the vertical velocity increases, this behavior becomes increasingly important, and the vortices move towards the rotor. Because flow is unsteady, the continuity assumption on which the momentum theory is based breaks down. For ${}^{V_d}/_{V_h} \cong -1$, large recirculation and high turbulence near the disk bring high vibration levels and degraded control. Vortex rings encircling the rim of the disk, doughnut fashion, appear (Figure 2).

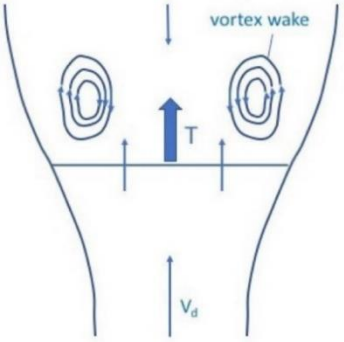


Figure 2: Vortex Ring State (Ref.25).

For this reason, this regime is called Vortex Ring State. Since momentum theory is not valid, computations of actual parameters are complicated. Thus, there is no clear-cut value for the transition velocity, which indicates when this state is encountered and currently, a system capable of warning when a rotary aircraft will enter VRS does not exist. In order to correctly conceptually design an algorithm that does so, understanding and detecting VRS boundaries in helicopter operations is of paramount importance due to its risky nature. Several models and criteria have been developed to predict and define these boundaries. Since momentum theory cannot explain the inflow model, most of the theories about the vortex ring describe it empirically. The research carried out on these allowed to select those parameters that can be used as features in the prediction algorithm.
The focus of this paper is the main rotor Vortex Ring State. Also, the tail rotor can encounter this condition, but it will lead to different considerations.

3.1. VRS condition and influential factors

Vortex Ring State is a hazardous flight condition that could occur for any rotor-based aircraft at any altitude when it is descending at low forward speed with a vertical velocity approaching the wake-induced velocity at the rotor disk, causing the aircraft to descend into its own downwash. During VRS, the rotor tip vortices fail to dissipate rapidly enough, leading to the accumulation and periodic breakaway of the wake. This results in the formation of a vortex ring, creating a circulating flow downward through the rotor disk and then outward and upward beyond it. Aerodynamically, the onset of the Vortex Ring State signifies the breakdown of the rotor wake's orderly structure into a turbulent, recirculating flow. In this unsteady flow, wake vorticity generated by the rotor blades accumulates near the rotor plane, resulting in the formation of large, irregular airloads, generating significant low-frequency vibrations and potential control issues, thereby increasing pilot workload and posing a serious safety risk.
In general, VRS may manifest in helicopter main rotors under the following conditions:

- At low speeds combined with steep descent angles.
- When descending downwind into a landing area.
- In the final stages of a quick stop maneuver, often executed in proximity to the ground.
- During power recovery initiation following engine power-off.
- When settling into the downwash of another aircraft.

Several studies have identified various symptoms indicative of a helicopter entering a Vortex Ring State. Castles and Gray (Ref.6) delved into induced velocity across multiple rotor configurations, highlighting significant fluctuations in its distribution.
Yaggy and Mort (Ref.7) examined rotor thrust during descent, uncovering substantial oscillations within the VRS.
Empey and Ormiston (Ref.8) carried out wind tunnel tests using a 1/8-scale helicopter model, detecting notable thrust oscillations under descent conditions.

Xin and Gao (Ref.9) conducted whirling beam tests under both axial and inclined descent conditions, observing remarkable fluctuations in rotor thrust and torque.
In the work by Chenglong et al. (Ref.10), it was observed that surpassing a specific threshold in descending speed triggers yaw oscillation. Concurrently, the roll and pitch within the inner loop exhibit extensive oscillation, culminating in a shock to divergence. These occurrences potentially signify the proximity of the rotorcraft to, or its entry into, the early stages of VRS, leading to diminished stability under such circumstances.
Additional effects noted in experiments and real-world scenarios encompass reduced airspeed, heightened rate of descent, and excessive loss of altitude.

VRS can arise unexpectedly and can be perilous if not promptly addressed. It typically manifests within certain descent velocities but ceases once forward velocity surpasses a certain threshold, causing the vortex to dissipate. Therefore, it is crucial to establish a safe vertical descending speed limitation. Recovery from Vortex Ring State should be initiated at the first indication by promptly applying forward cyclic input to increase airspeed and/or simultaneously reducing collective pitch to orient the rotor blades toward cleaner, less turbulent airflow. Combining lateral cyclic thrust with increased power and lateral antitorque thrust facilitates the swiftest exit from this regime, a technique commonly referred to as the *Vuichard Recovery*. Maintaining a reasonable gliding angle can also serve as a preventive measure against entering VRS. However, if VRS progresses to the windmill brake state, autorotation may become the only viable recovery maneuver.
Importantly, the Vortex Ring State is not solely dictated by rotor operating conditions like descent rate and forward speed but is also influenced by rotor design details. Modern helicopters and tiltrotors, characterized by higher disk loadings, may be more prone to vortex ring state across a wider range of low-speed flight conditions. Additionally, the distribution of thrust along the length of the rotor blades significantly affects wake stability, primarily controlled by blade twist. It's proposed that modern rotor designs with pronounced blade twists could extend the vortex ring state regime to cover a broader range of operational descent velocities and higher forward speeds compared to rotors with less twist.

3.2. Predicting VRS

Over the years, various criteria have been proposed for determining Vortex Ring State boundaries, each leading to different predictions. These boundaries demarcate the regions where the effects of VRS are pronounced, based on the selected criterion.

As can be understood from what was said previously, one common method of representing VRS boundaries is by utilizing freestream velocity components, V_x and V_z (horizontal and vertical respectively), normalized by hover-induced velocity V_h. Another frequently used variable is the descent angle. Figure 3 illustrates a traditional graph depicting the flight envelope alongside its constraints, including the autorotative boundary.

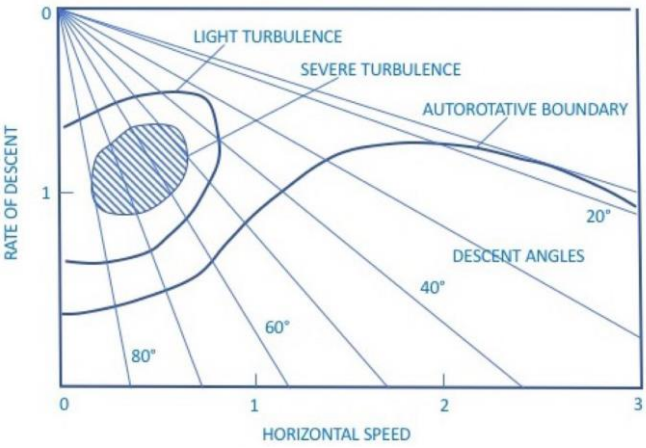
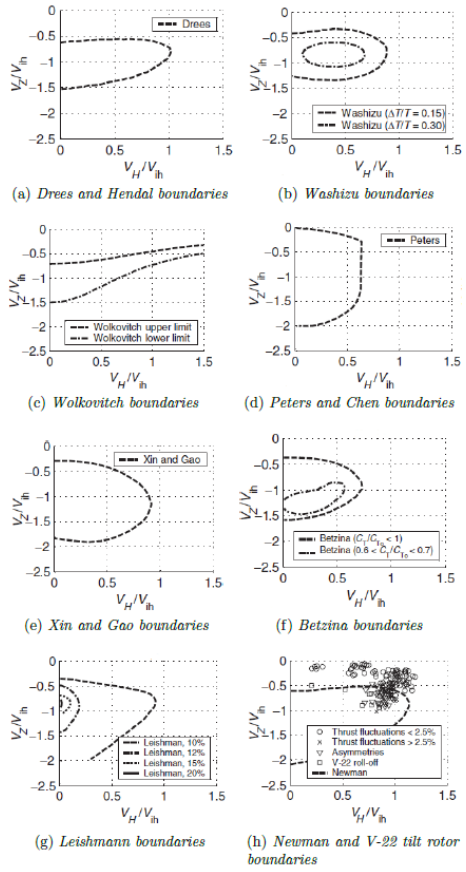


Figure 3: Flight envelope and its limits for velocity components and glide angle (Ref.25).

The determination of factors and conditions affecting the initiation of VRS on rotors cannot be reliant only on descent velocity and descent angle. Figure 4 reports various additional criteria developed for delineating VRS boundaries, all outlined below.
Drees and Hendal (Ref.11) identified a region of roughness, where rotor behavior exhibited roughness concerning attitude and control. This region also witnessed unexpected altitude loss and significant nose-down pitching motion.
Using wind tunnel tests, Washizu (Ref.12) defined the boundary based on thrust fluctuations ($^{\Delta T}/_{T}$), highlighting changes in lift generation during VRS.
Wolkovitch (Ref.13) analytically predicted the VRS boundary in descent conditions by considering a model comprising a slipstream with uniform flow, surrounded by a protective tube of tip vortices leaving the rotor. The onset of VRS was associated with a breakdown of this protective tube, occurring when the net velocity of tip vortices became zero. Wolkovitch's boundaries were close to those of experimental data at low forward speed; nevertheless, they were not consistent with other experiments, as VRS could be encountered at any forward speed. Peters and Chen (Ref.14) refined Wolkovitch's idea by addressing flow model inconsistencies and considering wake skew angle, improving boundary prediction accuracy.
Xin and Gao (Ref.9) observed irregular rotor torque variations at certain descent velocities and proposed an improved VRS boundary consistent with flight and model test data. They found that as the descent angle

decreased, the torque fluctuations also decreased and finally disappeared below a certain level.
Betzina (Ref.15) observed a significant reduction in the mean thrust. For that reason, he used the thrust ratio $C_T/C_{T,h}$ as an indicator for VRS, where C_T and $C_{T,h}$ represented the mean thrust coefficient and the thrust coefficient at hover, respectively.
Leishman et al. (Ref.16) focused on excessive blade-flapping fluctuations caused by steady airloads near or in VRS, suggesting them as a warning sign.
Newman (Ref.17) developed a wake transport criterion for VRS assessment, defining an effective wake transport velocity with a critical value indicating flow breakdown onset in the wake stream tube.
In (Ref.18), (Ref.19) the testing of a tilt-rotor aircraft was conducted to determine the criteria for a quasi-steady state VRS boundary. As the descent rate increased from hover, the test team observed thrust fluctuations followed by an uncommanded roll response defining the VRS boundary.
The VRS boundary and its severity are influenced by factors discussed earlier. Moreover, as the vehicle mass increased it is possible to see that the size of the regime, where VRS is encountered, shifts downwards, and grows dimensionally.



(a) *Drees and Hendal boundaries* (b) *Washizu boundaries*
(c) *Wolkovitch boundaries* (d) *Peters and Chen boundaries*
(e) *Xin and Gao boundaries* (f) *Betzina boundaries*
(g) *Leishmann boundaries* (h) *Newman and V-22 tilt rotor boundaries*

Figure 4: Summary of VRS boundaries from the literature (Ref.24).

An alternative method for predicting VRS is explored in (Ref.20). This study seeks to develop a concise method for detecting the onset of VRS before it occurs. Its starting point is the previous, already exposed literature, in which distinctive fluctuations in different parameters, like pressure at specific rotor points, were noted for a rotorcraft descending into VRS. These fluctuations could potentially be utilized to design and implement a VRS warning system for rotary aircraft. Wind tunnel experiments have been designed to measure rotor lift as it descends into VRS. *Computational Fluid Dynamics* (CFD) will then be employed to confirm conclusions drawn from wind tunnel experiments and to investigate the presence of pressure fluctuations at different rotor points.
The results, depicted in Figure 5, show an apparent discontinuity in lift occurring between approximately 8 and 9.5 seconds when lift starts to decrease before increasing again. This discontinuity aligns with the formation and arrival of a vortex at the rotor plane of the model helicopter. The vibration frequency experienced during these experiments is within the low-frequency range. Notably, these continuity changes vanished when the wind tunnel was not operational (preventing VRS), solidifying the link between this phenomenon and VRS entry. Ultimately, this investigation suggests the potential for predicting when a model helicopter will enter VRS by monitoring its lift in real time and using detected lift discontinuities from wind tunnel experiments as triggers for a VRS warning system. While obtaining lift data in real-world scenarios is challenging, this research presents a promising starting point for further exploration.

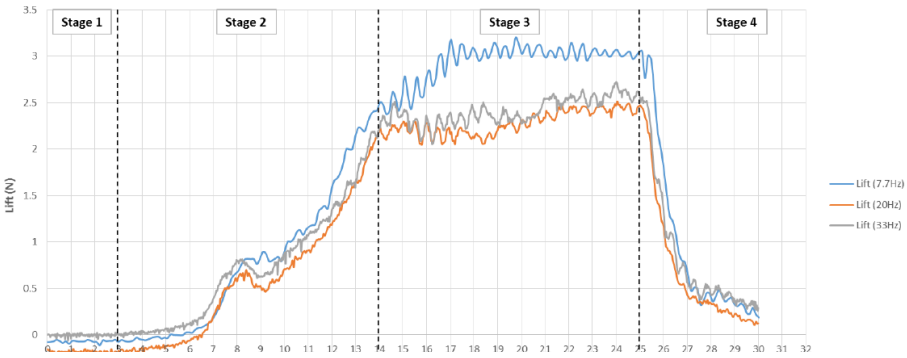


Figure 5: Lift function of time from Ref.20.

The only instance concerning a VRS warning system found is detailed in Varnes' thesis (Ref.21). This work focused on designing an algorithm that could predict VRS and provide a timely warning to pilots. The system would be integrated into the *Ground/Air Display for Geographical/Hydrographic Tracking* (GADGHT) unit, which would display both audible and visual alerts when the helicopter approaches VRS conditions. The author evaluated different criteria for defining VRS boundaries, including those proposed by Wolkovitch, Peter and Chen, and Gao and Xin. Ultimately, Gao and Xin's criterion was chosen because

it encompassed a larger region of potential VRS based on flow visualization data. This approach offered a more realistic basis for the warning system. The study also investigated using vibrations as an indicator of VRS.

4. MACHINE LEARNING FOR PREDICTING VRS

The design virtual assistant for predicting VRS intends to prevent entry into this regime, advising the pilot when certain parameters indicate the proximity to the VRS boundary conditions, enabling proactive mitigation strategies.
For this purpose, Supervised Learning was deemed the best choice. After collecting several significant data on the phenomenon in a certified flight simulator, a classification algorithm could be trained to recognize whether the helicopter is in VRS, close to VRS, or far from it, advising the pilot and enabling proactive mitigation strategies.

4.1. Supervised Learning

Supervised Learning is a fundamental paradigm within machine learning where the algorithm is trained on a labeled dataset. In the SL approach, the algorithm learns to map input data to corresponding output labels, mimicking the relationship between input and output pairs provided in the training data. The overarching goal is to generalize this learned mapping to make predictions on new, unseen data accurately. Inputs are also called *features*, and choosing the most important is a fundamental problem in ML. Outputs correspond with labels, which can be elements dividing features into categories (or classes). The algorithms that perform what has just been described fall into the category of classification problems.
Some of the most important algorithms that will be considered later are now explained briefly:

- *Decision Tree* learning recursively partitions the data into subsets based on the values of input features. It creates a tree-like structure to make decisions, making it highly interpretable. A decision tree consists of *nodes*, where each node represents a decision based on the value of a specific feature. Nodes are connected by branches, leading to subsequent nodes or leaves. In each branching node, a specific feature is examined. The algorithm selects the best feature and value to split the data at each node. Different splitting criteria exist based on features. The process continues until a stopping criterion is met, such as a predefined tree depth, or when the leaf node is reached, then the decision is made. The deeper the tree, the more complex the decision rules and the more accurate the model. It's a simple and easy-to-interpret method, but it has some limitations, such as generalizing the data well and overfitting[2]. It could be improved with fine-tuning.
- *K-Nearest Neighbors* (kNN) is a non-parametric, instance-based learning algorithm that makes predictions based on the majority class or average of the k-nearest data points in the feature space; that is, once a new example *x* comes in, the kNN algorithm finds *k* training examples closest to x and returns the majority label, where the closeness is given by a distance function, as Euclidean distance. The parameter *k* is a critical hyperparameter[3] in kNN: smaller values make the model more sensitive to noise, while larger values may oversmooth the decision boundary. kNN makes no assumptions about the underlying data distribution, making it suitable for a wide range of scenarios. Anyway, kNN is a lazy learner, meaning it does not explicitly build a model during training. The algorithm stores the entire training dataset and uses it at prediction time. For this reason, it can be computationally expensive and slow at predicting.
- *Naive Bayes* is a simple, yet effective probabilistic algorithm based on *Bayes' theorem*, which calculates the probability of a hypothesis based on observed evidence. It is stated mathematically as the following equation:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)} \quad (1)$$

where p(A|B) is the probability of event A occurring given the event B. It assumes that features are conditionally independent given the class, leading to its "*naive*" designation. While this assumption may not hold in reality, the algorithm often performs well in various classification tasks. During training, the algorithm estimates the probabilities needed for Bayes' theorem using the training dataset. For each class, it calculates the *prior*

[2] Overfitting is the property of a model such that this works well using the training sample but frequently makes errors when applied to examples that weren't seen by the learning algorithm during training.
[3] A hyperparameter is a configuration setting used to control the learning process. Unlike model parameters, which are learned from the training data, hyperparameters are set by the designer before the training process begins. The process of finding the optimal values for hyperparameters is known as tuning.

probability (p(A)) and the *conditional probabilitie*s (p(B|A)) of each feature given the class. To classify a new instance, it calculates the *posterior probability* (p(A|B)) for each class and selects the class with the highest probability. When dealing with continuous data, a typical assumption is that the continuous values associated with each class follow a normal (or Gaussian) distribution. A strong limitation is that the algorithm requires knowledge of all the problem's data, especially simple and conditional probabilities, often difficult and expensive information to obtain. Furthermore, the *naïve* assumption, which does not consider the correlation between the characteristics of instances, is an approximation, often violated in real-world scenarios.

- *Support Vector Machine* (SVM) is a machine learning algorithm that aims to find a hyperplane (a linear decision boundary) in a high-dimensional space that best separates instances of different classes. Support vectors are the data points that lie closest to the *hyperplane* and contribute to defining the *margin*, i.e. the distance between the hyperplane and the nearest data point of each class. The best hyperplane for an SVM is the one with the largest margin between the two classes when the data is linearly separable. However, SVM can efficiently handle non-linear relationships in the data through the use of a *kernel function*. Common kernels include linear, polynomial, and *Radial Basis Function* (RBF). The kernel trick allows SVM to implicitly map the input data into a higher-dimensional space, making it possible to find a separating hyperplane in that space. The choice of the kernel function and its parameters significantly influences SVM's performance. Its ability to handle high-dimensional data and non-linear relationships makes it suitable for diverse applications. Furthermore, it tends to be robust to overfitting. Its flaw is the sensitivity to outliers, as they may affect the position and orientation of the hyperplane.
- Inspired by the human brain, *Neural Networks* (NN) are computational models composed of interconnected nodes (neurons) arranged in layers. Neurons are basic units that receive inputs, apply a weighted transformation, and produce an output. Layers are made up of neurons and are the building blocks of neural networks. In other words, NNs are nested mathematical functions called activation functions, just like:

$$f_l(z) = g_l \cdot (W_l z + b_l) \quad (2)$$

Learning involves adjusting weights and biases to minimize the difference between predicted and actual outputs. Common activation functions include sigmoid, hyperbolic tangent (tanh), and *Rectified Linear Unit* (ReLU). Neural networks can have various architectures, ranging from simple shallow networks to complex deep networks with multiple hidden layers. Deep neural networks are often referred to as *deep learning* models. They have various hyperparameters, including the number of layers, number of neurons per layer, learning rate, batch size, and others. Three types of layers can be recognized: *input layer*, which receives input features; *hidden layers*, intermediate layers that transform input data into a representation; *output layer*, which produces the final output or prediction. During the training phase, neural networks use *feedforward* and *backpropagation*. The former involves passing input through the network to produce a prediction. From the input layer, all inputs are multiplied by their respective weights and then summed; afterward, the output is passed through the activation function, which determines the output. If that output exceeds a given threshold, it "fires" (or activates) the node, passing data to the next layer in the network: the output of one node becomes the input of the next node. Backpropagation adjusts weights and biases based on the difference between predicted and actual outputs. This iterative process is designed to minimize the error and improve the model's performance. Neural networks represent a powerful class of machine learning models that have demonstrated remarkable success in capturing complex patterns from data. Their ability to automatically learn hierarchical representations makes them suitable for a wide range of applications, contributing significantly to advancements in Artificial Intelligence and deep learning.

- *Ensemble learning* aims to enhance the accuracy and robustness of machine learning models by combining the predictions of multiple base models. This approach often outperforms individual models by leveraging the diversity of different learners. To obtain the prediction for input x, the predictions of each weak model are combined using some sort of weighted voting. One of the most used and effective ensembles learning algorithms is *Bagging* (*Bootstrap Aggregating*), which uses multiple subsets of the training data to train different models and then averages their predictions. One example of a bagging algorithm is *Random Forest*, built by multiple

decision trees. The term "*random*" refers to the fact that at each split in the learning process, a random subset of the features is inspected, ensuring low correlation among decision trees.

The process of designing AI algorithms is a systematic and iterative journey that involves several key steps. From conceptualization to implementation and refinement, the workflow is characterized by a continuous cycle of development. Typical stages involved in crafting effective AI algorithms, in particular for SL, are reported in Figure 6.

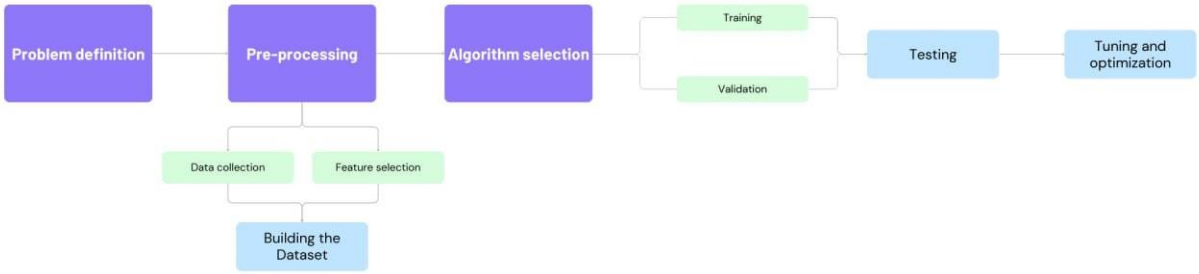


Figure 6: Supervised Learning design process.

4.2. Problem definition

This phase consists of defining the problem or task that the AI algorithm aims to solve.
For the case at hand, this part consists of the phenomenon analysis exposed so far.
As just said, the AI algorithm will be chosen from the classification category. Its purpose is to categorize the analyzed situation into one of three classes:

- *no-VRS Zone*: the helicopter is not encountering VRS.
- *pre-VRS Zone*: the helicopter is approaching the VRS zone, requiring pilot intervention.
- *VRS Zone*: the helicopter is currently experiencing VRS.

This classification approach allows for real-time decision support by identifying potential VRS situations and enabling timely pilot action when necessary.
These three classes are already defined within the simulated environment used for training the algorithm. After data collection from real-world flights, however, the behavior of the selected features will be further studied to understand the underlying physical phenomena that differentiate each state.

4.3. Features selection

In Machine Learning, identifying the most relevant features or variables is crucial for building an effective dataset. These features will be monitored and used to train the AI algorithm.
Building upon the previous study of VRS prediction methods, the most relevant parameters, i.e. the features, are selected. It was chosen that the onset boundary of the VRS will be identified by sequentially mapping out combinations of flight path angle, descent velocity, and forward speed (features of the problem) where the rotor experiences high fluctuations in torque, thrust, lift and pitch and roll. The detection of low-frequency variability or other distinctive characteristics in those should be a reliable indicator of the entry in the VRS since this is most likely associated with the erratic motions of the collapsed rotor flowfield.

4.4. Data collection

Four different flight scenarios were simulated in which the helicopter encountered VRS. The collected data from these simulations forms the foundation for algorithm development. The results are four time-series databases, each containing all flight data from a respective simulated flight. To create a suitable dataset for the next phases, data preprocessing is necessary.

4.5. Building the dataset

During pre-processing, only the previously identified features and parameters are isolated from the raw data. Here's a breakdown of the selected features.
Groundspeed and vertical velocity are taken as forward and descent velocities. Because the flight path angle is not directly collected, this is computed as:

$$\gamma = \arctan\left(\frac{vertical\ velocity}{groundspeed}\right) \quad (3)$$

The other parameters are pitch and roll angles, the requested main rotor torque, and the main rotor thrust. While lift is considered a valuable predictor of VRS as already exposed, it's not typically available to pilots in real-world scenarios. To maintain fidelity with a potential onboard implementation, lift is excluded from the dataset. Data from all four databases are combined into a single dataset. Each data point is labeled based on the simulator's classification within the previously mentioned zones (no-VRS, pre-VRS, VRS). The labels are assigned numerical values (0, 1, 2) for consistency with machine learning

algorithms. This number will be called “VRS trigger” from now on.
Since time itself isn’t a relevant feature for training, the data points are combined without maintaining the original separation from the four databases.

4.6. VRS trigger study

Simulation classification in the three zones is studied to see if a discernible relationship between features and Vortex Ring State is evident at first glance for designers. As expected from previous discussions, VRS occurs at low forward speeds; conversely, increasing forward speed helps exit VRS. Descent rate and glide angle reach about their maximums (in magnitude) during VRS. Visual inspection doesn’t suggest a clear correlation between VRS and pitch/roll oscillations. On the contrary, during the VRS the main rotor required torque and the thrust exhibited a sudden increase during VRS.
While these initial observations provide some insights, recognizing complex patterns for VRS prediction might be challenging through human analysis alone. Therefore, supervised machine learning algorithms are employed in later stages to identify potentially less obvious or intricate patterns within the data that might be more indicative of VRS.

4.7. Algorithm training and selection

The appropriate machine learning model based on the nature of the problem needs to be chosen. For this use case, the *Classification Learner App* on Matlab is used. The application offers a comprehensive selection of algorithms for training, including Decision Trees, Discriminant Analysis, Support Vector Machines, Logistic Regression, Nearest Neighbors, Naive Bayes, Kernel approximation, Ensembles, and Neural Networks. By taking advantage of parallel computing features in Matlab, all these algorithms were trained simultaneously. The automatic validation process in Matlab then ranked them based on decreasing accuracy. The final selection of the most appropriate algorithm wouldn’t be based solely on this initial ranking; after a dedicated testing phase, additional factors will be considered: the test accuracy, so the performance of the chosen model on unseen data; the overall cost, that is the combined cost of training, validation, and testing the model.
The complete dataset is divided into three parts:

- *Training and Validation* (90%): this larger portion will be used to train and validate the chosen machine learning model.
- *Testing* (10%): this reserved portion will be used to assess the model’s real-world performance on unseen data.

To prevent overfitting and ensure the model generalizes well to new data, a *5-fold cross-validation* technique is employed during training and validation. Thus, the dataset is randomly divided into five equal subsets, or folds. In this iterative process, the model is trained on four folds (80% of the data), and the trained model is validated on the remaining fold (20% of the data). This process is repeated 5 times, each time using a different fold as the validation set. After each iteration, the model’s performance is evaluated using an appropriate metric (e.g. accuracy). Then, the performance metrics from all five folds are averaged to obtain a more reliable estimate of the model’s overall performance. This cross-validation process helps assess the model’s ability to perform well on unseen data and reduces the risk of overfitting to the specific training data used.
Table 1 shows the results for 15 of the algorithms analyzed after each phase.

Table 1: Results from classification learning.

Model type	Accuracy (Validation)	Total cost (Validation)	Accuracy (Test)	Total cost (Test)
Fine kNN	99.96%	14	99.97%	1
Wide NN	99.95%	17	99.94%	2
Boosted Tree	99.94%	18	99.94%	2
Tri-layered NN	99.70%	94	99.91%	3
Medium NN	99.95%	16	99.88%	4
Bagged Tree	99.95%	17	99.88%	4
Narrow NN	99.89%	34	99.88%	4
Fine Tree	99.91%	27	99.86%	5
Cosine kNN	99.90%	31	99.86%	5
Gaussian SVM	99.80%	62	99.65%	12
SVM Kernel	98.89%	346	98.76%	43
Kernel Naïve Bayes	93.90%	1898	94.27%	198
Gaussian Naïve Bayes	89.86%	3155	90.14%	341
Linear Discriminant	85.67%	4459	86.46%	468
Efficient Logistic Regression	82.34%	5494	82.38%	609

The *accuracy* represents the percentage of observations correctly classified by the model. Higher accuracy values are desirable.
Total cost refers to the combined cost associated with misclassifications. These occur when the model

makes mistakes: *False Positives*, cases occurring when the classifier predicts a positive outcome when it should have been negative, so the model predicts VRS when it's not occurring, potentially leading to unnecessary pilot intervention; *False Negative* for the vice versa case, thus the model fails to predict VRS when it is happening, potentially causing a dangerous situation. *True Positive* and *True Negative* are correct classifications that typically do not incur additional costs. While prioritizing high accuracy, a lower total cost is also preferred. Given the comparable accuracy values among the majority of algorithms in Table 1, total cost becomes a significant factor in selecting the final model. Since the VRS assistant will be implemented onboard, *prediction speed* is crucial too. It represents the estimated speed for processing new data and making predictions, measured in observations per second (*obs/sec*). Higher obs/sec values indicate faster algorithms. Kernel Naive Bayes demonstrates the fastest prediction speed, but its accuracy and cost might not be optimal. Conversely, despite the hish accuracy, kNN resulted in the slowest algorithms. Considering a trade-off between accuracy, cost, and prediction speed, Wide NN emerges as a strong candidate, offering prediction speeds an order of magnitude faster than Fine kNN, which shows the best accuracy and cost performance. Based on these considerations, Wide NN is chosen as the final algorithm for VRS prediction. It is a Neural Network with a single fully connected layer composed by 100 neurons. A summary of its characteristics is provided in Table 2.

Table 2: Final algorithm characteristics.

Accuracy (Validation)	99.95%
Total cost (Validation)	17
Prediction speed	250000 obs/sec
Training time	426.91 sec
Model size	15 kB
Accuracy (Test)	99.94%
Total Cost (Test)	2

4.8. Tuning and optimization

This phase consists of fine-tuning the model parameters to enhance its performance further. This process may involve adjusting hyperparameters and optimizing the algorithm. The chosen Wide NN model already exhibits excellent performance. However, even with strong results, a more in-depth analysis is valuable.
Figure 7 shows the confusion matrix for the model. This visualization tool helps identify areas for potential improvement. The matrix has rows representing the actual classes and columns representing the predicted classes. Each cell shows the number of instances that fall into a specific combination. Ideally, a good model has high values along the diagonal, indicating accurate classifications. In this case, the confusion matrix suggests the model performs well, with only a few misclassifications. In this case, only 3 pre-VRS instances and 5 VRS intances were misclassified as pre-VRS. The latter was misclassified 7 times as no-VRS and 2 times as VRS.

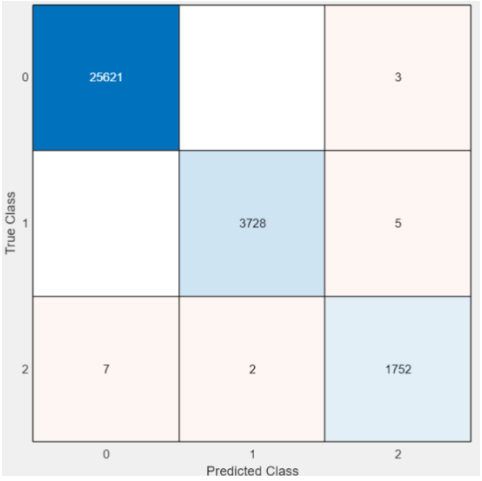


Figure 7: Wide NN confusion matrix for the use case in exam.

While the confusion matrix provides valuable insights, additional metrics can offer a more comprehensive view of the model's performance. Another optimization approach involves filtering features based on their influence on the model's performance. *Minimum Redundancy Maximum Relevance* (MRMR) algorithm is used to rank features based on their relevance to the target variable (VRS state) while minimizing redundancy amongst the features themselves. As expected from the initial feature analysis, the MRMR algorithm ranks pitch and roll angles as having a lower influence on VRS prediction compared to other features (Figure 8).

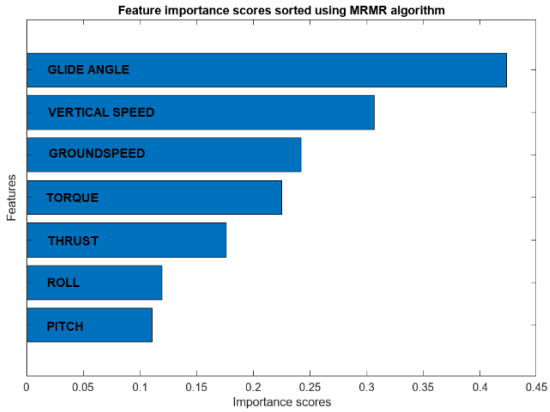


Figure 8: Features importance for the final algorithm.

This application demonstrates that machine learning can be a powerful tool for developing a VRS prediction assistant for helicopters. While the current model already exhibits good results, there's always potential for further optimization, particularly in feature selection. The Minimum Redundancy Maximum Relevance

technique identified features with lower predictive power for VRS, aligning with initial observations made during feature analysis. Overall, this approach using machine learning shows promise for real-world implementation onboard or in a simulated environment to aid pilots in avoiding VRS events.

5. ENGINE MALFUNCTIONS

Despite significant advancements in helicopter technology in recent years, major system and component failures (or malfunctions) remain a concern. Studies on helicopter safety consistently identify pilot error and technical failure as the most critical risk factors. Among these technical failures, engine malfunction is a particularly serious concern for pilots, as it can have catastrophic consequences.

5.1. Helicopter safety

Between 1947 and 1960s accidents were frequent (Ref.26), with most caused by human error or unknown factors. Engine failures were less common than airworthiness issues (failures in non-engine components) and maintenance problems; this might be due to well-developed engines adapted from airplanes and the new concept of helicopters with complex, dynamic parts requiring specialized maintenance.

The introduction of turbine engines in the 1960s offered more power with less weight compared to reciprocating engines. Continuous engine improvements made both engine types more reliable, leading to fewer accidents. The most recent improvement has been the use of electronic controls and digital and analogical assistants on board, enhancing engine performance and safety. An important example is the *Health and Usage Monitoring System* (HUMS). It monitors various parameters to identify potential component failures before they occur, ranging from basic event monitoring to advanced systems detecting issues through vibration analysis.

Thus, Bell's civil turbine helicopters, taken as an example in Ref.26, show a significant improvement in safety over time, with a declining annual accident rate.

Accidents due to airworthiness issues (both engine and non-engine) are infrequent and remain steady over time. Failures originating from components other than the engine account for about 4%. Confirmed engine failures and suspected/claimed power loss incidents have been reduced, accounting for 11.2% of accident causes. The majority, approximately 85%, of accident causes are attributed to factors unrelated to airworthiness. This statistic underscores the significance of addressing a broader spectrum of safety considerations beyond purely mechanical or structural aspects.

According to Ref.27 analysis of over 1,000 U.S. civilian helicopter accidents from 1990 to 1996 revealed that engine power loss was the most frequent initiating event (over 25%) for all but the most expensive helicopter category. In the latter category, airframe and system failures were the most common cause. This difference can be attributed to the use of multiple engines in some helicopters, particularly for critical operations. Large helicopters may have three engines, while modern twin-engine helicopters (like the one used as an example in this work) typically fall under Category A. These Category A twins are designed to maintain safe flight even with one engine inoperative. The higher proportion of twin-engine helicopters in the very high-cost category explains the lower rate of accidents due to engine failure.

All things considered, although helicopter safety has been improving over the years and the accident frequency due to engine malfunctions appears to decrease, these continue to be hazardous risks to pay attention to. For this reason, the industry will continue to keep airworthiness issues corrected, developing new technologies such as machine learning-based assistants to aid the pilots.

Ref.28 reports that helicopter engines experience higher failure rates compared to fixed wing aircraft with similar engines. This is primarily due to the harsher operating environment of helicopters. Unlike airplanes, helicopters frequently operate at or near maximum power, and their engines undergo numerous power fluctuations per flight hour. These factors contribute to an accelerated engine life cycle in helicopters compared to fixed-wing applications. An engine's most stable state is at constant power. The frequent adjustments to power settings during typical helicopter flights, such as those needed during hovering maneuvers, increase the risk of failure. Furthermore, these power changes cause fluctuations in temperature and lubrication, accelerating wear on engine parts and raising the likelihood of failure.

Engine failures can have various causes, including:

- Design or manufacturing errors.
- Improper engine operations (pilot or maintenance crew errors).
- Ground or air traffic control errors.

Therefore, human error is often cited as the leading contributor to engine failures. For example, one of the most common causes of engine failure is fuel starvation. This can arise from various factors, with pilot error in fuel planning being the most frequent culprit, rather than a fundamental flaw in the aircraft or engine itself. Fuel contamination is another significant contributor to fuel starvation accidents. Other causes of engine failure, not directly connected to the human factor are: *Foreign Object Damage* (FOD), debris on runways or birds flying into the engine that can cause damage; mechanical problems, so broken or worn parts, overheating, etc.

During normal operation, engine torque is transmitted to the main and tail rotors through the powertrain.

This system consists of interconnected gearboxes and drive shafts. Freewheel units link the main gearbox to each engine. These freewheels engage when engine speed exceeds rotor speed, and disengage during engine failure, preventing drag from the inoperative engine(s) on the rotor system.
According to the RFM of the AW189, different indications of power failure are present on-board. The message of 1 or two engine(s) out appears in case the *gas generator* velocity (NG or N1) is less than 50% or its rate of change outside the prescribed limits.
Following engine drive shaft failure, the *turbine speed* NF or N2, may overspeed and reach the *overspeed trip point* of 119%. Also, in this case, the pilot would receive an alert message. In general, a low RPM horn and a message indicate power failure.
Engine and rotor speeds, along with torque readings, will decrease. Without engine power, rotor RPM will decline. A sudden reduction in torque accompanied by either low collective pitch or high rotor speed suggests a problem other than engine failure. However, a sudden torque drop with fixed collective and decreasing rotor speed is a strong indicator of engine failure. Additionally, a jerk may be felt on the yaw channel as the tail rotor compensates for the reduced main rotor torque. Detecting gradual power loss may be more challenging. The pilot's response to engine failure is critical. First, they must identify the failed engine and feather it (adjust the propeller to minimize drag). Next, they need to maintain the best glide speed, the speed that maximizes unpowered flight distance.
Finally, a suitable landing site must be identified, and a safe landing initiated. With one engine inoperative, the helicopter can often maintain altitude and airspeed while searching for a landing site. Several factors determine this capability, including aircraft weight, altitude, airspeed, and the helicopter's single-engine performance specifications. Pilot reaction time and control technique also play a role. Identifying the malfunctioning engine correctly is critical because misidentifying the engine could have disastrous consequences. In the event of a dual-engine failure, flight characteristics, and required crew control responses are similar to those during a normal power-on descent, except for the case it could require the entry in autorotation.

6. MACHINE LEARNING FOR FAILURES DETECTION

This section explores the use of machine learning for detecting engine anomalies and proposes an alternative communication channel between sensors and pilots. Here, unsupervised learning algorithms are employed to identify deviations from normal engine behavior without the need for labeled data on specific failures. The design process is similar to the SL learning one previously exposed.

6.1. Unsupervised Learning

Unlike the supervised one, in the unsupervised learning approach, there are no predefined output labels for the algorithm to learn from. Instead, the algorithm explores the inherent structure within the data to uncover meaningful insights. The objective is to discover patterns, relationships, or groupings within the data. As a kind of learning, it resembles the methods humans use to figure out that certain objects or events correlate with themself, such as by observing the degree of similarity between them. Different types of unsupervised learning exist, and here are briefly reported two:

- *Clustering* is the most common unsupervised learning technique. The main goal of clustering is to discover inherent structures or patterns in the data based on similarities between data points; so it is useful for exploratory data analysis. Clustering algorithms use a similarity or distance metric (such as Euclidean distance) to measure how closely related or similar data points are to one another. The effectiveness of clustering can be sensitive to the choice of distance metric and algorithm parameters. Algorithms fall into two broad groups: *hard clustering*, where each data point belongs to only one cluster; *soft clustering*, where each data point can belong to more than one cluster. Different techniques of the two types exist. The principal ones are: *Hierarchical Clustering*, which forms a hierarchy of clusters, which can be visualized as a tree-like structure, by analyzing similarities between pairs of points; *k-Means* divides the data into a predefined number of clusters (k) based on the mean or centroid of data points in each cluster, i.e. defined k number of centroids, each element in the dataset is assigned to a centroid to which it is closest; DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) identifies dense regions in the data space, separating them from less dense areas.
- *Anomaly detection*, also known as *outlier detection*, aims to identify abnormal or unusual instances in a dataset that deviate from the expected patterns, or rather the primary goal of anomaly detection is to identify instances that differ significantly from the majority of data points. It is valuable in various domains where detecting rare or unexpected events is crucial, such as fraud detection, network security, and equipment failure prediction. *Normal-only methods* train an algorithm on normal data only and identify data outside those norms as anomalous. Some of these methods are reported. *Thresholding* identifies an anomaly when data exceeds a threshold.

In *One-Class Support Vector Machines* the training is done considering only one class composed of data that can be considered normal, which allows the detection of anomalies without having any labeled anomalies available for training. It does not work well on high-dimensional data.

An *Isolation Forest* is constructed by randomly selecting a feature and a random split value to create a decision tree recursively until each data point is isolated in a leaf node. Anomalies are expected to have shorter paths in the decision tree, as they are isolated more quickly. Therefore, the path length in the tree can be used as a measure of anomaly score.

Autoencoders are neural networks trained on normal data that attempt to reconstruct the original input.

Unsupervised learning is chosen because the dataset lacks labels for specific failures. This allows the machine learning model to identify anomalies in sensor data based on patterns observed in normal flight conditions. For the case in exam, normal-only methods will be analyzed, so the situation is about a flight without malfunctions or with a small percentage of malfunctioning data. Based on the engine malfunction and failure sections of the RFM, four features were initially chosen for analysis:

- Gas generator velocity NG (or N1).
- Rotor Speed NR.
- Turbine speed NF (or N2).
- Yaw.

In the last sections, what malfunctions each one could indicate were reported.

Among the UL normal-only methods explored, three were compared: One-class Support Vector Machine, Isolation forests, and Autoencoders.

6.2. One class SVM results

This model effectively identifies abnormalities that significantly deviate from the normal data distribution. The tuning focus was on a single hyperparameter: the contamination fraction. This parameter represents the expected proportion of anomalies in the training data. The latter was constructed from flight data collected in a certified flight simulation environment, ensuring the quality and realism of the flight scenarios. To create the training data, the original dataset was skewed to 95% normal flight data and 5% OEI data, reflecting a contamination fraction of 0.05. The test dataset comprised unseen data from the original dataset, containing a higher anomaly ratio (83% anomalies, 17% normal).

The model outputs two key values for anomaly detection: *anomaly indicators t* and *anomaly scores s*. These scores help assess the effectiveness of the trained model. Anomalies are identified by comparing their scores to a pre-defined score threshold. In this case, the model automatically sets the threshold to the maximum score in the training data. Any new data point exceeding this threshold is flagged as an anomaly. Figure 9 shows the distribution of anomaly scores for the training data.

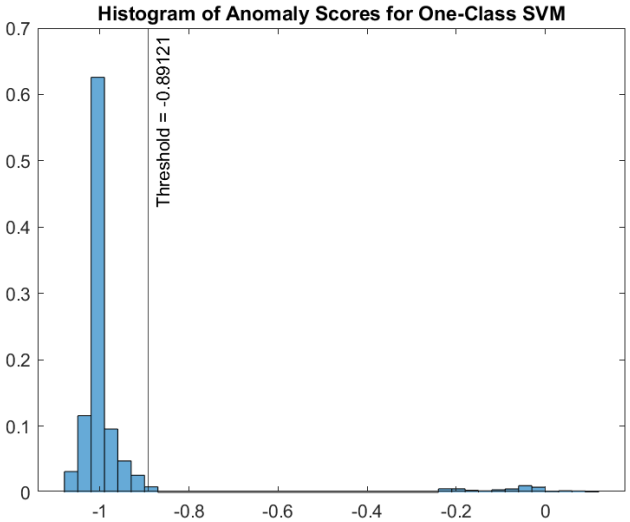


Figure 9: Trained SVM anomaly scores.

As expected, the identified anomalies (sum of all t values) were about 5% of the training data, reflecting the contamination fraction. However, the test phase accuracy was only around 84%. The model flagged 98% of the test data as anomalies, significantly higher than the actual anomaly ratio (83%). A potential reason for this discrepancy is feature relevance: the features used for training might not adequately capture the variations associated with anomalies across the scenario. Including additional features potentially enhances anomaly detection. Therefore, One-class SVMs are sensitive to outliers and can struggle with datasets where anomalies are not well-separated from normal data points. Additionally, the model learns a boundary around the majority class (normal flight data). If the distribution of anomalies differs significantly between the training and test data, the One-Class SVM may not generalize well, resulting in decreased performance. Exploring alternative methodologies for such scenarios could bebeneficial.

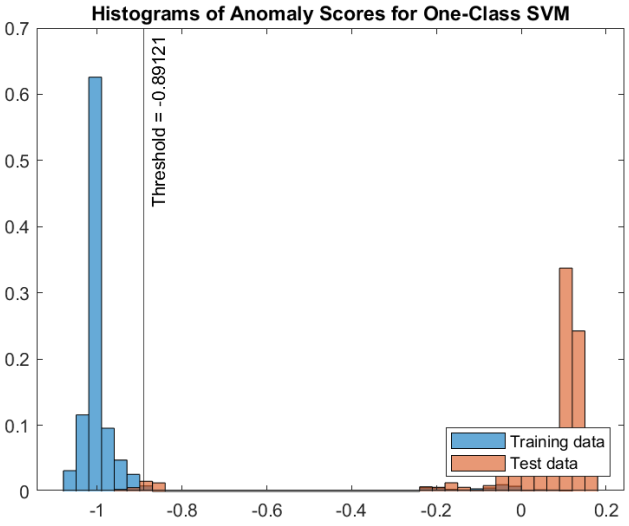


Figure 10: SVM histogram of anomaly scores in the test phase.

6.3. Isolation Forest results

Similar to the One-Class SVM, Isolation Forest assigns anomaly scores to data points. These scores are based on the average path length a data point takes through the ensemble of isolation trees used by the algorithm. The distribution of these anomaly scores for the training data is visualized in Figure 11.

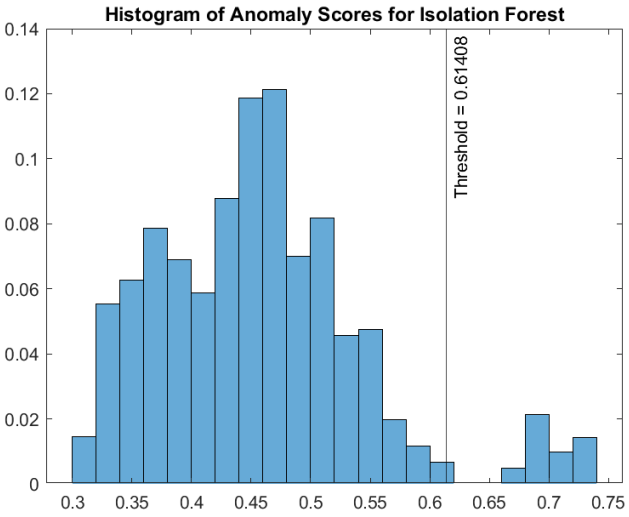


Figure 11: Trained Isolation Forest anomaly scores.

Following the approach used for One-Class SVM, Isolation Forest's performance was evaluated on the same test datasets. Encouragingly, Isolation Forest achieved an accuracy of 98.84%. This superior performance compared to the One-Class SVM suggests several advantages of Isolation Forest for this application:

- Isolation Forest is less sensitive to the specific distribution of anomalies within the training data. Unlike One-Class SVM, it doesn't require a specific data shape distribution. This makes it more robust for scenarios where anomalies are subtle or where the distribution of anomalies differs between training and test data.
- Isolation Forest thrives in high-dimensional data spaces (with many features) and can handle large datasets more efficiently compared to One-Class SVM. This characteristic could be particularly beneficial for future applications where additional features might be incorporated.

Overall, Isolation Forest demonstrates the promising potential for anomaly detection in engine data due to its robustness and scalability.

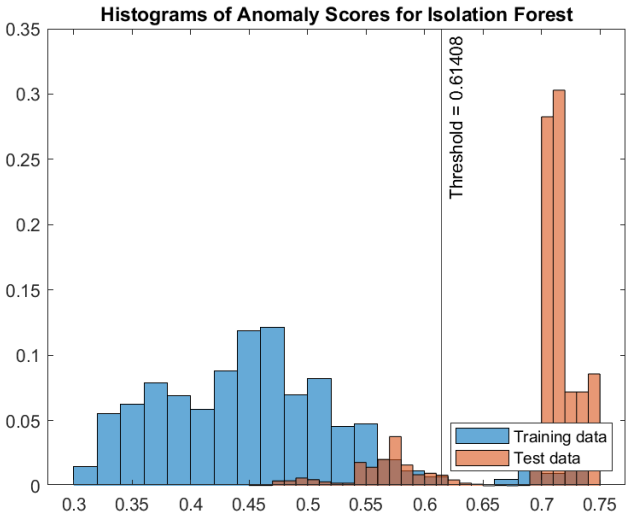


Figure 12: Isolation Forest histogram of anomaly scores in the test phase.

6.4. Autoencoders results

Autoencoders are a powerful neural network architecture adept at learning efficient representations of data. Their methodology involves compressing input data into a lower-dimensional latent space through an encoder and then endeavoring to reconstruct the original input from this condensed representation via a decoder. During this reconstruction process, autoencoders essentially learn to identify and discard irrelevant information in the data.

Anomalies deviate from the typical patterns observed in normal data. By training on normal engine data, the autoencoder learns to reconstruct these "normal" data points accurately. However, when presented with an anomaly, the reconstruction attempt by the decoder will result in a significantly higher error compared to normal data. This higher reconstruction error serves as a potential indicator of anomalies within the input data.

The specific architecture of an autoencoder, including the complexity of the encoder and decoder, can be tailored to the specific problem and data characteristics.

Hyperparameters were left at their default settings as per Matlab's configuration. The training

performances are illustrated in the accompanying Figure 13.

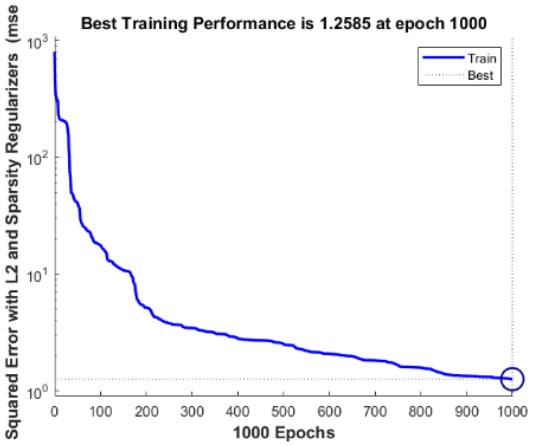


Figure 13: Autoencoders performance.

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	1000	1000
Elapsed Time	-	00:00:09	-
Performance	803	1.26	0
Gradient	845	0.655	1e-06
Validation Checks	0	0	6

Figure 14: Autoencoders hyperparameters.

In this application, the Mean Squared Error (MSE) was used to evaluate the autoencoder's performance. A lower MSE indicates better data reconstruction. The trained autoencoder achieved an MSE of 0.5458. The testing phase yielded an MSE value of 377.89. This increase in MSE signifies a decline in performance for anomaly detection on unseen data. This indicates that the model cannot generalize well beyond the training data. The possible causes could be: the autoencoder has too many parameters (e.g., too many neurons or layers), allowing it to memorize the training data, thus reducing the model complexity by using fewer neurons or layers or using early stopping during training to prevent overfitting may be solutions to the problem; the chosen hyperparameters (e.g., learning rate, batch size) may not be optimal; not using a validation set to tune the model can lead to overfitting.

As demonstrated in the previous sections, Isolation Forest achieved superior performance in this application compared to autoencoders and SVMs. This suggests that Isolation Forest might be a more suitable choice for on-board or simulator implementation due to its robustness and potentially lower computational cost. However, expanding the training dataset and incorporating additional features could potentially improve the performance, potentially yielding different outcomes and decisions.

7. AUTOROTATION

As already extensively said, reaction to system failures is an example where a virtual assistant could be useful, aiding the pilot promptly as the vehicle needs to be recovered. One particularly demanding maneuver, rotorcraft autorotation, highlights this potential.
Autorotation is a critical maneuver that requires quick reaction and critically timed control inputs in multiple axes, to allow helicopters to maintain lift and control even in the complete absence of engine power. Pilots have minimal margin for error, making autorotation proficiency a constant training challenge, especially in low visibility conditions. Due to its inherent danger, rotorcraft pilots have limited resources for training and maintaining proficiency in autorotation, often viewing it as a maneuver to be approached with uncertainty. For this reason, a guidance methodology and cueing system can be transformative; for example, a system that could either autonomously execute the autorotation maneuver or provide real-time, tailored assistance to the pilot would be beneficial.
This chapter will delve deeper into the complexities of autorotation and the challenges it presents; then a possible machine learning implementation designed to assist pilots during this maneuver will be introduced.

7.1. Pilot training

Autorotation is a lifesaving skill used to land safely on a helicopter descending without engine power. Instead, air flowing from the bottom up through the rotor disk drives the blades, allowing for a controlled descent. It involves a complex trade-off between altitude, airspeed, and rotor energy to achieve a controlled descent with minimal vertical and horizontal velocity upon touchdown: during autorotation, the helicopter rotor has a minimum descent rate, the same as a parachute of equal size.
However, mastering autorotation is notoriously difficult due to several factors, including highly coupled and non-linear aircraft dynamics as well as pilot unfamiliarity with entry conditions and terminal constraints. The latter is because unfortunately, practicing autorotation is costly and potentially dangerous, as delays or improper execution can escalate an incident into a fatal accident. Traditional training methods often separate autorotation into unrealistic scenarios, like hovering autorotation near the ground or practicing simulated landings from high altitudes.
Hence, employing a helicopter flight simulator is a logical approach. Unlike subjective instructor evaluations in real aircraft, simulators provide detailed, objective feedback on control inputs, allowing for immediate performance improvement. The demonstrations can show how to do the maneuver, presenting common errors and ways to avoid them. In addition, simulators can dynamically adjust difficulty based on pilot proficiency, ensuring continued challenge and

motivation throughout training and precise monitoring of pilot trainee progress.
While helicopter simulators offer numerous advantages for pilot training, replicating the precise sensations and cues required for real-world autorotation maneuvers remains a challenge.
The autorotation maneuver of a helicopter following engine failure is typically divided into three distinct phases:

- *Entry*. Upon engine failure, the pilot rapidly lowers the collective to preserve rotor RPM and applies the right pedal to counter yaw, thus preventing a decay in rotor speed, which is no longer regulated by the engine governor, and arresting the angular motion of the helicopter. To avoid blade stall, the pilot quickly reduces blade pitch while maximizing blade inertia. A pilot reaction time of two seconds following engine failure is usually employed doing this sequence.
- *Steady-state descent*. During this phase, the pilot pulls the cyclic control aft to slow airspeed and increase airflow through the rotor disc, initiating a controlled glide, selecting a landing area, and planning the approach to the wind. It is essential to manage the potential energy by trading it for airspeed to achieve the optimal descent rate. The rotor RPM is regulated by gently lifting the collective if it exceeds optimal levels during the glide. These actions must be performed flawlessly within seconds of engine failure. Descent usually occurs in a manner that maximizes gliding range to offer the pilot more landing options. The helicopter's aerodynamic efficiency plays a significant role in achieving a good glide range. Essentially, well-designed rotor systems minimize drag, allowing the helicopter to travel further for the same amount of descent.
- *Cyclic flare and collective pull*. As the helicopter approaches the chosen landing area, the pilot initiates a flare by raising the helicopter's nose. The objective is to level the aircraft, preserve or increase rotor RPM, and significantly reduce both vertical and horizontal speed near zero for a soft touchdown. So, the pilot focus is split between coordinating collective pitch increase to cushion the landing with cyclic control movements that level the fuselage and prevent tail rotor strike. Tilting the rotor disk allows more air to flow through it, converting rotor energy into lift for a gentle landing while simultaneously reducing airspeed and descent rate. Improper timing during the flare can lead to a dangerous rotor stall (initiated too early) or a hard landing (initiated too late).

A typical autorotative descent is depicted in Figure 15.

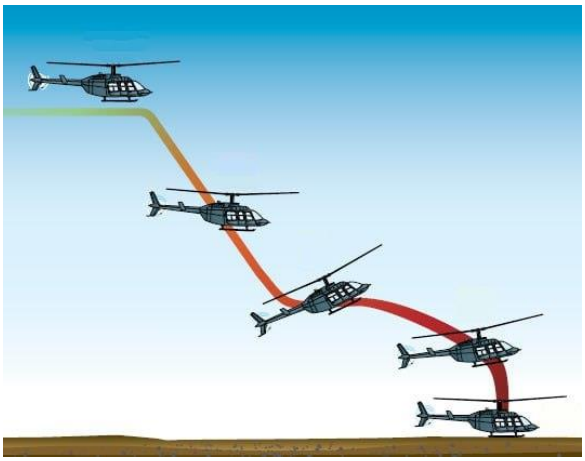


Figure 15: Straight-in autorotation from Ref.34.

A straight-in autorotation is here considered, that is the maneuver made with no turns. However, turns can be incorporated during an autorotation to facilitate landing into the wind or avoiding obstacles. Any turns should be performed early in the autorotation, as the remaining steps should be identical to a straight-in autorotation to ensure a smooth and controlled landing.
Studying both analytically and experimentally the transient dynamics and control of a helicopter after engine failure, these three phases were then further broken down into different segments and tasks, reported in below and resuming what had just
been described:

- Autorotation entry: detect the emergency; lower collective to maintain sufficient rotor rpm; add right pedal preventing yaw; pull cyclic aft prevent pitch down and keeping rotor rpm up.
- Steady state descent: establish normal glide, so set airspeed with cyclic and control rpm with collective; select landing area, so identify potential landing positions, reduce the area given wind effect, reduce the area given terrain problems and imagine a flight course to reach there; normal glide, so adjust helicopter heading with cyclic and glide path angle with cyclic.
- Deceleration and touchdown: flare, so rotate aircraft nose upward with cyclic to slow or stop the ROD and lower collective to increase rotor rpm; soft touchdown, so level the aircraft with cyclic, lift collective, employ right pedal to maintain heading and prevent rollover, lower collective on touchdown.

While engine failure is commonly recognized as a primary reason for employing autorotation in helicopter flight, and this paper will mainly refer to this case, it

also plays a critical role in the event of tail-rotor failure. If the tail rotor fails, keeping the engine running creates a strong torque that spins the helicopter uncontrollably. Autorotation, by stopping the engine and eliminating torque, enables a more stable descent for landing.

7.2. H-V envelope

The ability of a helicopter to execute a safe autorotative landing following a power failure depends on its structural and aerodynamic design, particularly at specific altitudes and airspeed combinations. During certification, an H-V (height-velocity) diagram is established. The avoid areas defined by this curve are combinations of altitude and airspeed where a safe landing becomes extremely challenging, even for skilled pilots. Operating within the avoid area means the pilot has limited ability to manage the descent; there's insufficient airspeed or altitude to effectively trade for rotor energy needed for a safe touchdown.
As operations in these unsafe regions are not uncommon, ideally one would like to eliminate or minimize these unsafe regions. High rotor inertia, low disk loading, and a high maximum thrust coefficient can all help shrink the avoid area. Also, headwinds significantly reduce the difficulty of autorotation landings. Figure 16 shows a typical height-velocity diagram.

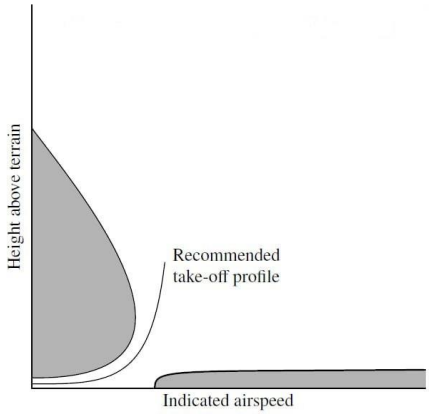


Figure 16: A typical helicopter H-V curve, Ref.35.

Typically, a height-velocity diagram is derived through flight tests. Pilots simulate engine failures under various conditions, progressively pushing into more critical scenarios until they feel unable to safely land from conditions more critical than the last. These tests establish limits across altitude and airspeed combinations to construct the H-V diagram.
If engine failure occurs in a hover well clear of the ground, the pilot reduces collective to maintain rotor speed and uses cyclic to gain forward speed before initiating a flare and landing. In forward flight, the presence of forward speed can be used to improve the glide angle, gain height initially, or maintain rotor RPM. Nonetheless, careful control of forward speed during the flare is paramount to prevent loss of tail rotor authority, particularly at low RPM. Touching down with too much horizontal velocity can cause the helicopter to tip over upon ground contact, while landing with insufficient main rotor speed can cause the rotor blades to flex sharply downward, increasing the risk of a blade strike against the tail boom, and causing significant damage. Certain combinations of height and forward speed may lead to uncontrolled landings in total engine failure scenarios and thus should be avoided during normal operations. However, autorotation can still be performed safely within the avoid region of the H-V curve with well-timed and appropriate control inputs. In the restricted high-speed region, the tail clearance becomes a concern. Pilots must flare and decelerate quickly to control rotor RPM before touchdown. For high-altitude entries, above the altitude at the knee of the curve, the pilot reduces collective and pitches the nose down to gain airspeed and build rotor RPM before initiating a decelerating flare. This also helps establish an optimal airspeed for minimum drag, allowing more time for selecting a landing site. However, if hovering very near the ground, collective pitch cannot be lowered further, and rotor RPM decays rapidly. The pilot relies on precise timing to extract the remaining rotor energy and arrest the descent rate.
The maximum safe height from which a landing can be made is therefore determined mainly by the energy-absorbing properties of the undercarriage.

7.3. Autorotation procedures and specific limitations

This section outlines the general steps a pilot should follow during an autorotation descent according to the AW189 Rotorcraft Flight Manual. The airspeed limitations outlined in the manual, detailed in Table 3, serve as a basis for the algorithm design phase and will be presented later.

Table 3: Autorotation limitations from the RFM.

Maximum airspeed with TQ above 100%	90 KIAS
Maximum landing gear operating airspeed	150 KIAS
Minimum airspeed for flight under IFR	50 KIAS
Maximum airspeed for IFR approach	150 KIAS
Minimum airspeed in autorotation	60 KIAS

The general procedure for autorotative descent is as follows:

1. Smoothly reduce collective to enter autorotation.
2. Maximum NR 110%.
3. Adjust attitude to obtain approximately 80 KIAS.

4. To recover to powered flight, slowly increase collective pitch until freewheels are joined and at least 15% torque is indicated. Finally, increase power, gently, in not less than 3 seconds, to arrest the rate of descent.

The RFM indicates that a restart attempt may be possible if the engine failure occurs at a significant height *Above Ground Level* (AGL), assuming a quick diagnosis of the cause. Considering an average autorotation descent rate of 3,000 feet per minute, a minimum AGL of 5,000 to 6,000 feet would be necessary to allow sufficient time for completing *Auxiliary Power Unit* (APU) and engine restart procedures.

The autorotative landing procedure on land reported on the RFM closely resembles the one presented previously. However, during the initial autorotation entry, the pilot should adjust the cyclic control to achieve an airspeed between 70 KIAS and 100 KIAS, which is the optimal glide speed. Additionally, the collective pitch needs to be adjusted to maintain a rotor speed of around 110%.

The cyclic flare maneuver should begin at approximately 200 ft AGL, with a 15° pitch attitude. At around 35 ft, the pitch attitude should be reduced to 10°, and collective should be applied as needed to achieve a touchdown with a vertical descent rate of 300 fpm or less. The procedure for landing on water is very similar, with adjustments made based on the specific water conditions. An interesting feature noted in the consulted RFM is the *Automatic Flight Control System's* (AFCS) autorotation protection. This system prevents unintended autorotation entry during steep descents by limiting the minimum collective pitch commanded by the AFCS. This function activates when engine torque (TQ) falls below 15%.

7.4. Background and previous works

Building on the established understanding of helicopter dynamics covered in previous chapters, the helicopter can be viewed as a controllable system, defined by its state at any given time. This system receives pilot inputs (collective, cyclic, and pedal) and transforms them into outputs. When it becomes necessary for one or more output variables of the system to track desired values over time, a controller may adjust the system's inputs to achieve the desired outcomes. Optimal control theory extends this concept further. Here, the controller determines the input signals to minimize a specific performance criterion while adhering to the system's physical limitations. As with many nonlinear control problems, finding a closed-form solution is typically not feasible; therefore, numerical algorithms are employed.

The pioneering application of this theory to study the autorotative descent and landing of helicopters was presented by Johnson (Ref.29), in 1977. He utilized a 2D nonlinear helicopter model with a momentum theory-based inflow model, including ground effect correction, and an empirical model for VRS-induced flow. The optimal cost function aimed for minimal vertical and horizontal velocities at touchdown while maintaining acceptable flight conditions during descent. The method showed that the optimal control solution for descent from hover, after power loss, was a purely vertical flight. Interestingly, the optimal control solution for descent from hover after power loss suggested a purely vertical flight path. Johnson's findings were later expanded upon by Lee (Ref.30), theorizing that optimal pilot inputs during autorotation could significantly reduce the avoid areas within the height-velocity (H-V) restriction curve.

Based on this statement, Aponso et al. (Ref.31 and 32), proposed a novel approach to helicopter autorotation using a fast and robust optimal control algorithm. This algorithm is capable of directly commanding control motions in an autonomous rotorcraft or guiding a pilot through an intuitive display. The algorithm leverages a direct optimization method to generate optimal control trajectories for the helicopter's longitudinal axis, focusing on collective and aircraft pitch as control inputs. Notably, the computed trajectories demonstrated good agreement with those achieved by an expert pilot during flight tests. Additionally, the research group developed a guidance display specifically for autorotation training. This display presents a ground plane, the horizon, and a unique helicopter bug symbol that centralizes critical rotorcraft state information, such as the attitude and the ground clearance. Time-based guidance profiles for collective and pitch control are also shown on the display. Pilots use this information to adjust their controls and minimize the discrepancy between their current control settings and the recommended inputs displayed.

Ref.33 explored the use of augmented cueing in a realistic simulated cockpit environment. This system aimed to improve student training by correcting pilot perceptions and control inputs. By providing additional, relevant information through augmented cues, the learning curve can be accelerated. The cues can simplify complex tasks, allowing trainees to more rapidly grasp the correct visual cues and control responses needed for a successful flight. However, the challenge lies in striking a balance. While the cues should be helpful, they shouldn't create an over-reliance. Ideally, pilots should develop the skills to perform tasks without depending solely on the augmented information. Additionally, the cues themselves should highlight the critical information present in real-world scenarios, ensuring a smooth transition from simulation to actual flight. The cueing system consists of a set of eight pairs of F-poles augmented visual aids for glideslope visualization. The specific cueing system described in the study utilized eight pairs of F-poles with augmented visuals to aid in glideslope visualization. The horizontal and vertical distances between these poles represented the

correct approach path for landing. Furthermore, a flight path predictor, depicted as a small aircraft symbol, provided real-time information on the predicted flight path, aircraft roll, and direction.

Various methods for optimizing helicopter trajectories during autorotation descent are discussed in the literature. Studies by Yomchinda (Ref.36) utilize a modified Dubin's curve for pathfinding. This onboard algorithm allows for variable turn rates and constant acceleration/deceleration along three segments: two turns and a straight section. It is assumed the helicopter is in a quasi-steady autorotation, maintaining constant bank angle, rotor speed, and horizontal acceleration throughout the descent. This approach applied to a generic point mass helicopter model, generates 3D autorotation paths for successful landings. The modified strategy integrates descent rate modeling and allows for acceleration in addition to standard turn-straight-turn parameterization. By utilizing bank angle and horizontal airspeed, the method defines a modified Dubin's path in the horizontal plane, meeting the desired final position, heading, and horizontal velocity constraints. Following horizontal path determination, acceleration is calculated as a non-linear function of these parameters. Vertical descent is then addressed using a map of the aircraft's autorotation performance, where rotor speed is used to regulate the descent rate. The method offers several advantages: paths are built using basic geometries (circles and lines), the method generates the shortest path, and at least two paths are always produced, with one being the optimal solution. Tierney and Langelaan (Ref.37) focused on the flare phase. They defined a "Safe Landing Set" - a region in the helicopter's state space (airspeed, descent rate, rotor speed, etc.) that guarantees a safe landing. The optimal control method identifies this set, providing a target zone for the mid-descent phase and indicating the ideal flare initiation point for a desired landing location. These two works were reunited in Ref.38 and other follow-on papers. Ref.39 extended the Safe Landing Set approach to encompass wind conditions, including strong headwinds and light tailwinds. This addresses a crucial limitation of previous research, which often neglected wind despite its significant impact on autorotation safety. Although numerous methods exist for optimizing autorotation trajectories, there has been a growing interest in autonomous autorotation. Currently, manned helicopters in service do not engage in automatic autorotation. This said the concept of having an avionics system performing an automatic autorotation is not new. In 1970 such a system was already described in Ref.40. Its role was to detect engine failure and to automatically lower the collective stick promptly before a rotor stall was encountered. The concept however did not go as far as to describe the automatic landing of a helicopter after power failure. It is also interesting to note that an automatic autorotation system could potentially represent an alternative to multiple-engine helicopter configurations, as a safeguard against catastrophic engine failure. By eliminating the need for multi-engine helicopters, the availability of such a system could obviously translate into substantial cost savings.

In 2004 in Japan, Hazawa et al. (Ref.41) developed two autorotation models, one using a neural network to compute the flight path parameters, which used a PI controller to land a small, unmanned helicopter in simulation.

The first autonomous controller to successfully pilot a Remotely Controlled (RC) helicopter during an autorotation descent and landing was performed at Stanford University in 2008 (Ref.42) using an apprenticeship learning-based approach. The procedure included the collection of flight data from an expert pilot, used to build a dynamics model of the helicopter in autorotation, or better to find a good target trajectory. The
landing itself was achieved by forcing the helicopter to hover at 0.5 m. *Differential Dynamic Programming* (DDP), an extension of the *Linear Quadratic Regulator* (LQR) formalism for non-linear systems, was used to derive a feedback controller. The helicopter performs 25 autorotations, each of which results in a successful landing.

Further work in automatic autorotation through reinforcement learning can be found in Ref.43, where the algorithm was evaluated by simulations based on a point-mass model of a modified OH-58A helicopter. The Q-learning algorithm was adopted; the final state-action space was covered using *Radial Basis Functions* (RBF), whose parameters were updated using backpropagation. The success rate was around 80%. In this approach, the accuracy of the simulation model used is very important. Furthermore, the controller needs to be repeatedly trained under all possible conditions. In a problem with high state dimensionality, this can lead to a large computational load. Inspired by biological systems, the time-to-contact (τ) theory offers a powerful approach to modeling pilot guidance during landing maneuvers. This theory suggests that pilots use the perceived time to contact the ground to regulate their control inputs. Here, the pilot acts as the observer and the desired flight path serves as the motion guide. The core idea is that pilots aim to minimize the gap between their perceived visual information (optical flow) and the desired flight trajectory. They achieve this by adjusting their control inputs based on the rate of change of this gap.

In Ref.44 and Ref.45 control algorithms derived from this theory have been employed to provide cues to pilots for executing maneuvers manually. The cueing system consists of a head-up cockpit display featuring visual markers indicating desired and actual collective pitch and longitudinal cyclic positions throughout the entire maneuver, from engine failure to main gear touchdown. The initial step involved analyzing flight data from simulated autorotation maneuvers.

This analysis revealed a common τ-guidance strategy. Subsequently, this strategy formed the basis for developing the τ-based control law for automated flare maneuvers during autorotation landings. Here the outcomes of a pilot support system integrating fuzzy logic techniques for helicopter autorotation are mentioned.
In Ref.46 two applications are reported: one is a fuzzy system designed to automatically control the helicopter after engine failure, reducing pilot workload and potentially improving outcomes, and the other is a fuzzy system that acts as a decision support tool, suggesting optimal flight paths for autorotation landing and providing realistic visual cues to aid the pilot. Fuzzy systems offer a way to incorporate human expertise into computer systems by representing imprecise or subjective human knowledge in a way computers can understand. Fuzzy rules express human knowledge or experience in a natural language format, using terms like “low”, “medium”, or “high” instead of precise numerical values. A membership function translates the vagueness of linguistic terms into numerical values that computers can process. Traditionally, such quantification has relied on binary definitions. By combining fuzzy rules and membership functions, a fuzzy system can perform “fuzzy inference”. This process involves evaluating the current situation using the membership functions and applying the appropriate fuzzy rules to generate control commands or suggestions for the pilot.
An autorotation controller for unmanned helicopters is presented in Ref.47. This controller operates on a high-fidelity mathematical model of a real light utility helicopter. The system utilizes fuzzy logic to blend collective pitch commands generated by different control laws specific to each phase of autorotation (entry, descent, flare). These controllers provide both forward velocity references and maximum allowable pitch limits to a lower-level control loop, along with the collective commands.
Dalamagkidis et al. (Ref.35) present a real-time controller for the autorotation landing of unmanned helicopters. This controller combines a *Nonlinear Model Predictive Control* (NMPC) strategy with a recurrent neural network to enable safe landings from low hover altitudes. Unlike previous work focused on vertical descent models, this controller operates on a full autorotation model. In piloting a helicopter there are strict real-time requirements that may be difficult to satisfy with traditional methods. Neural networks can be implemented effectively allowing high processing speeds for the optimization. This controller aims to offer two key benefits. Firstly, it allows for safe landings even in scenarios where a malfunction might have caused an uncontrolled crash. Secondly, it minimizes the helicopter’s descent rate before touchdown, reducing the risk of damage to the aircraft and injuries to occupants.

Despite advancements, current autonomous autorotation methods, above exposed, face limitations that prevent their direct integration into commercial helicopters and UAVs. Significant work is needed to address these shortcomings and achieve the levels of safety and reliability required for certification and onboard use.

8. MACHINE LEARNING FOR AUTOROTATIVE DESCENT

From what has just been said, automatization of flight control or some pilot tasks can increase time for decision making. After an engine failure, pilots must quickly identify potential landing sites. They then visualize a flight path to each location and assess its feasibility based on factors like distance, wind direction, and wind speed. If a path seems achievable, it becomes a candidate for the emergency landing. This process repeats for multiple locations until time constraints force a decision.
The concept driving the development of a virtual assistant for autorotation training is to equip the agent with the capability to execute autorotation maneuvers. The resultant policy can then assist pilots in searching for suitable landing sites or evaluating potential flight paths to a predetermined location. This would reduce the pilot’s workload and allow them to focus on critical decision-making during emergencies. Reinforcement Learning seems to be the best choice to deal with this problem.

8.1. Reinforcement Learning

Reinforcement learning is inspired by behavioral psychology, where an agent learns to make decisions by interacting with an environment. The agent perceives the state of that environment as a vector of features, and it receives feedback in the form of rewards or penalties based on the actions it takes. The goal is to learn optimal strategies or an optimal sequence of actions, called policies, through trial and error. The action is optimal if it maximizes the expected average reward. It is already easy to notice that RL is very far from supervised learning since it does not need supervisors or a label dataset; all the information and all the correct answers are collected by interacting with the external world. Furthermore, RL solves problems where decision-making is sequential, and the goal is long-term.
In summary, the protagonists in Reinforcement Learning are the *environment* and the *agent*. The environment is everything that exists outside of the agent, which it interacts with, sending *actions*, and receiving *rewards* and *observations*. The environment includes the system dynamics, so the agent is just a bit of software that generates the actions and updates the policy through learning. Since RL uses trial-and-error experience, the agent does not require complete knowledge or control of the environment; it only

needs to be able to interact with the environment and collect information. So, sometimes the dynamics of the environment are not explicitly modeled. This type of reinforcement learning is called *model-free* (the other one model-based). A *policy* defines the learning agent's way of behaving at a given time, so it is the strategy of mapping from states to actions that the agent follows to make decisions. The reward is a numerical feedback signal, that represents the goodness of an agent being in a particular state and taking a particular action. Whereas the reward signal indicates what is good in an immediate sense, a *value function* specifies what is good in the long run, or rather it is an estimate of the expected cumulative future rewards for a given state or state-action pair.
A critical aspect of reinforcement learning is the tradeoff between *exploration* and *exploitation*: exploration involves trying new actions or visiting different states to discover their effects and gather information about the environment; exploitation involves selecting actions that the agent believes will lead to the highest immediate reward based on its current knowledge. Too much exploration may lead to suboptimal decisions initially, as the agent invests time in discovering the environment. Overreliance on exploitation may result in the agent missing out on discovering better, yet unexplored, actions or states. Balancing these two aspects is crucial for achieving optimal performance in a given environment.

The *Markov Decision Process* (MDP) is the formal framework that defines the RL problem with states, actions, rewards, and *transition probabilities*. The latter defines the probability of transitioning from one state to another, given a particular action. This process is governed by the *Markov property*: the future state depends only on the current state and action, not on the sequence of states and actions, that preceded it. Solving an MDP means identifying the optimal policy, so apply reinforcement learning.
Reinforcement learning algorithms can be categorized differently depending on their properties:

- Model-based and model-free methods, already explained.
- In *on-policy methods* the learning agent uses the same policy for both exploration and exploitation, while *off-policy methods* use a different policy from the one to make decisions under evaluation.
- *Online methods* operate in real-time on environmental data and can interact with it or with a simulated version of it, while in *offline methods* the agent learns from a fixed dataset, collected independently of the learning process.

These categories can be combined in various ways, obtaining different solution methods. In general, all methods can be traced back to an *actor-critic* scheme, which consists of an actor, responsible for selecting actions based on the current policy, and a critic, which evaluates the actions chosen by the actor by estimating their value.
Here some of the RL methods are reported:

- *Dynamic programming*: leveraging the principle of optimality, it systematically solves subproblems and combines their solutions to find an optimal policy for the overall problem.
- *Q-learning*: learns the values associated with state-action pairs through iterative updates based on observed rewards and estimated future values.
- *Policy Gradient Methods*: utilize gradient ascent to optimize policy parameters for maximizing expected rewards.
- *Temporal Difference Learning*: updates value estimates incrementally by combining current rewards with estimated future values.
- *Monte Carlo Methods*: averages returns obtained from sampled episodes to evaluate- and improve policies.

8.2. Imitation Learning

Imitation Learning is a machine learning paradigm where an agent learns a task by observing and replicating the behavior of an expert demonstrator. Unlike Reinforcement learning, instead of relying solely on rewards from the environment, the agent learns a policy directly from the expert's actions. Two principal approaches exist. The first one is *Behavior Cloning*, in which the agent directly mimics the demonstrator's actions in a supervised learning fashion. The training data are composed of pairs of states and expert actions. The second is *Inverse Reinforcement Learning*, which attempts to infer the underlying reward function that guided the demonstrator's actions. The former is the simpler method, but susceptible to compounding errors, especially if the expert's policy is suboptimal in some situations. The second one offers a better final solution, but it can be really complex to implement. Imitation learning finds applications in various domains, including robotics, autonomous vehicles, and human-computer interaction. It offers a practical approach for training agents to perform tasks where expert guidance is readily available, resulting in a better solution to reinforcement learning for some applications.
Machine learning is a really vast discipline, and in this chapter, many things have not been covered. When it comes to programming the algorithms, some topics will be revisited and explored in depth.

8.3. RL model definition

The Simulink model of a civilian helicopter serves as the model environment.

Observations
The agent receives information about the helicopter's state through observations, which are essentially real-time data feeds. For this application, the chosen observations include:

- Rate of Descent (ROD), the vertical velocity in NED axes, in [fpm].
- True Airspeed (TAS), in [kts].
- Altitude, in [m].
- Pitch angle θ, in [rad].
- Rotor speed (NR), as a percentage.
- X-body acceleration, in [m/s].
- Z-body acceleration, in [m/s].

To ensure data stability, these observations are kept within a reasonable range based on the helicopter's specifications derived from the RFM or the dataset collected during some autorotative descent in a certified simulation environment (Table 4).

Table 4: Observations limits.

Parameter	Lower limit	Upper limit
ROD	-3000 *fpm*	1000 *fpm*
TAS	0 *kts*	150 *kts*
Altitude	-200 *m*	600 *m*
θ	0	2*π*
NR	80%	113%
x-acceleration	-2 *m/s*	2 *m/s*
z-acceleration	-10 *m/s*	12 *m/s*

These limits also allow for normalization between 0 and 1, simplifying calculations for the agent, using the formula:

$$par_{norm} = \frac{par - par_{min}}{par_{max} - par_{min}}$$

Actions
The agent influences the environment through actions, which for this model are:

- Collective deflection, limited between 0 and 1.
- Lateral cyclic, limited between -1 and 1.
- Longitudinal cyclic, limited between -1 and 1.
- Pedals, limited between -1 and 1.

Agent's actions were normalized too for simplifying computations.

Rewards
The agent's actions are guided by a reward function. This function assigns positive or negative rewards based on how well the agent performs the autorotation maneuver. The goal, as defined by the autorotation procedures and limitations in the previous section, is to achieve a safe landing. The reward function is:

$$\begin{aligned} r = \; & c_{V,madeit} \cdot (-1.5V + 1) - 0.1 \cdot \left(1 - c_{V,madeit}\right) \\ & + \; c_{V_z,madeit} \cdot (-0.3V + 0.5) - 0.3 \\ & \cdot \left(1 - c_{V_z,madeit}\right) + 0.2 \cdot c_{rotor} \\ & + c_{a_x madeit} \cdot (-23(a_x - 0.4)^2 + 0.5) \\ & + c_{a_x madeit,2} \cdot (-4(a_x - 0.4)^2 + 0.5) \\ & - 0.3 \cdot \left(1 - c_{a_x madeit} + c_{a_x madeit,2}\right) \\ & + c_{a_z madeit} \cdot (-3(a_x - 0.44)^2 + 0.7) \\ & - 0.1 \cdot \left(1 - c_{a_z madeit}\right) + 0.6 \cdot c_{\theta,madeit} \\ & - 0.3 \cdot \left(1 - c_{\theta,madeit}\right) + 200 \cdot c_{madeit} \\ & - 150 \cdot c_{failed} \end{aligned}$$

The terms which refer to $c_{V,madeit}$ rewards the helicopter for keeping its horizontal speed (TAS) between 60 and 100 knots when the altitude is above 200 feet. This reward is set as a linear function with greater values for lower speed. If the helicopter is above that height but its velocity is outside the range, it will receive a negative reward. $c_{V,madeit}$ is the logical value that results in true if altitude is above 200 ft, false on the contrary. In the same way. the terms that refer to $c_{V_z,madeit}$ encourage the agent to slow down the descent rate towards a minimum value, promoting a controlled landing. This value was assumed to be about 1500 fpm. Thus, $c_{V_z,madeit}$ is true if V_z is between 1500 fpm and 60 fpm. The agent will receive a positive reward if this condition is respected. For the other cases, descent velocity bigger than 1500 fpm or a positive value of this (climb), the model will receive a penalization.
Then, the agent is rewarded for keeping the rotor speed within a safe range, between 95% and 110% of the maximum (c_{rotor} is true in that case).
$c_{a_x madeit}$ and $c_{a_x madeit,2}$ are conditions that give the model a parabolic reward greater if a_x is near zero, a penalization vice versa. The same is valid for $c_{a_z madeit}$ for a_z. To design the acceleration functions a simulation dataset with 4 autorotative descent examples was analyzed.
$c_{\theta,madeit}$ is the condition that gives a reward if the pitch angle is maintained below 15°, this is to avoid VRS and to encourage the helicopter to flare.
Finally, c_{madeit} is true if the model reaches its goal: a significant positive reward is provided if the agent successfully lands the helicopter (successful safe autorotation), meeting the following criteria:

$$\begin{cases} h \leq 35\ ft \\ V \leq 15\ kts \\ V_z \geq -300\ fpm \\ 10° \leq \theta \leq 15° \end{cases}$$

These criteria coincide with the flare-end conditions found in the RFM, already exposed.

The agent receives negative rewards for exceeding parameter limits, hitting the ground, for a pitch angle exceeding the range $-10° < \theta < 20°$, or experiencing a positive descent rate (indicating climbing). c_{failed} is true if these conditions occur or parameters exceed their limits. When the agent meets either the success (*madeit*) or failure (*failed*) conditions, the simulation terminates (the *check if done* condition becomes true). This prevents unnecessary computation and allows the agent to move on to the next training episode.
Designing the reward function involves an iterative process. The coefficients multiplying each term of the function (i.e. the numbers) were adjusted multiple times based on the agent's performance during training. This process is akin to trial and error, where the goal is to find the optimal balance between different factors. The reward function guides the agent's learning. By adjusting the coefficients, the influence of which actions the agent prioritizes is defined.

Reset function
To expose the agent to a wide range of scenarios, the simulation environment is reset at the beginning of each training episode, essentially a single training step. This is achieved through a reset function by randomizing the initial conditions. Thus, the agent is forced to learn and adapt its decision-making for various flight situations, improving its overall performance in real-world autorotation scenarios. The inputs and their initial values for each reset are delineated below:

- Latitude. Randomized between $\frac{-\pi}{2}$ and $\frac{\pi}{2}$, based on its geographical definition.
- Longitude. Randomized between 0 and 2π, covering the entire globe.
- Altitude. Randomized between 400 *m*, a reasonable starting point for autorotation, and 600 *m*.
- Wind velocity. Components along the wind axes are initialized as $w_x = 5\ m/s$, $w_y = w_z = 0\ m/s$.
- Attitude. Angles initialized as $\phi, \theta, \psi = 0$.
- x-body velocity. Randomized between 70 *kts* and 100 *kts*, as recommended for autorotation in the RFM.
- y-body velocity. Set to zero (no initial sideways movement).
- z-body velocity. Randomized between between 0 kts and 29.62 *kts* (3000 *fpm*), representing a reasonable range for the initial descent.
- Rotor speed, NR. Fixed at 110% of maximum (34.12 *rad/s*), as suggested by the RFM.

Agent
The chosen Reinforcement Learning method is *Deep Deterministic Policy Gradient* (DDPG). This algorithm excels in problems with continuous action spaces. This means the agent can take a wide range of actions, not just discrete choices like "left" or "right". DDPG relies on two deep neural networks working together: an actor and a critic, already briefly introduced. The actor is a network that analyzes the current state of the environment (observations) and proposes the most suitable action. It essentially acts as the agent's brain, aiming to make decisions that will lead to a successful outcome.
The critic is a second network that acts as a reviewer, assessing the value (potential reward) of the action chosen by the actor in a given state. By comparing the actual rewards received with its predictions, the critic network continuously learns to better estimate future rewards. Based on the improved critic's feedback, the actor-network adjusts its strategy to choose actions that lead to higher predicted future rewards. In essence, both the actor and critic are neural networks striving to learn optimal behavior. The actor refines action selection based on the critic's appraisal, while the critic hones its value estimation from received rewards, facilitating constructive critiques of the actor's actions. This synergy enables the agent to leverage the strengths of both policy and value function algorithms.
The actor architecture is shown in Figure 17. The feature input layer serves as the entry point for the network. It takes the 5 observations as input. The two fully connected layers (or dense layers) in the middle process the incoming observations from the previous layer. Each layer has 150 neurons (artificial processing units) that apply mathematical functions to the data. *REctified Linear Unit* (ReLU) layers introduce non-linearity into the network. Essentially, they activate neurons only if their input is positive, forcing the network to learn more complex patterns from the data. The last fully connected layer connects the previous layers to the output layer. It has 4 neurons, corresponding to the number of possible actions the agent can take during a flight simulation. The final layer applies a hyperbolic tangent (tanh) activation function. This function squashes the output values between -1 and 1, making them suitable for control signals sent to the simulated helicopter.

Figure 17: Actor architecture.

Critic architecture is shown in Figure 18. Two paths, one for observations and another for actions, could be recognized. Its layers are from the same type of the already described actor-network.

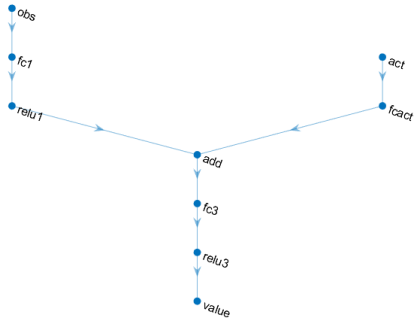


Figure 18: Critic architecture.

The scheme of the RL model is illustrated in figure 19.

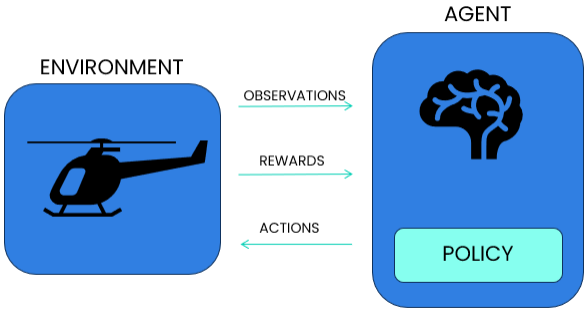


Figure 19: Reinforcement learning model scheme.

Model hyperparameters

Hyperparameters are crucial settings that control the learning process in Reinforcement Learning. These parameters define how the agent interacts with the environment, learns from its experiences, and updates its internal policies. Choosing the right hyperparameters significantly impacts the agent's performance and training efficiency. Throughout the design process, the model's hyperparameters underwent iterative adjustments based on simulation results, culminating in the final set reported in Table 5.

Table5: Model hyperparameters.

Hyperparameter	*Value*
Learning rate	$5 \cdot 10^{-5}$
Gradient threshold	10
Weight decay	10^{-4}
Sample time [s]	0.017
Simulation time [s]	600
Standard deviation	0.3
Standard deviation decay rate	10^{-6}
Experience buffer size	10^{6}
Mini batch size	128
Max episodes	5000
Max steps per episode	72000

Here's a breakdown of what these hyperparameters are and how they have been chosen. Learning algorithm parameters applied to networks:

- *Learning rate* controls how much the agent updates its internal policy based on new experiences. A low learning rate leads to slow learning, while a high learning rate can cause the learning process to diverge or get stuck in suboptimal solutions. Typical learning rate values are in the order of 10^{-5}/ 10^{-4}.
- *Gradient threshold* limits the magnitude of updates to the agent's policy during training. Large gradients can cause erratic updates, so clipping them helps stabilize the learning process.
- *Weight decay* penalizes large weights in the agent's network, promoting simpler policies and reducing the risk of overfitting, especially for complex networks with many parameters (*L2 regularization*). Increasing weight decay can help improve generalization.

These hyperparameters are kept equal for both the actor and the critic.

Experience replay parameters:

- *Sample time* T_s defines the frequency at which the agent samples experiences from the environment to populate the experience buffer. Commonly, this parameter is equal to the step used in the Simulink model

integration. In this application, sample time was set as half the Simulink model step value.

- *Simulation time* T_f expresses the time of a single simulation (*episode*). This was set based on the simulation time of an autorotation for the model with the four controls in a neutral position.
- It is possible to introduce noise to the agent's actions during exploration to help it escape local minima and improve the diversity of experiences collected. This parameter controls the amount of noise added (*standard deviation*) and how it diminishes over time (*decay rate*).
- The *experience buffer* stores past interactions with the environment. This parameter determines the maximum capacity of the buffer. A larger buffer allows for storing more diverse experiences but also increases memory usage.
- *Mini batch size*. During training, the agent updates its policy based on a subset of experiences randomly sampled from the buffer. This parameter defines the size of this mini batch. Larger mini batches reduce variance in gradient updates but require more computation.

Training termination parameters:

- *Max episodes* set the maximum number of episodes the agent will train for, regardless of other stopping criteria.
- *Max steps per episode* defines the maximum number of steps allowed in each episode. It acts as a safety net in case the environment doesn't have a built-in termination condition. This is computed by approximating the ratio of simulation time to sample time.

In addition to the above, it is possible to define additional criteria for stopping training, such as achieving a desired performance level (certain reward value) or encountering diminishing returns.

Reinforcement learning algorithms often require significant training time, which can be a major bottleneck in development and deployment. This challenge has been addressed by regulating sample time, simulation time, and learning rate. Utilizing multiple processors or GPUs (*parallelization*) can significantly speed up training by distributing the computational workload across multiple cores or machines. This approach can be particularly beneficial for complex environments or large neural networks.

8.4. Results

Various reward functions, including different coefficient values and mathematical formulations, were tried for several training sessions to learn the helicopter model to perform autorotation. At the time of writing, despite 50 simulations with varied initial conditions, the agent achieved a 0% success rate. Analysis revealed two key issues. In most simulations, the agent attempted to slow its descent but ended up exceeding zero vertical velocity (i.e., unintentionally climbing). This is particularly puzzling since the simulations were conducted with zero engine torque. In the remaining simulations, the agent failed to decelerate sufficiently and surpassed the predefined descent rate limits. While the other aspects of the reward function seemed functional, the terms related to vertical velocity (V_z) require further investigation, perhaps playing more with the initial conditions and terms related to the management of vertical velocity by the helicopter (pitch angle and vertical acceleration). Moreover, simplification of the model could be beneficial too. What has just been said exemplifies also a general challenge in RL training: the significant time investment required for effective agent training: a single training session it can last even tens of hours. Although the desired results were not achieved, this work is a good starting point for the study of an assistant for autorotation.

9. CONCLUDING REMARKS

This thesis explored the potential of machine learning to assist helicopter pilots in critical situations. The research focused on three key areas: predicting Vortex Ring State, engine failures and malfunctions detection, and autorotation assistance.

Supervised learning algorithms demonstrated remarkable capabilities in classifying flight regimes based on various flight data features. The chosen algorithm, Wide NN, achieved nearly 100% accuracy in predicting Vortex Ring State, using features like forward and descent velocities, glide descent angle, pitch and roll oscillations, and thrust and torque fluctuations. Further improvements can be achieved through expanding datasets with more diverse data and fine-tuning the algorithm hyperparameters for optimal performance. Design of a pilot interface for this application needs to be done. It will be crucial to ensure that only critical information is presented, avoiding information overload during demanding situations like VRS encounters.

Unsupervised learning proved effective in detecting anomalies in engine data. Isolation Forest emerged as the most effective algorithm among those compared. To further improve performance, the following steps are recommended: enriching training datasets with more engine data, especially during engine malfunctions simulations; incorporating additional engine-related features, as those already presented, for analysis; optimizing hyperparameters for better anomaly detection sensitivity. For practical implementation, a study should explore how this AI assistant can integrate with existing onboard sensors. Visual or aural cues could be implemented to alert

pilots of potential engine anomalies, complementing existing warning systems.
Reinforcement learning presents the most complex challenge due to the inherent difficulty and the constraints associated with autorotation maneuvers. While significant progress has been made, further development is needed to achieve a comprehensive autorotation assistant. Key areas for improvement include:

- Refining the reward function to ensure safe and controlled autorotations. This might involve exploring alternative mathematical formulations that incentivize the desired behaviors during training.
- Techniques like Behavior Cloning or Imitation Learning hold significant potential. These techniques could allow the agent to learn from existing pilot data, effectively mimicking human decision-making during autorotation. First, it is needed to segment expert demonstrations (e.g. human pilot actions in a simulator) into state-action pairs. This data could then be used to train a supervised learning model that mimics the expert's behavior. One approach within Behavior Cloning is Data Aggregation (Dagger).

The ultimate goal is to develop an assistant that can help pilots identify suitable landing zones within a reachable footprint. Additionally, the assistant could compute feasible flight paths to reach a chosen landing site. Building a database of potential landing zones could further enhance this application's effectiveness.

This research lays the groundwork for future advancements in machine-learning-based helicopter aiding systems. Several promising avenues for further exploration exist, as exposed, and conducting extensive testing and validation of these systems in simulated and real-world flight environments to ensure safety and reliability will be essential. Ultimately, these efforts hold the potential to significantly improve the automation, accuracy, and efficiency of helicopter-aiding systems, leading to advancements in both the aviation industry and machine learning technologies.

10. REFERENCES

1. Catherine M Webb, Steven J Gaydos, Arthur Estrada, and Lalla S Milam, "Toward an operational definition of workload: A workload assessment of aviation maneuvers", Fort Belvoir, Virginia, USA, United States Army Aeromedical Research Laboratory, Warfighter Performance and Health Division, 2010.
2. EASA, "Easa artificial intelligence roadmap 2.0", 2023.
3. Michael Guevarra, Srijita Das, Christabel Wayllace, Carrie Demmans Epp, Matthew Taylor, and Alan Tay, "Augmenting flight training with ai to efficiently train pilots", in proceedings of the AAAI Conference on Artificial Intelligence, volume 37, pages 16437–16439, 2023.
4. Kaira Samuel, Matthew LaRosa, Kyle McAlpin, Morgan Schaefer, Brandon Swenson, Devin Wasilefsky, Yan Wu, Dan Zhao, and Jeremy Kepner, "Ai enabled maneuver identification via the maneuver identification challenge", arXiv preprint arXiv:2211.15552, 2022.
5. Harvis project, https://www.harvis-project.eu/.
6. Walter Castles Jr and Robin B Gray, "Empirical relation between induced velocity, thrust, and rate of descent of a helicopter rotor as determined by wind-tunnel tests on four model rotors", technical report, NASA, 1951.
7. KW Mort and PF Yaggy, "Wind-tunnel tests of two vtol propellers in descent", technical report, 1963.
8. R Wayne Empey, "Tail-rotor thrust on a 5.5-foot helicopter model in ground effect", in proc. 30th Annual AHS National Forum, 1974.
9. Hong Xin and Z Gao, "An experimental investigation of model rotors operating in vertical descent", in European Rotorcraft Forum, Cernobbio, Italy, 1993.
10. Li Chenglong, Fang Zhou, Wang Jiafang, and Zhang Xiang, "A vortex-ring-state avoiding descending control strategy for multi-rotor uavs", in 2015 34th Chinese Control Conference (CCC), pages 4465‑4471, IEEE, 2015.
11. J Meijer Drees, LR Lucassen, and WP Hendal, "Airflow through helicopter rotors in vertical flight", NLR V, 1535, 1949.
12. Kyuichiro Washizu, Akira Azuma, JlRO KOO, and Toichi Oka, "Experiments on a model helicopter rotor operating in the vortex ringstate", Journal of Aircraft, 3(3):225‑230, 1966.
13. Julian Wolkovitch, "Analytical prediction of vortex-ring boundaries for helicopters in steep descents", Journal of the American Helicopter Society, 17(3):13‑19, 1972.
14. David A Peters and Shyi-Yaung Chen, "Momentum theory, dynamic inflow, and the vortex-ring

state", Journal of the American Helicopter Society, 7(3):18–24, 1982.

15. Mark D Betzina, "Tiltrotor descent aerodynamics: A small-scale experimental investigation of vortex ring state", in American Helicopter Society 57th Annual Forum, Washington, DC, 2001.
16. John Gordon Leishman, Mahendra J Bhagwat, and Shreyas Ananthan, "Freevortex wake predictions of the vortex ring state for single-rotor and multi-rotor configurations", in AHS International, 58 th Annual Forum Proceedings, volume 1, pages 642–671, 2002.
17. Simon Newman, Richard Brown, John Perry, Steven Lewis, Matthew Orchard, and Ajay Modha, "Predicting the onset of wake breakdown for rotors in descending flight", Journal of the American Helicopter Society, 48(1):28–38, 2003.
18. Albert Brand, Ron Kisor, Robert Blyth, Dave Mason, and Chris Host, "V-22 high rate of descent (hrod) test procedures and long record analysis", in American Helicopter Society 60th Annual Forum, 2004.
19. Ron Kisor, Robert Blyth, Albert Brand, and Tom MacDonald, "V-22 lowspeed/high rate of descent (hrod) test results", in American Helicopter Society 60th Annual Forum, Baltimore, MD, 2004.
20. Oliver Westbrook-Netherton and CA Toomer, "An investigation into predicting vortex ring state in rotary aircraft", proc., RAeS Advanced Aero Concepts, Design, and Operations, pages 1–14, 2015.
21. David J Varnes, "Development of a helicopter vortex ring state warning system through a moving map display computer", PhD thesis, Monterey, California, Naval Postgraduate School, 1999.
22. H Glauert, "The analysis of experimental results in the windmill brake and vortex ring states of an airscrew", r & m no. 1026. British ARC, 1926.
23. Wayne Johnson, "Model for vortex ring state influence on rotorcraft flight dynamics", technical report, 2005.
24. C Chen, JVR Prasad, P Basset, and S Kolb, "Prediction of vrs boundaries of helicopters in descent flight", in annual forum proceedings American Helicopter Society, volume 62, page 1161, INC, 2006.
25. Guglieri Giorgio, slides from the course "Meccanica del volo dell'elicottero", Politecnico di Torino.
26. Roy G Fox, "The history of helicopter safety. In International helicopter safety symposium", volume 690, 2005.
27. Laura Iseler, Joe DeMaio, and Michael Rutkowski, "An analysis of us civil rotorcraft accidents by cost and injury (1990-1996). Technical report, 2002.
28. Franklin R Taylor and Rich Adams, Investigation of hazards of helicopter operations and root causes of helicopter accidents, Federal Aviation Administration, Program Engineering and Maintenance Service, 1986.
29. Wayne Johnson, "Helicopter optimal descent and landing after power loss", Technical report, 1977.
30. Allan Y Lee, Arthur E Bryson Jr, and William S Hindson, "Optimal landing of a helicopter in autorotation", Journal of Guidance, Control, and Dynamics, 11(1):7–12, 1988.
31. B Aponso, E Bachelder, and Dongchan Lee, "Automated autorotation for unmanned rotorcraft recovery", in AHS international specialists' meeting on unmanned rotorcraft, 2005.
32. Edward N Bachelder and Bimal L Aponso, "An autorotation flight display/director for helicopter training", in Helmet-and Head-Mounted Displays VIII: Technologies and Applications, volume 5079, pages 370–381. SPIE, 2003.
33. Steven P Rogers and Charles N Asbury, "A flight training simulator for instructing the helicopter autorotation maneuver" (enhanced version). Technical report, 2000.
34. FEDERAL AVIATION ADMINISTRATION, "Helicopter Flying Handbook".
35. Konstantinos Dalamagkidis, "Autonomous vertical autorotation for unmanned helicopters", University of South Florida, 2009.
36. Thanan Yomchinda, Joseph Horn, and Jack Langelaan, "Flight path planning for descent-phase helicopter autorotation", in AIAA

Guidance, Navigation, and Control Conference, page 6601, 2011.

37. Shane Tierney, “Autorotation path planning using backwards reachable set and optimal control”, 2010.

38. Thanan Yomchinda, J Horn, and J Langelaan, “Autonomous control and path planning for autorotation of unmanned helicopters”, in Proceedings of the American Helicopter Society 68th Annual Forum, Fort Worth, Texas. Citeseer, 2011.

39. Nicholas Alexander Grande, “Safe autonomous flare and landing during autorotation through wind shear”, 2013.

40. HH McIntyre, “A simplified study of high speed autorotation entry characteristics”, in 26th Forum of the American Helicopter Society, 1970.

41. Kensaku Hazawa, Jinok Shin, Daigo Fujiwara, Kazuhiro Igarashi, Dilshan Fernando, and Kenzo Nonami, “Autonomous autorotation landing of small unmanned helicopter”, Nippon Kikai Gakkai Ronbunshu C Hen (Transactions of, 16(10):2862–2869, 2004.

42. Pieter Abbeel, Adam Coates, Timothy Hunter, and Andrew Y Ng, “Autonomous autorotation of an rc helicopter”, in Experimental Robotics: The Eleventh International Symposium, pages 385–394. Springer, 2009.

43. D Jin Lee, Hyochoong Bang, and Kwangyul Baek, “Autorotation of an unmanned helicopter by a reinforcement learning algorithm”, in AIAA Guidance, Navigation and Control Conference and Exhibit, page 7279, 2008.

44. MM Alam, Michael Jump, and Neil Cameron, “Can time-to-contact be used to model a helicopter autorotation”, in Asian/Australian Rotorcraft Forum, pages 1–7, 2019.

45. Brian Eberle, Jonathan Rogers, Michael Jump, and Neil Cameron. Time-to-contact-based control laws for flare trajectory generation and landing point tracking in autorotation. In Annual Forum Proceedings-AHS International, volume 2018, 2018.

46. Makoto Uemura, Munenori Ishikawa, Naoki Sudo, Ikuo Sudo, and Yoshiharu Kubo, “Fuzzy intelligent pilot support system for an autorotative flight of helicopter”, in EUROPEAN ROTORCRAFT FORUM, volume 19, pages D5–D5, ASSOCIAZIONE ITALIANA DI AERONAUTICA ED ASTRONAUTICA, 1993.

47. Kaan S, ansal, “Control of a helicopter during autorotation”, Master’s thesis, Middle East Technical University, 2018

APPENDIX A: VRS use case.

In this appendix, other analysis of the SL trained algorithm are reported**.**

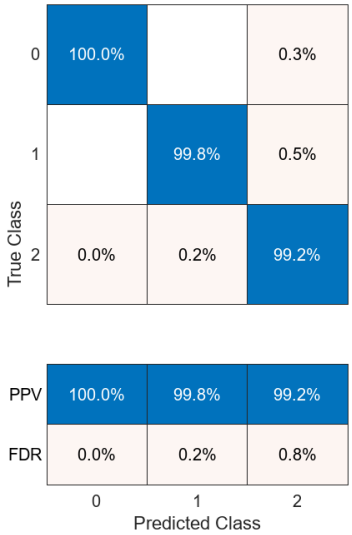


Figure A1: True Positive Rates and False Positive Rates for the final algorithm.

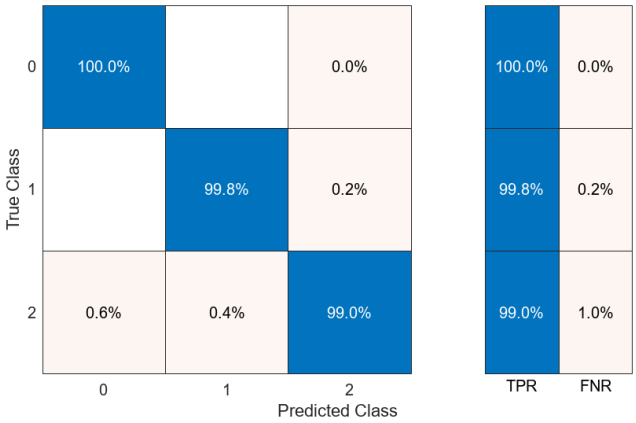


Figure A2: Positive Predictive Values and False Discovery Rate for the final algorithm.

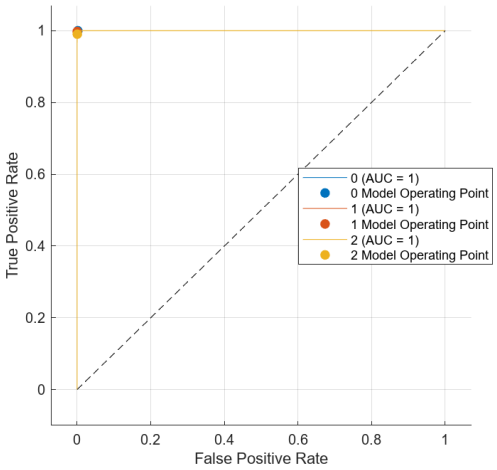


Figure A3: ROC curve of the final algorithm.

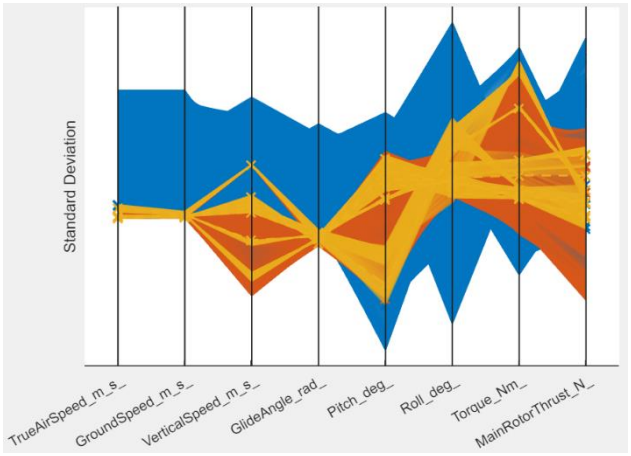


Figure A4: Parallel Coordinates plot for the final algorithm.

APPENDIX B: Autorotation use case.

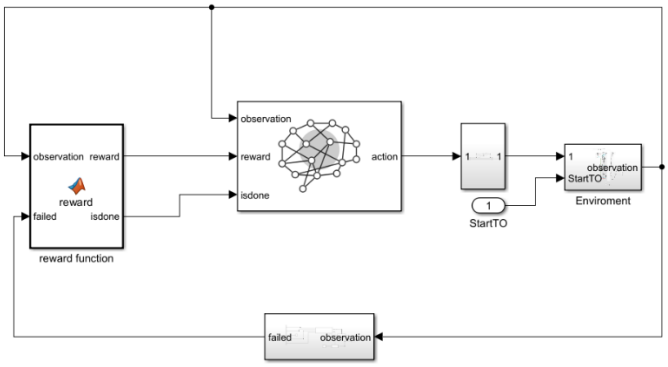


Figure B1: RL Simulink model.

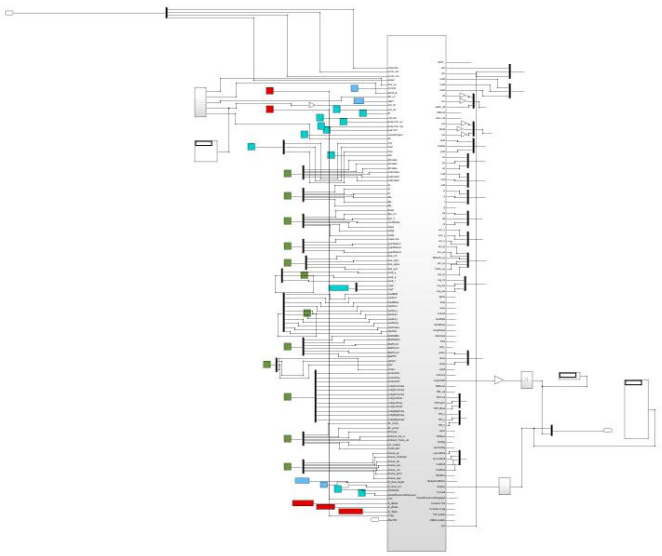


Figure B2: Model environment.

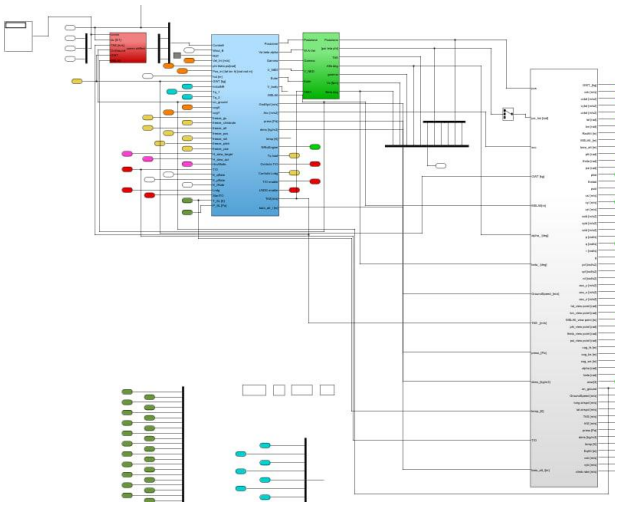


Figure B3: Model environment more in depth.

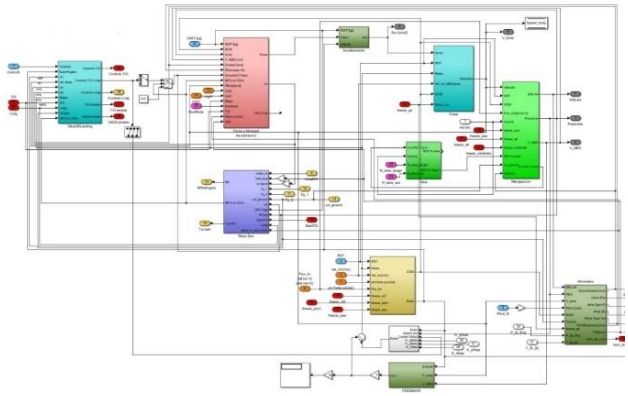


Figure B4: Model environment more in depth part2.

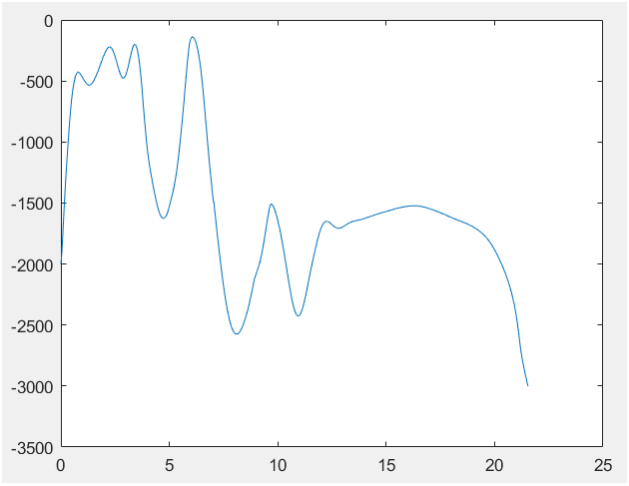


Figure B5: Example of after-training simulation (vertical velocity exceeding the lower limit).

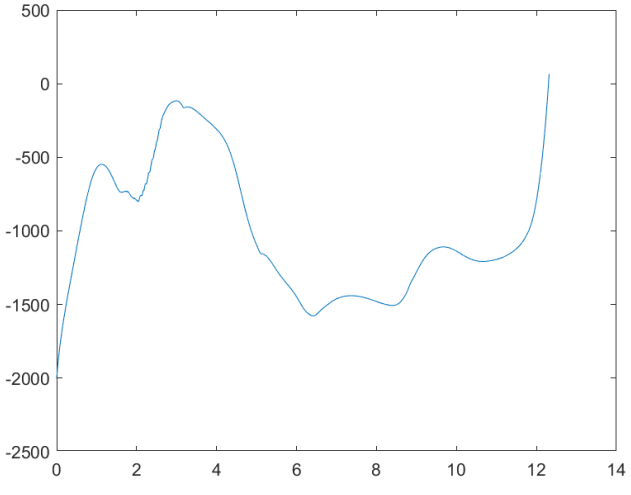


Figure B6: Example of after-training simulation (vertical velocity exceeding the upper limit).