Buletin Ilmiah Sarjana Teknik Elektro
Vol. 5, No. 4, December 2023, pp. 475-497
ISSN: 2685-9572, DOI: 10.12928/biste.v5i4.9407

Advancing UAV Path Planning System: A Software Pattern Language for Dynamic Environments

Gregorius Airlangga
Information Systems Study Program, Universitas Katolik Indonesia Atma Jaya, Jakarta, Indonesia

ARTICLE INFORMATION

Article History:

Submitted 28 October 2023
Revised 28 November 2023
Accepted 02 December 2023

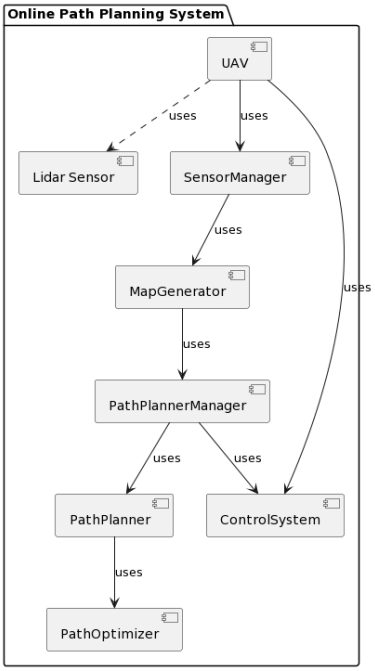
Keywords:

Kata Kunci 1;
Kata Kunci 2;
Kata Kunci 3;
Kata Kunci 4;
Kata Kunci 5

Corresponding Author:

Gregorius Airlangga,
Universitas Katolik Indonesia Atma Jaya, Jakarta, Indonesia.
Email: gregorius.airlangga@atmajaya.ac.id

ABSTRACT



In the rapidly advancing domain of Unmanned Aerial Vehicle (UAV) technologies, the capability to navigate dynamic and unpredictable environments is paramount. To this end, we present a novel design pattern framework for real-time UAV path planning, derived from the established Pattern Language of Program Community (PLOP). This framework integrates a suite of software patterns, each selected for its role in enhancing UAV operational adaptability, environmental awareness, and resource management. Our proposed framework capitalizes on a blend of behavioral, structural, and creational patterns, which work in concert to refine the UAV's decision-making processes in response to changing environmental conditions. For instance, the Observer pattern is employed to maintain real-time environmental awareness, while the Strategy pattern allows for dynamic adaptability in the UAV's path planning algorithm. Theoretical analysis and conceptual evaluations form the backbone of this research, eschewing empirical experiments for a detailed exploration of the design's potential. By offering a systematic and standardized approach, this research contributes to the UAV field by providing a robust theoretical foundation for future empirical studies and practical implementations, aiming to elevate the efficiency and safety of UAV operations in dynamic environments.

Document Citation:

G. Airlangga, "Advancing UAV Path Planning System: A Software Pattern Language for Dynamic Environments," *Buletin Ilmiah Sarjana Teknik Elektro*, vol. 5, no. 4, pp. 475-497, 2023, DOI: 10.12928/biste.v5i4.9407.

1. INTRODUCTION

The trajectory of Unmanned Aerial Vehicles (UAVs) has been marked by rapid technological progress, shifting their use from controlled, predictable environments to more complex and variable scenarios. This evolution necessitates a reassessment of path planning methodologies [1]-[3]. The foundational work by [4]-[6] laid the groundwork by detailing static path planning techniques, suitable for environments with minimal change. However, as UAV applications expanded into more dynamic settings, the limitations of these methods became increasingly apparent [7]-[8]. The transition to dynamic environments brought new challenges to the forefront of UAV path planning. The work from [9]-[12] spearheaded research into adaptive algorithms designed for real-time responsiveness. Their work illuminated the intricate challenges involved in balancing computational efficiency with the flexibility needed for unpredictable settings such as disaster relief or urban surveillance. This highlighted a significant gap in existing path planning methodologies - the need for a system capable of adapting on-the-fly to unforeseen changes in the environment.

Concurrently, the domain of software engineering has been witnessing a renaissance in the application of design patterns [13]-[15]. Pioneering studies by [16]-[18] demonstrated the transformative impact of these patterns in streamlining software development, optimizing processes, and enhancing system reliability. The author from [19]-[21] built upon this, showcasing how these patterns could be leveraged to tackle complex, modular challenges in scalable system architectures. However, the application of these sophisticated software engineering principles in the realm of UAV path planning remains scant, signaling an untapped potential [22]-[24]. This juxtaposition of advanced UAV technology and refined software engineering practices presents an unexplored frontier. The research gap is evident: there is a profound opportunity to harness software engineering patterns to meet the complex demands of dynamic UAV path planning. This confluence represents an untapped potential for enhancing UAV adaptability, responsiveness, and operational efficacy in real-time navigation. Addressing this lacuna, this study introduces a groundbreaking approach that amalgamates the structured methodologies of software engineering patterns with the nuanced requirements of dynamic UAV path planning. The research endeavors to create an online path planning system pattern that is inherently adaptable, efficiently responsive, and capable of navigating the multifaceted aspects of dynamic environments. This pattern is envisaged to revolutionize UAV path planning, elevating real-time decision-making processes, and significantly enhancing operational efficiency.

The ramifications of this research are extensive, with the potential to impact diverse sectors such as emergency response, environmental conservation, and urban infrastructure management. It serves as a crucial resource for UAV manufacturers, autonomous system software developers, and robotics researchers, offering a new lens through which to view the integration of software engineering principles with UAV navigational technology. The paper is structured as follows, section 2 delves deeper into the existing literature on UAV path planning and software engineering patterns, elaborating on the historical context, recent advancements, and identified gaps. In section 3 we detail the software pattern language template to develop the online path planning system pattern, emphasizing the integration of software engineering concepts. In Section 4, we provide an in-depth description of the proposed design pattern, delineating the functionalities of its various components. Finally, we conclude the paper by summarizing the key findings and contributions of the research and outlines potential future research directions in this area.

2. LITERATURE SURVEY

The evolution of Unmanned Aerial Vehicles (UAVs) has transitioned from basic navigation solutions to sophisticated path planning systems. This evolution has been largely driven by the expanding use cases of UAVs, from controlled, predictable settings to dynamic, unpredictable ones. The foundational research, exemplified by the studies of [25]-[27], focused on static path planning methodologies, effective in environments where variables remain constant. However, as UAV applications permeated more complex realms like disaster response, environmental monitoring, and urban surveillance, the static nature of these early path planning methodologies revealed significant constraints. They lacked the flexibility and responsiveness required to adapt to rapidly changing scenarios. Pioneering work by [28]-[30] marked a pivotal shift towards dynamic path planning.

This research highlighted the need for UAV systems capable of real-time adaptability, able to recalibrate paths instantaneously in response to unforeseen environmental changes and obstacles. In the realm of software engineering, the concept and application of design patterns have evolved significantly, offering structured solutions to recurring problems in software development. The work of [31]-[33] is pivotal in this respect, illustrating how design patterns can enhance the efficiency, robustness, and maintainability of software architectures. Following this, [34]-[36] expanded on the application of these patterns in complex system architectures, particularly emphasizing their role in scalability and modularity. However, the direct application of these structured software engineering strategies to UAV path planning, especially from the perspective of a software engineer, remains a largely unexplored area [37][38].

There exists a notable research gap in the synthesis of software design patterns with the adaptive and dynamic requirements of UAV path planning. The absence of integration of software design patterns in the development of UAV path planning systems. From a software engineering standpoint, the application of design patterns to UAV path planning is not only a novel concept but also a necessary evolution. The current landscape of UAV technology necessitates a paradigm shift where the adaptability and dynamism of UAV systems are complemented by the structured and systematic approach inherent in software design patterns. This integration is essential for enhancing the architectural quality of UAV systems, ensuring they are not only adaptable to environmental changes but also scalable, maintainable, and robust, adhering to the highest standards of software engineering. From a software engineer's perspective, addressing this gap involves reimagining UAV path planning through the lens of software design patterns. It calls for a harmonization of the principles of software architecture with the functional demands of UAV technology. This approach promises to revolutionize UAV path planning, making these systems more versatile, efficient, and aligned with established software engineering methodologies. The application of design patterns to UAV path planning is poised to address several key challenges, including system scalability in response to varying operational demands, maintainability in the face of evolving technological landscapes, and robustness against a range of environmental variables.

3. SOFTWARE PATTERN LANGUAGE TEMPLATE

Pattern language can be traced back to the field of architecture in the 1970s, when Christopher Alexander introduced an innovative system for organizing and interconnecting design patterns to create solutions to complex architectural problems [39]. This revolutionary idea obtained traction in the field of software engineering, where it manifested similar benefits, such as promoting reusable solutions, improving cohesiveness, fostering modularity, and ensuring extensibility [40]. Several distinguishing characteristics characterize pattern language. The ability to interconnect individual design patterns while emphasizing their relationships and providing guidelines for their use and integration is the most important [41]. This interconnectedness enables developers to implement patterns in a consistent and structured manner. Moreover, pattern language is typically tailored to a particular problem domain or system, providing a structured approach to resolving a wide variety of design problems within the given domain [42]. In addition, pattern language promotes iterative development, enabling developers to refine and improve their designs concurrently with a deeper comprehension of the problem domain and its requirements [43].

Within the scope of software engineering, pattern language offers numerous benefits. They provide reusable solutions to recurring problems in software design, like design patterns, but with a more comprehensive and structured approach that addresses a broader spectrum of problems within a particular domain [44]. Moreover, pattern language advocate for modularity, thereby promoting the separation of concerns and the division of a system into smaller, more manageable components, which improves maintainability and facilitates system comprehension, modification, and extension [45]. Extensibility is another notable advantage of pattern language. Software systems that adhere to established pattern language can be extended or adapted more easily to accommodate changing requirements and novel functionality, thereby ensuring the system's long-term viability and evolution [46]. Lastly, pattern language also can improve communication by establishing a common vocabulary and promoting a shared understanding among developers, thereby facilitating more effective communication, collaboration, and dissemination of knowledge [39]. The Pattern Template, as proposed by [47], presents a structured methodology for articulating and interpreting a design pattern. Below is a detailed description of each pattern subsection.

3.1. Intent

Intent of a pattern is an important aspect of its definition and usage. In Prof. Fernandez's pattern template [48], the intent describes the primary purpose or goal that the pattern aims to achieve. It is a summary statement that provides a high-level understanding of what the pattern does and why it is important. The intent typically answers questions such as: What does this pattern do? Why and when is this pattern useful? What kind of problem does this pattern solve? By addressing these queries, the intent encapsulates the fundamental essence of the pattern, making it easy to understand immediately.

Importantly, the intent also provides the rationale for the pattern's existence. In a pattern language or a collection of patterns, each pattern should have a unique intent that differentiates it from others. It must identify a distinct problem and provide a specific solution to it. This means that the intent should be specific enough to highlight its unique value, but broad enough to be applicable in various scenarios. Moreover, the intent serves as a guiding principle for the pattern's usage. It helps users decide whether the pattern is appropriate for their specific problem or situation. This is especially crucial in complex domains where there are multiple patterns to choose from. By clearly stating what the pattern does and when it should be used, the intent assists designers

or developers in selecting the most suitable pattern for their needs. The intent of a pattern is a critical element that defines its purpose, guides its usage, and sets it apart from other patterns. It is an essential component of a well-defined and useful pattern.

3.2. Example

Example subsection serves as a concrete illustration of the pattern in action, typically showing its application in a real-world context. The inclusion of an example is essential in demystifying the abstract concept of the pattern, as it helps users visualize how the pattern functions and how it can be applied to solve a specific problem. The example usually takes a practical scenario or situation and shows how the pattern can be applied within that context. It often details how the pattern addresses a specific problem, outlining the steps taken and the results achieved. This makes the pattern more relatable, by demonstrating its practical use and showing its effectiveness in addressing real-world challenges.

For instance, if a pattern is designed to enhance user authentication in web applications, an example might illustrate how the pattern can be implemented in an e-commerce website to securely verify users' identities and protect their personal information. The example would explain how the pattern is applied, what changes were needed, and how it improved the system's security. Examples can also highlight the versatility of a pattern by demonstrating its application in a variety of contexts or scenarios. This helps users understand the breadth of the pattern's applicability and its adaptability to different situations. Moreover, examples can showcase the pattern's handling of different "forces" or constraints in a real-world context. This provides users with a better understanding of how the pattern navigates the trade-offs and challenges in a practical scenario. An example in a pattern template is a practical demonstration of the pattern's application, offering a tangible illustration of how the pattern works and how it can be used to address real-world problems. It is a vital part of the pattern template as it bridges the gap between the abstract concept of the pattern and its practical utility.

3.3. Context

The context helps to set the stage for when and where a pattern might be applied most effectively. It offers a description of the situations or conditions under which the pattern can be used. The context section usually includes a description of specific situations where the pattern is applicable, the kinds of problems that it is designed to solve, and the conditions that must be in place for the pattern to work effectively. This can involve the broader environment in which the pattern operates, the specific technological constraints, or even the organizational factors that might influence the pattern's applicability. In more concrete terms, the context might outline aspects such as: Is the pattern suitable for a distributed system or a single-user application? Is it more useful in a real time system or a batch processing system? Does it require a particular infrastructure or programming paradigm to function effectively? All these specifics provide users with a clearer understanding of when and where to apply the pattern.

Importantly, the context can also articulate any preconditions or requirements that must be met for the pattern to be applicable. This could be certain system characteristics, user behaviors, or technological frameworks. For instance, a pattern for user authentication in a web application might have a context of an internet-connected system where users need to be uniquely identified and securely verified. Understanding the context is crucial for successful pattern application, as it ensures that the pattern is used in appropriate circumstances, thus optimizing its effectiveness, and preventing potential misuse or misapplication. The context also helps users to choose the most suitable pattern from among a set of possible patterns, based on their specific situation or requirements. The context of a pattern defines the specific scenarios or situations where the pattern can be most effectively implemented, detailing the relevant conditions, preconditions, and problem spaces. It's a critical component for correctly interpreting and applying a pattern.

3.4. Problem

The "Problem "section in a pattern template distinctly states the core issue or challenge that the pattern aims to address. This serves as a focal point for understanding the pattern's intent and application. The nature of the problem could span a wide range, from technical issues like managing concurrency in multi-threaded environments, to more conceptual challenges such as promoting modularity in large software systems. Typically, this section presents the problem as either a question or a statement. This format makes it easier for users to discern if the pattern is applicable to their specific situation. By articulating a clear problem statement, the pattern specifies a target for its solution and lays the foundation for a thorough explanation [49].

Frequently, this section incorporates a discussion on "Forces ". Forces refer to the conditions, limitations, or trade-offs that affect the problem and its possible solutions. For instance, a pattern addressing data security in a cloud environment would factor in forces such as regulatory compliance, performance impacts of encryption, and the need for user transparency. These forces shape the problem, defining its limits and affecting its potential solutions. Their recognition enables a comprehensive understanding of the problem, its

complexities, and prepares for a more effective solution. Understanding these forces is crucial when designing a suitable pattern. They can take the form of performance requirements, system limitations, security concerns, or usability trade-offs, and they influence the overall design and implementation of the pattern.

In line with Prof. Fernandez's pattern template [50], these forces are considered implicitly in the problem description, the context, and the solution sections. The pattern acknowledges the forces that need to be addressed, the context specifies the conditions under which the pattern operates, and the solution is designed to balance these forces effectively. For example, in a pattern designed to handle high-volume data processing in a cloud computing environment, forces might encompass network latency, data security requirements, and resource allocation. A well-constructed pattern addresses these forces, aiming to ensure efficient data processing, minimize latency, maintain security, and optimize resource utilization. A thorough understanding of these forces enables more effective pattern design and more informed decision-making when selecting and implementing patterns.

3.5. Solution

Solution represents the specific approach or method that the pattern proposes to solve the identified problem. It describes the general principle or strategy without being tied to a specific implementation, thus allowing for flexibility and adaptability. The solution is the heart of the pattern, demonstrating its practical value. It details how the pattern can be used to address the problem effectively, enhance efficiency, or improve other relevant aspects of the system or situation at hand. This typically includes a description of the entities involved, their relationships, their responsibilities, and the dynamics between them [51]. For example, if the problem described in the pattern is how to handle high-volume data processing in real time systems, the solution might propose a strategy like dividing the data into smaller chunks and processing these chunks concurrently, possibly across multiple servers. It's important to note that the solution doesn't prescribe a specific implementation, but rather provides a guideline or a blueprint.

This allows the pattern to be adapted to a variety of contexts and to be implemented using various technologies or programming languages. The solution should also demonstrate how it addresses and reconciles the forces described in the problem section. It should show how the proposed approach navigates the trade-offs and constraints that are inherent in the problem, providing a balanced and effective response to these forces. Furthermore, the solution may also describe the pat- tern's interaction with other patterns, showing how they can be combined or sequenced to address more complex problems or to create a more comprehensive design. The solution section of a pattern offers a general principle or strategy that effectively addresses the problem described earlier in the pattern. It is the core value proposition of the pattern, demonstrating how it navigates the forces at play and provides a beneficial impact on the system or situation.

3.6. Structure

The Structure section of pattern template visually represents the pattern through diagrams such as Unified Modeling Language (UML) diagrams or Block diagrams. This section offers a graphical depiction of the pattern, making it easier to comprehend and conceptualize. The structure of a pattern, in essence, provides a snapshot of how the pattern works. It typically illustrates the key elements of the pattern and their interactions, highlighting the roles, relationships, and collaborations among these elements. UML diagrams are a common choice for representing the structure of patterns because of their wide adoption in software engineering and their capability to depict complex designs in a standardized, comprehensible way. They may include class diagrams to show the static structure of the pattern, sequence diagrams to depict the interactions over time, or state diagrams to outline the states and transitions of a system following the pattern.

On the other hand, Block diagrams can be used when a more abstract, high-level view of the system is sufficient. They can effectively represent the main components of the pattern, their roles, and the interactions among them, without delving into the intricacies of the system. The Structure section is a powerful tool for understanding a pattern. It gives users a visual interpretation of the pattern, complementing the textual descriptions in other sections. With it, the pattern's design, its key elements, and their interactions become much clearer and easier to understand. By visualizing the pattern, users can more readily understand its mechanics, discern its flow, and identify its key elements. This understanding can enhance the user's ability to implement the pattern correctly and effectively in their own context. The Structure section provides a graphical representation of the pattern, helping to clarify its design, its key elements, and their interactions. It complements the textual descriptions in the other sections, making the pattern easier to understand and apply.

3.7. Dynamics

"Dynamics" in the context of pattern languages, describes the behavior of a pattern or system as it evolves over time, influenced by various internal and external factors. It elucidates how a pattern behaves, adapts, and

reacts to different scenarios, which is paramount for its effective use and adaptation to specific needs. We can further dissect the dynamics of a design pattern into three key components:

1. Interaction: Interaction signifies the relationships and interdependencies among different components within a pattern or system. Understanding these connections can help in anticipating systemic responses to changes in any part of the system.
2. Flow: Flow encapsulates the sequence and interaction of information, resources, or actions within a system. Gaining insights into this flow can provide a blueprint for the effective implementation of the pattern.
3. Behavior Under Different Conditions: This element describes how a pattern performs under a diverse array of conditions. Comprehending these variances helps in optimizing a pattern's effectiveness across a wide range of scenarios.

3.8. Implementation

The Implementation section of pattern template provides detailed guidelines on how to apply the pattern. This part of the template serves as a roadmap for developers, helping to ensure the pattern is effectively used and appropriately tailored to the specific context. The implementation guidance does not dictate an exact procedure but offers direction and advice to help users apply the pattern correctly. It typically involves a sequence of steps, considerations, or principles that users should follow when putting the pattern into practice. This can involve aspects like the order of operations, important considerations, potential pitfalls to avoid, common adaptations, and tips for success. For instance, if the pattern is related to database access in a web application, the Implementation section might provide guidelines on setting up the database connection, handling queries, managing transactions, dealing with exceptions, and ensuring security and performance.

While the Implementation section provides guidance on how to apply the pattern, it is not prescriptive and does not limit flexibility. The pattern can often be adapted to a variety of situations and can be implemented using various technologies or programming languages. Therefore, the Implementation section is intended to guide, rather than restrict, the user's approach to implementing the pattern. The Implementation section may also reference relevant standards, best practices, or industry conventions, helping users align their implementation with established norms. It can also cite examples of real-world implementations, helping users understand how the pattern can be applied in practice. The Implementation section of a pattern is a vital tool for users, offering guidance and advice on how to effectively put the pattern into practice. It helps ensure the pattern is correctly and effectively used, enhancing its value and impact.

3.9. Known Uses

The known uses in pattern template provides concrete examples where the pattern has been successfully applied. This section helps to establish the pattern's credibility and practicality, demonstrating its value in real-world contexts. Typically, the known uses section presents three or more examples of real systems or projects where the pattern has been effectively utilized. These examples can span a range of domains, applications, and technologies, illustrating the versatility and adaptability of the pattern. Each example usually includes a brief description of the system or project, the specific problem that was addressed, and how the pattern was applied to solve that problem. These details help to bring the pattern to life, demonstrating its practical application and effectiveness in resolving real-world issues. For example, in a pattern that addresses concurrency in multi-threaded environments, known uses could include instances where the pattern was applied in a database system, a web server, and a real-time data processing application.

These real-world examples serve as evidence of the pattern's utility and its practical value, enhancing its credibility and making it more relatable and understandable. They demonstrate that the pattern is not just a theoretical construct but a practical tool that has been proven to work in real systems. Moreover, these examples can also provide inspiration and guidance for those considering using the pattern, offering insights into how it can be applied and the benefits it can deliver. The known uses section of a pattern provides concrete, real-world examples of the pattern's application, demonstrating its practical value, enhancing its credibility, and offering inspiration and guidance for its potential use.

3.10. Consequences

The consequences section articulates the results of applying the pattern, including both its advantages and potential drawbacks. Understanding the consequences of a pattern's use is fundamental to evaluating its suitability for a given context and making informed decisions about its implementation. The consequences of a pattern can manifest in several ways, and they extend beyond the immediate solution to the problem at hand. They can impact system properties such as performance, scalability, maintainability, or security, and might also influence aspects like development time, complexity, and the learning curve for developers. Advantages

of applying a pattern might include facilitating code reuse, improving system modularity, enhancing scalability, or increasing security, to name just a few.

These positive outcomes typically align with the intent of the pattern and help solve the problem identified in the template. However, every solution also carries potential liabilities or trade-offs, which are equally important to understand. For example, while a pattern might improve system performance, it could potentially increase code complexity or require additional resources. Identifying these consequences helps to highlight the trade-offs involved in using the pattern and to understand the impact on other aspects of the system or process. It ensures that developers and designers have a well-rounded understanding of the pattern, enabling them to make informed decisions about its use. The consequences section of a pattern provides an in-depth understanding of the impact of applying the pattern. It articulates both the advantages and potential liabilities, helping users to make informed decisions and to anticipate and manage the trade-offs involved in the pattern's application.

3.11. Related Patterns

The Related Patterns section in pattern template plays a significant role in demonstrating the pattern's relationship and interaction with other patterns. This section provides a broader context for the pattern and shows how it fits into the larger landscape of pattern language. Related Patterns can be those that complement the current pattern, offering solutions that can be used in combination with it, or those that offer alternative solutions to the same problem. This can help users understand different approaches to the same problem and decide which pattern best fits their specific needs. For instance, if the current pattern describes a way to handle concurrency in a multithreaded system, related patterns could include ones that describe different strategies for concurrency management, or ones that describe how to handle related issues like synchronization or deadlock prevention.

Additionally, some patterns naturally fit together in a sequence, forming a pattern sequence or pattern language. In such cases, the Related Patterns section can describe this relationship, explaining how the current pattern fits into this sequence and interacts with the other patterns in it. In some cases, variants of the pattern that fit specific contexts might also be included in the Related Patterns. This shows that the pattern is adaptable and can be modified to fit different situations. By providing this information, the Related Patterns section helps users understand the larger context for the pattern and its interactions with other patterns. This can inform their decisions about how and when to use the pattern, and how it can be combined with other patterns to address more complex problems or to create a more comprehensive design.

4. PROPOSED ONLINE PATH PLANNING SYSTEM PATTERN DESIGN

4.1. Intent

The principal intent of the online path planning system pattern for a single UAV is to enhance the path and motion planning process in terms of efficiency, accuracy, and adaptability within dynamic environments. It introduces an online scheme where the environmental assessment and planning calculations occur concurrently and in real-time when the UAV undertakes a mission. This approach negates the need for a predetermined path or prior knowledge of the map, thus allowing for a more flexible and adaptive navigation system. This pattern is designed to allow the UAV to respond to unforeseen changes in its environment promptly and effectively, thereby improving its overall navigation performance and ability to complete its mission successfully.

4.2. Example

Consider a scenario where a search and rescue drone is deployed in an area struck by a natural disaster, such as an earthquake or a flood. The area is filled with unpredictable obstacles and the landscape has drastically changed due to the disaster. Traditional navigation methods, reliant on predetermined paths and prior knowledge of the terrain, might fail in this dynamic environment. Here, the Online path planning system pattern becomes indispensable. As the drone embarks on its mission to locate survivors, this pattern enables real-time assessment of the environment and dynamic path planning. Thus, the drone can efficiently navigate the complex environment, avoiding obstacles, altering its path based on new information, and adapting in real-time to ensure the optimal route for successful mission completion [52]. Further, imagine a drone navigating through a dense forest with varying altitude and numerous obstacles such as trees, branches, and rocks. The terrain is complex and the environment is constantly changing due to factors like wind and wildlife. A predetermined path or prior map knowledge might not be sufficient for safe and efficient navigation in this case. Utilizing the online path planning system pattern, the drone can adapt its path in real-time, identifying and avoiding potential collisions, and navigating efficiently through the forest. This pattern hence empowers

the UAV with the capability to adapt to unexpected changes, improving its performance and effectiveness in dynamic environments.

4.3. Context

The path planning for a single UAV pattern is applicable in various contexts where an autonomous system like a UAV is required to operate in a dynamic environment. Here, 'dynamic environment' denotes environments that are subject to frequent or unpredictable changes, which can affect the UAV's path and overall mission.

1. UAV in Changing Landscapes: One of the most common applications of this pattern is in the operation of UAV in changing landscapes. These could be natural environments such as forests with varying altitudes and obstacles, disaster-stricken areas with sudden changes due to earthquakes or floods, or urban environments where construction or traffic might alter the UAV typical path. In these scenarios, the UAV's environment is not static. Obstacles can appear, disappear, or move, and the terrain might change due to natural or human-induced factors. Therefore, the UAV requires an online motion system that can adapt to these changes in real-time. The system must consider the UAV's current speed, direction, and orientation to navigate efficiently and safely.
2. UAV in Cluttered Indoor Spaces: Another significant context of application for this pattern is in the operation of UAV in cluttered indoor spaces. These could be warehouses, factories, hospitals, or even homes, where the UAV's path might be filled with moving or stationary obstacles. Unlike a static environment where the layout remains the same, cluttered indoor spaces can change frequently. Items or structures might be moved, new obstacles might appear, and humans or other robots might cross the UAV's path. Therefore, the UAV needs an online path planning system pattern that can adapt to these changes promptly. It needs to consider its current speed, direction, and orientation to efficiently navigate through the cluttered space without collisions.

4.4. Problem

Traditional path planning algorithms have significant limitations when dealing with dynamic and unpredictable environments. They typically rely on static maps or predetermined paths, operating under the assumption that the environment will remain unchanged during the UAV's operation. In reality, environments can be dynamic and constantly changing due to various factors like wind, moving obstacles, or terrain alterations. Moreover, these algorithms often fail to consider the orientation and direction of the moving object. In the context of drones, these factors are crucial for efficient path planning, especially in complex environments. Ignoring these aspects might lead to inefficient routes, potential collisions with obstacles, or the inability to navigate through narrow spaces.

For instance, a search and rescue drone operating in a disaster-stricken area or a forest surveillance drone might encounter unexpected obstacles or terrain changes. Without the ability to dynamically adapt to these changes in real-time, the drone might crash into obstacles or fail to reach its destination, thereby failing its mission. This pattern, which includes the propulsion system, control surfaces, and the onboard control algorithms, directly influences the UAV ability to maneuver and orient itself in response to the path planning algorithm. Traditional motion systems often employ a simple, reactive approach, responding to immediate obstacles or environmental changes rather than proactively planning for efficient navigation. They may also fail to effectively consider the UAV current orientation and direction in real-time, leading to poor efficiency in path following. For example, a UAV might need to make a sharp turn to avoid an obstacle. If the motion system does not consider the drone's current speed, inertia, or orientation, the UAV might not be able to make the turn in time or efficiently, leading to suboptimal path following or even collisions.

The problem extends beyond improving the path planning algorithm itself. It requires enhancements in the motion system of the UAV to respond more effectively and efficiently to dynamic path planning in real-time. This involves accounting for the UAV's current speed, direction, and orientation. The ultimate goal is to ensure safe, efficient, and adaptive navigation in dynamic environments. Thus, both the efficiency, safety, and adaptability of the path planning process and the UAV's motion system need significant improvements to tackle the challenges posed by dynamic environments.

4.5. Solution

The pattern proposes a solution that integrates an improved path planning algorithm with a sophisticated UAV motion system. Our main component of this solution contains of eight modules such as modified A* algorithm, map generator by using Lidar, altitude information storage, partial goal estimator, adaptive module manager, direction pattern generator, automatic speed & altitude configuration and the last is collision avoidance module can be seen in Figure 1.

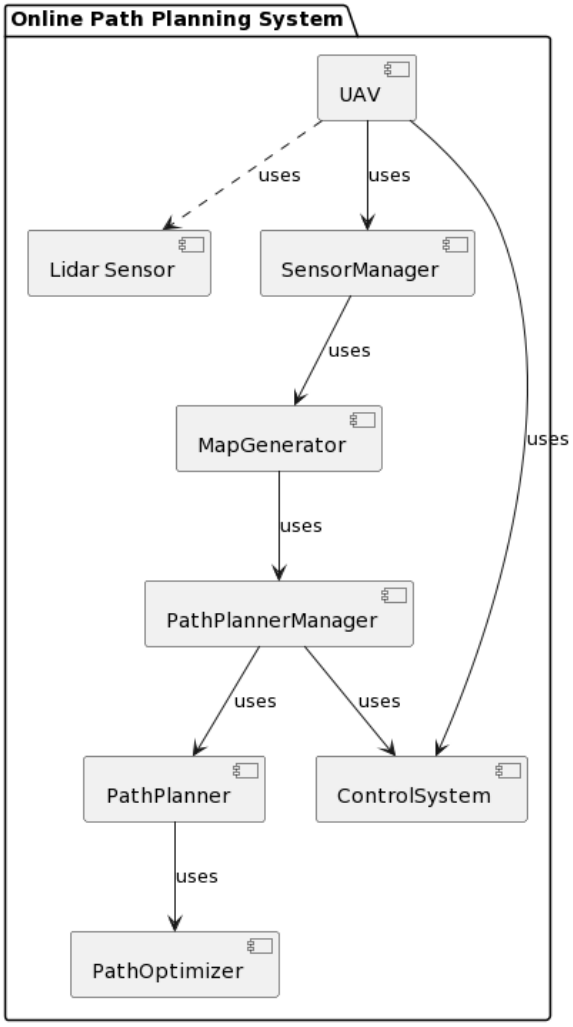


Figure 1. Online path planning component diagram

4.6. Modified A* Algorithm (Direction Aware Algorithm)

This module features a modified version of the A* algorithm, a widely used path planning algorithm in robotics. The standard A* algorithm excels at finding an optimal path from a start point to an endpoint while avoiding obstacles. However, it has limitations, particularly with respect to accounting for the UAV's current orientation or direction. This module aims to address those limitations and enhance the algorithm's capabilities.

Standard A* Algorithm: The traditional A* algorithm uses a grid-based approach, assigning costs to each grid cell or node based on its distance from the start and end points and any obstacles present. However, this algorithm isn't inherently direction aware, meaning it doesn't factor in the UAV's current orientation when selecting the next move. This could lead to inefficient paths in certain scenarios, particularly in complex and dynamic environments. Direction-Aware Modifications: The modified A* algorithm in this solution incorporates the UAV's current orientation into its calculations. By taking this direction aware approach, the algorithm can select the next move that is not only efficient in terms of distance but also aligns better with the UAV's current direction. This approach reduces unnecessary turns and detours, making navigation more efficient, particularly in complex environments.

Real-Time Performance: An essential feature of this modified algorithm is its ability to compute paths quickly or in 'real-time'. This is particularly important in dynamic environments where obstacles might appear or move suddenly. In such cases, the algorithm needs to swiftly recalculate the path to avoid new obstacles. Interaction with Other Modules: The output from this module feeds into the direction Pattern Generator and the Adaptive Module Manager. If the Lidar-based Map Generator detects a new obstacle, the modified A* algorithm can quickly re-plan the path to avoid it. Also, if the Collision Avoidance Module detects a risk of collision, the path can be instantly adjusted.

4.7. Map Generator by using Lidar

This module uses Lidar technology to produce a detailed 2D map of the UAV's environment in real-time. This up-to-date environmental information is critical for the UAV's efficient navigation and safety. Lidar Technology: Lidar, which stands for Light Detection and Ranging, is a remote sensing method that uses pulsed laser light to measure distances to objects. By measuring the time delay between the emission of a laser pulse and the detection of the reflected signal, Lidar can determine the distance to the object that reflected the signal. It can perform these measurements in multiple directions to build a detailed picture of the surrounding environment.

Real-Time Map Generation is the Map Generator module that takes the distance measurements from the Lidar and uses them to create a mini 2D map of the UAV's immediate surroundings. This map is updated for each time iteration, meaning that a new map is generated in real-time as the UAV moves and the Lidar collects new measurements. These continual updates ensure the map accurately reflects the current state of the environment, including any moving or newly appeared obstacles. Obstacle Detection is a real-time 2D map that greatly enhances the UAV's ability to detect and avoid obstacles. The map provides a clear representation of where obstacles are in relation to the UAV, enabling other modules, such as the Collision Avoidance Module, to take appropriate action if necessary. Integration with other modules concept is not only used for obstacle detection but also feeds into other modules, such as the Modified A* Algorithm, which uses the map to plan the most efficient path, and the Direction Pattern Generator, which uses it to determine the best orientation for the UAV.

4.8. Altitude Information Stored in a Hash Map

The Altitude Information Stored in a Hash Map module is a vital part of the system that efficiently handles and stores altitude data for the UAV. This information is integral to the UAV's ability to navigate complex environments with variations in altitude. There are several important modules such as Hash Map Data Structure, Key-Value Pairs, Data Acquisition and Enhanced Environment Awareness.

Hash Map Data Structure module uses a hash map data structure. A hash map stores data in pairs of keys and values, and it is designed to enable extremely fast access to the values when given the corresponding keys. This fast access is due to a technique called hashing, where each key is processed through a hashing function to produce a unique index (the 'hash') that determines where the paired value is stored. Key-Value Pairs is the module that stores the keys in the hash map for coordinates of the UAV's path, and the corresponding values would be the altitude information at those coordinates. This setup allows the system to quickly retrieve the altitude of any location on the UAV's path just by providing its coordinates. Data Acquisition is comprising of altitude information that could be gathered from various sources such as onboard altimeters, terrain data maps, or real-time Lidar scanning.

Enhanced Environment Awareness concept means by storing altitude data in this way, the module provides the UAV with a method of quickly accessing altitude information, improving its environmental awareness. This is particularly valuable when navigating through complex environments with significant altitude variations, like hilly landscapes, multi-store urban environments, or disaster-stricken areas. Integration with Other Modules means the altitude information can be used by other modules, such as the Automatic Speed & Altitude Configuration, Modified A* Algorithm, and Collision Avoidance Module. For example, when planning a path, the Modified A* Algorithm could use the altitude information to avoid steep inclines or to navigate around high obstacles. The last is partial goal estimator, the module operates under the principle that when a target location is distant or in a complex environment, it's more effective and computationally efficient to plan the path in segments, rather than attempting to compute the entire path in one go. This is because the full path from the start to the end point could be exceptionally long, contain numerous potential obstacles, or might change over time due to dynamic factors in the environment. Here is a step-by-step breakdown of how this module works:

1. Path Segmentation: This process starts with dividing the entire journey from the starting point to the destination into smaller, manageable segments or 'chunks'. Each of these segments represents a 'partial goal' that the UAV needs to reach before proceeding to the next one.
2. Route Planning: For each of these partial goals, the UAV, with the help of the Modified A* Algorithm module, plans a route from its current location to the partial goal. This segmentation of path planning tasks makes the process less computationally intensive, as the UAV doesn't have to process the entire path all at once.
3. Dynamic Adjustments: As the UAV starts moving towards a partial goal, it continually scans the environment using its onboard sensors (like Lidar). If it detects new obstacles, it will then adjust its planned path to avoid these obstacles. This ensures that the UAV's path planning remains flexible and adaptable to changes in the environment. The new path to the partial goal is calculated by the Modified A* Algorithm, keeping in mind the UAV's current location and orientation.

4. Repetition: Once a partial goal is reached, the UAV moves onto the next segment of the journey and repeats the process until it reaches the destination.

4.9. Adaptive Module Manager

This approach allows for efficient path planning and a quick response to changes in the environment. The reduction in computational load facilitates real-time processing and quick decision-making, essential in a dynamic environment where obstacles could appear unexpectedly. Furthermore, it enables the UAV to optimize its path based on the most current information available about the environment, making its navigation safer and more efficient. The Adaptive Module Manager essentially serves as the central coordination system or "brain" of the UAV's operation. Its primary role is to ensure smooth communication and cooperation between all the other modules, adapting the UAV's behavior according to the incoming data. Here is a more detailed explanation:

1. Data Reception and Analysis: The Adaptive Module Manager constantly receives data from all other modules in the system. This data could be regarding the UAV's current location, the detected obstacles, altitude information, or direction pattern. The manager analyzes this data to understand the status and situation of the UAV.
2. Decision Making: Based on the received data, the Adaptive Module Manager makes decisions on how the UAV should adjust its behavior. This decision-making process considers the current state of the UAV, the final goal, and the environmental conditions. It decides which module needs to be activated, deactivated, or adjusted based on the data received.
3. Signal Transmission: Once the decisions are made, the manager sends signals to the appropriate modules to execute the necessary actions. For example, if the Lidar detects a new obstacle, the manager will send a signal to the Modified A* Algorithm module to recalculate the path, avoiding the obstacle. Similarly, if the altitude data indicates a change in the landscape, the manager might signal the Automatic Speed & Altitude Configuration to adjust the UAV's altitude accordingly.
4. Adaptive Behavior: One key feature of the Adaptive Module Manager is its ability to adapt to changing conditions. Unlike a rigid system, it can modify its decisions and signals based on the updated data it receives. This flexibility allows the UAV to react in real-time to changes in its environment or flight conditions, enhancing its safety and efficiency.
5. Continuous Monitoring: Throughout the flight, the Adaptive Module Manager continuously monitors the operation of all other modules and the overall behavior of the UAV. This constant monitoring ensures that all systems are working as intended and allows for quick adjustments if any issues are detected. In essence, the Adaptive Module Manager acts as a bridge between the various modules, facilitating efficient information exchange and decision-making processes, ultimately ensuring the smooth operation and adaptation of the UAV in response to dynamic conditions.

4.10. Direction Pattern Generator

The Direction Pattern Generator plays an essential role in guiding the UAV from its current location to its desired destination. It translates the planned path (created using the Modified A* Algorithm) into a series of step-by-step directional commands that the UAV can follow. This process involves both maintaining a sense of the UAV's global direction towards its final goal and considering its local navigation needs based on immediate environmental factors.

1. Receiving Path Information: The Direction Pattern Generator starts its work by receiving the planned path from the Modified A* Algorithm. This path information is usually in the form of a sequence of points or nodes that the UAV needs to pass through to reach its destination.
2. Generating Directional Instructions: The module then translates this sequence of points into a series of directional instructions or a 'pattern'. Each instruction guides the UAV from one point to the next. These instructions are not just simplistic commands like "go left" or "go right", but they consider the UAV's current orientation, the required change in direction, the distance to the next point, and any potential obstacles.
3. Managing Orientation and Trajectory: The generated pattern also keeps the UAV's orientation and trajectory optimized. By carefully calculating each turn or change in altitude, the module ensures that the UAV doesn't make any abrupt or inefficient movements. This minimizes energy usage and wear and tear on the UAV.
4. Dynamic Adjustments: If the Adaptive Module Manager signals a change in the path due to a new obstacle or change in the environment, the Direction Pattern Generator will create a new set of directions based on the updated path. This ensures the UAV can adapt to changes in real-time.

5. Relaying Directions to UAV: The series of directions is then relayed to the UAV control system which in turn adjusts the UAV's motors and control surfaces to follow the given directions. The Direction Pattern Generator is thus instrumental in turning a high-level path into actionable, moment-to-moment navigation instructions. Its functions enhance the overall stability, efficiency, and safety of the UAV's flight.

4.11. Automatic Speed and Altitude Configuration

This module is crucial in managing two fundamental parameters of UAV's flight:

1. Speed and altitude: During a flight, a UAV might need to vary its speed and altitude based on the characteristics of its environment, like the presence of obstacles, variations in terrain elevation, and weather conditions. This module enables real-time adjustments of these parameters to ensure the safety and efficiency of the UAV's flight.
2. Data Gathering: The Automatic Speed & Altitude Configuration module receives and processes data from several other modules. This includes altitude information from the Altitude Information Storage and environmental data from the Mini 2D Map Generated by Lidar. It may also take inputs from other sensor modules that provide data about wind speed, air pressure, and other environmental factors.
3. Situation Analysis: Once the data is collected, the module analyzes it to assess the current situation. For example, if the Lidar detects a steep hill ahead, the module will recognize the need to adjust the UAV's altitude and speed.
4. Decision Making: Based on its analysis, the module then makes decisions about the optimal speed and altitude for the UAV. These decisions aim to balance the efficiency and safety of the flight. For example, in the presence of a steep hill, the module might decide to increase the UAV's altitude to clear the hill and reduce the speed to manage the ascent safely.
5. Implementation of Adjustments: The decisions are then converted into control signals and sent to the UAV's control system. These signals adjust the UAV's motor speeds, propeller angles, and other control parameters to change the UAV's speed and altitude as decided.
6. Dynamic Adaptation: This process is not a one-time calculation but happens continuously throughout the flight. The module dynamically adapts the speed and altitude based on the most recent data, enabling the UAV to react to changes in the environment in real time.
7. In essence, the Automatic Speed & Altitude Configuration module plays a pivotal role in maneuvering the UAV through different environments. By making real-time adjustments to the UAV's speed and altitude, it helps the UAV navigate safely, efficiently, and effectively in various conditions.

4.12. Collision Avoidance

The Collision Avoidance Module is a critical part of the UAV's operations, functioning as a real-time safety mechanism to prevent potential collisions with obstacles in the environment. These obstacles can be either static (like buildings or trees) or dynamic (like birds or other moving objects):

1. Data Collection: The first step is to gather environmental data from the UAV's on- board sensors. These sensors, which can include Lidar, radar, and cameras, constantly scan the surrounding area to provide a detailed and up-to-date view of the environment. Each sensor offers a unique type of data. Lidar and radar provide accurate distance and velocity measurements, while cameras can supply visual context and help in recognizing specific types of obstacles.
2. Risk Assessment: The module then processes this data to identify potential hazards. It calculates a risk of collision for each identified obstacle, factoring in the UAV's current speed, direction, and proximity to the obstacle. This risk calculation is continuously updated as new sensor data comes in and as the UAV's flight parameters change.
3. Decision Making: If the calculated risk of collision exceeds a certain threshold, the module decides that an evasive action is necessary. This decision is based on several considerations, including the type and size of the obstacle, the UAV's current flight parameters, and the available maneuvering options.
4. Evasive Action: The chosen evasive action can take several forms, such as altering the UAV's flight path, speed, or altitude. The goal is to ensure the UAV avoids the obstacle safely while minimizing any disruptions to its flight plan. These changes are implemented by sending appropriate control signals to the UAV's flight control system.
5. Communication with Other Modules: When a potential collision is detected and avoided, this module communicates the data to other components like the Modified A* Algorithm and the Direction Pattern Generator. They use this information to adjust the UAV's path and direction instructions, maintaining safety and efficiency. The module can also override current actions for immediate evasive maneuvers in case of sudden, unpredictable obstacles. The Collision Avoidance Module thus plays an indispensable role in the UAV's operation, particularly in complex and dynamic environments. By proactively detecting and avoiding potential collisions, it greatly enhances the reliability and safety of the UAV system.

4.13. Structure

The class diagram provided offers a detailed representation of a dynamic path planning system for UAVs. Starting with the LiDAR class, it serves as the primary source of sensor data for the entire system. The data it provides is vital for generating an accurate local map of the UAV's environment, allowing for informed path planning. Additionally, the LiDAR class continuously updates sensor data as the UAV navigates, ensuring the system's real-time responsiveness to environmental changes. Upon receiving raw sensor data from the LiDAR, the *SensorManager* class processes this information and produces a *ProcessedSensorData* object. As the intermediary between the LiDAR sensor and the UAV, this class is responsible for managing data flow and updating the processed sensor data to maintain an up-to-date representation of the UAV's surroundings can be seen in Figure 2.

The UAV class serves as the core component of the system, executing motion control based on the generated path and interacting with other classes to achieve successful navigation. Not only does the UAV send the processed sensor data to the *MapGenerator* class, but it also commands the *ControlSystem* class, ensuring proper motion control in accordance with the generated path. Taking the processed sensor data from the UAV, the *MapGenerator* class creates a 2D Mini Map and Altitude HashMap. In doing so, it provides the UAV with a clear understanding of its environment, essential for efficient path planning.

Furthermore, the *MapGenerator* class can update both the Mini Map and Altitude HashMap as the environment evolves, guaranteeing real-time adaptability. Acting as the coordinator for the path planning process, the *PathPlannerManager* class oversees the entire operation. This class is responsible for requesting and updating paths from the *PathPlanner* class, as well as communicating with the *ControlSystem* class to determine the UAV's motion. It serves as the bridge between the path planning and control system classes.

The *PathPlanner* class calculates an initial path and updates it based on the *MiniMap* and Altitude *HashMap* provided by the *MapGenerator.* This class is in charge of generating the safest and most efficient path for the UAV, taking into account the current environmental conditions. To further refine the generated path, the *PathOptimizer* class optimizes it for safety and efficiency. By receiving the path from the *PathPlanner* class, enhancing its overall quality, and returning the optimized path to the *PathPlannerManager*, it ensures the UAV follows the best possible route. Finally, the *ControlSystem* class manages the UAV's motion based on the received path. It works closely with the *PathPlannerManager* to receive and update the path, and it communicates with the UAV class to execute motion control accordingly. In conclusion, this class diagram showcases an intricate and interconnected system for dynamic path planning in UAVs.

Each class contributes a specific function and collaborates with others to create a solution for navigating dynamic environments. The integration of LiDAR, sensor management, map generation, path planning, optimization, and control systems guarantees the UAV's ability to reach its destination safely and efficiently. Due to the complexity, we only depict the structure that focusing on the online path planning algorithm, the main structure is involving multiple components such as modified A* algorithm, Lidar-generated mini 2D maps for each time iteration, and a hash map for altitude information. To begin, the LiDAR sensor serves as the primary data acquisition device, gathering crucial environmental information. This raw sensor data is then provided to the *SensorManager*, which is responsible for processing and filtering the collected data to ensure its reliability and accuracy. Once the data has been processed, it is subsequently forwarded to the UAV. As the central hub for coordinating all actions, the UAV is responsible for dispatching the processed data to the *MapGenerator* component. Consequently, the *MapGenerator* utilizes this data to create a 2D Mini Map, which provides a bird's-eye view of the environment, as well as an Altitude *HashMap,* which contains information about the elevation of various points in the environment. These generated maps are then passed on to the *PathPlannerManager*. Following this, the *PathPlannerManager* plays a crucial role in requesting the most efficient path from the *PathPlanner*, which takes into consideration the generated 2D Mini Map and Altitude HashMap. In order to ensure the path is not only efficient but also safe, the *PathPlanner* forwards the initially calculated path to the *PathOptimizer*.

The *PathOptimizer* is responsible for refining the path by taking into account various factors, such as potential obstacles and environmental constraints. Once the path has been optimized, it is returned to the *PathPlannerManager*. Subsequently, the *PathPlannerManager* communicates the optimized path to the *ControlSystem*, which is in charge of commanding the UAV to execute motion control based on the generated path. The *ControlSystem* ensures that the UAV follows the designated path while maintaining stability and adhering to any predefined motion constraints. During the entire process, an update cycle loop is in place to account for dynamic changes in the environment. The LiDAR sensor continuously updates the sensor data, which is processed by the *SensorManager* and sent to the UAV. The UAV then forwards this updated processed data to the *MapGenerator*, resulting in updates to the 2D Mini Map and Altitude *HashMap*. Consequently, the *PathPlannerManager* requests an updated path from the *PathPlanner*, which relies on the *PathOptimizer* to

refine the updated path. The optimized path is once again returned to the *PathPlannerManager* and then to the *ControlSystem*. As the UAV follows the updated path, it is essential to monitor its progress towards the goal. The UAV checks the goal position and communicates its status to the *PathPlannerManager*. If the goal has not been achieved, the *PathPlannerManager* instructs the *ControlSystem* to continue moving the UAV along the updated path. However, if the UAV has successfully reached its goal, the *ControlSystem* commands the UAV to cease movement and execute a safe landing procedure.

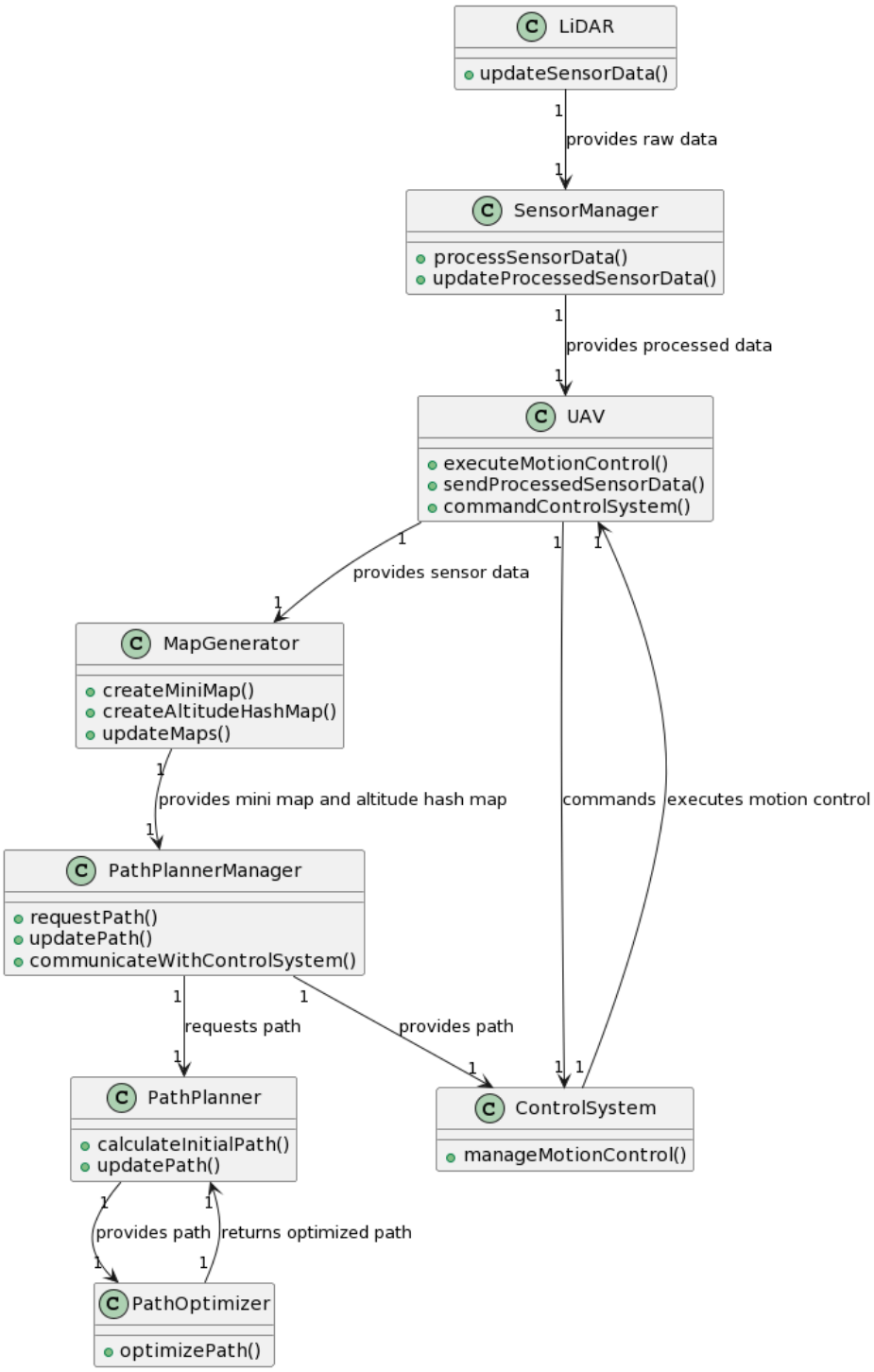


Figure 2. Online path planning system class diagram

4.14. Dynamics

First, the UAV's task is to read sensor data. The UAV comes equipped with various onboard sensors such as Lidar, accelerometers, and gyroscopes. The raw data from these sensors form the basis for understanding

the UAV's current state and the surrounding environment. This data collection is paramount in providing real-time information about the UAV's surroundings and its physical state. Following data collection, the UAV moves to the process sensor data step. Here, it processes the collected raw sensor data, eliminating noise and transforming the data into a more usable and understandable format. This stage may involve the implementation of a sensor fusion algorithm. Such an algorithm merges data from various sensors, creating a more comprehensive, accurate depiction of the UAV's status and the environment can be seen in Figure 3.

Using the processed sensor data, the UAV then generates mini maps and an altitude HashMap. The UAV creates a mini 2D map offering a bird's-eye view of its environment. This map displays the landscape and any potential obstacles in the UAV's path. Concurrently, the UAV constructs an Altitude HashMap representing the varying heights or elevations in the environment. These pieces of information play a crucial role in planning the UAV's flight path. The next stage involves path planning. The UAV generates a preliminary flight path based on the Mini Map and Altitude HashMap. It considers factors like the UAV's current position, its destination, and potential obstacles in the path. With this information, it calculates a feasible path to its destination. Having established an initial path, the UAV proceeds to optimize the path. At this juncture, the UAV refines the initially calculated path. Optimization ensures the safety of the UAV while enhancing efficiency. In this step, the UAV considers various factors like wind resistance, battery usage, and overall flight time, adjusting its path accordingly.

Finally, the UAV executes control signals. This involves the actual execution of the flight based on the optimized path. The UAV communicates with its propulsion and steering systems, issuing control signals for movement along the planned path. This process is subject to constant updates and adjustments depending on real-time changes in the environment or the UAV's physical state. In this system, the Environment acts as a secondary actor. Though it does not actively perform actions, it greatly influences the UAV's operations. Its role is to provide environmental changes. The environment is dynamic, changing constantly and often without warning. Changes range from sudden shifts in weather conditions to the introduction of new obstacles in the UAV's path. Consequently, the UAV must adapt its flight plan in real-time, ensuring a safe and efficient response to these environmental changes can be seen in Figure 4.

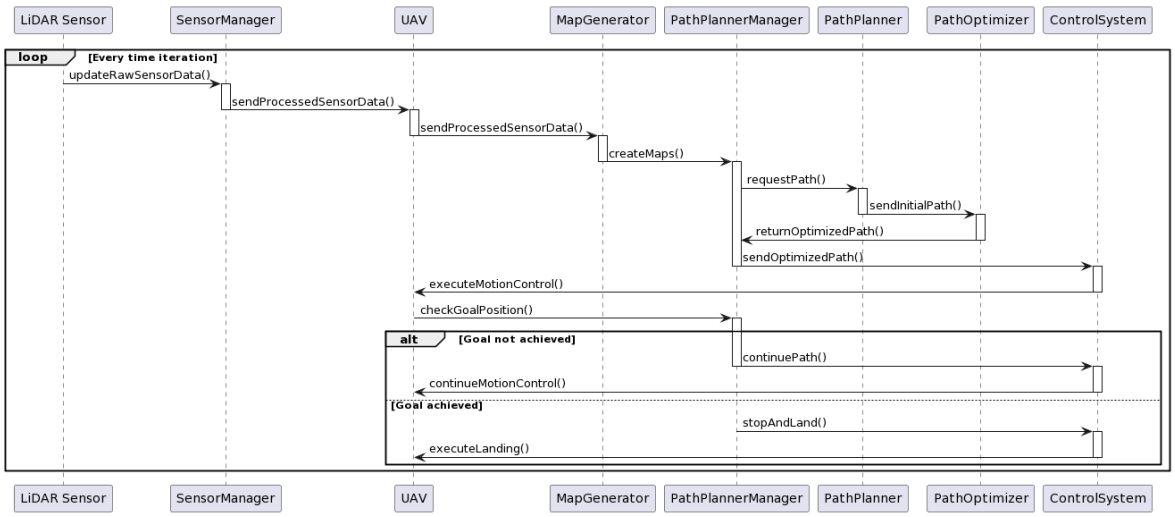


Figure 3. Online Path Planning Sequence Diagram

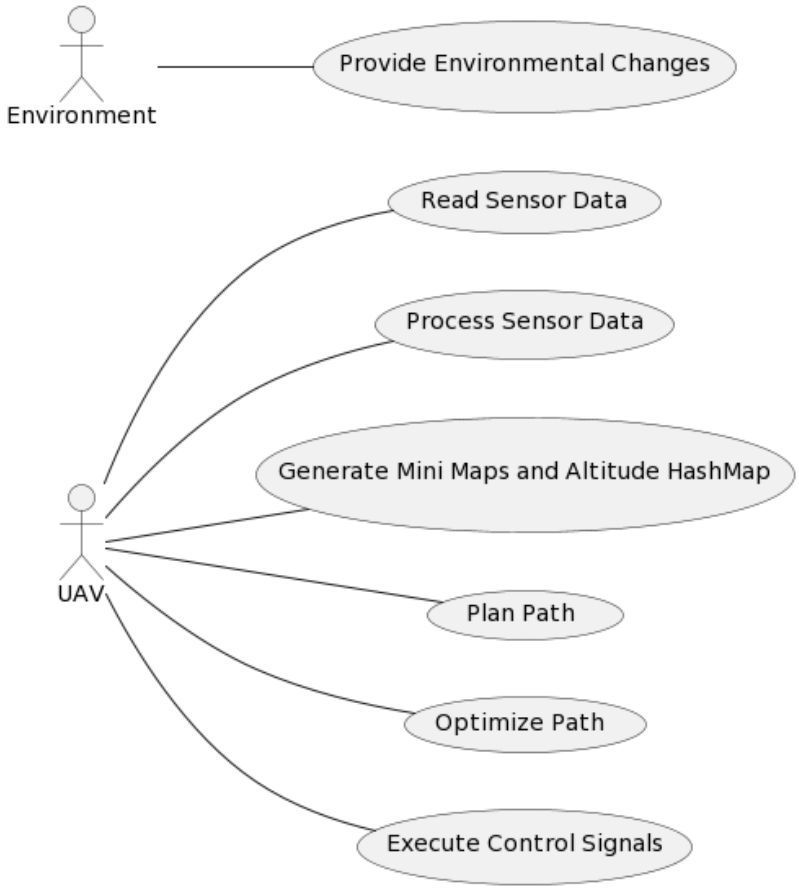


Figure 4. Use case diagram

4.15. Implementation

Implementing the pattern is a multi-step process that requires careful attention to various technical details. It involves the integration of sensor data, specifically Lidar and altitude information, into the path planning algorithm, and ensuring that the algorithm is capable of handling real-time adjustments and is direction aware. Below is the explanation of the process can be seen in Figure 5.

1. Understanding the Environment: The first step in implementing this pattern is to thoroughly understand the environment in which the UAV will operate. This includes comprehending potential obstacles, dynamic elements, and variations in the terrain. The environmental characteristics will guide the development and modification of the path planning algorithm and influence the calibration of the UAV's motion system.
2. Sensor Selection and Integration: The next step involves selecting appropriate sensors that can accurately perceive the environment in real-time. This typically includes Lidar sensors for creating 2D or 3D maps, accelerometers and gyroscopes for understanding the UAV's movement and orientation, and possibly other specific sensors depending on the application. It's important to ensure that the data from these sensors can be integrated and processed efficiently and accurately in real-time, which may require the development of efficient sensor fusion algorithms.
3. Development and Optimization of the Path Planning Algorithm: Developing a path planning algorithm that considers the dynamic nature of the environment and the UAV's direction and orientation is a crucial step. This involves modifying traditional path planning algorithms to work with real-time data and account for direction-awareness. The algorithm should be optimized to run efficiently in real-time while utilizing potentially limited computational resources onboard the UAV.
4. Integration of Lidar Data and Altitude Information: To enhance the UAV's environment awareness, you will need to integrate the Lidar-generated mini 2D maps and hash map-based altitude data into the path planning algorithm. This integration requires careful attention to data processing to ensure accuracy and efficiency.
5. Upgrading the UAV's Motion System: The UAV's motion system needs to be upgraded to effectively respond to the outputs of the path planning algorithm. This requires adjusting the control surfaces and

propulsion system, and developing onboard control algorithms that can control the UAV's speed, direction, and orientation according to the algorithm's instructions.

6. Testing and Iteration: After the development and integration phases, rigorous testing in various environments and conditions is necessary. Metrics for evaluating the performance of the system include path planning efficiency, accuracy, adaptability, and computational resource usage. Regular monitoring and testing of the system can identify any performance issues or areas for improvement. The testing feedback will likely lead to several iterations of the path planning algorithm, motion system, and sensor fusion algorithms to optimize performance.
7. Dealing with Potential Inhibitors: Potential obstacles to successful implementation could include computational limitations, sensor inaccuracies, or rapidly changing environments. It's crucial to identify these issues early and address them to prevent performance problems or system failures.
8. Final Integration and Deployment: Once the system has been thoroughly tested and optimized, it can be integrated into the UAV for deployment. This involves not only the physical installation of hardware and the integration of software on the UAV but also ensuring that the UAV can handle the computational load of the system and that the system performs well under real-world conditions. The success of implementing this pattern relies heavily on the accuracy of the Lidar data, the efficiency of the path planning algorithm, and the effective integration of direction-awareness. Regular monitoring, testing, and system adjustments are necessary to ensure optimal performance in various dynamic environments.
9. Considerations for implementing in Multi-UAV Operations: Consider scenarios where multiple UAVs are needed, for instance, in extensive search and rescue operations, agricultural surveys over large fields, or large-scale environmental monitoring. In such cases, employing a fleet of UAVs can significantly improve operational efficiency and coverage. Each UAV in the fleet could use the online path planning system pattern to navigate its own path, responding to environmental changes individually. However, when functioning as a part of a multi-UAV system, additional parameters need to be considered for effective operation. These considerations include maintaining formation structure, cooperative sensing and data sharing, task allocation, and collision avoidance among the UAVs themselves. Here are the key considerations for multi-UAV operations:
    - Maintaining Formation Structure: Formations allow multiple UAVs to effectively cover a large area or focus on specific tasks simultaneously. The structure could range from simple geometrical arrangements (like line, grid, or column) to complex, adaptive formations based on environmental factors or mission goals. Maintaining formation requires sophisticated control algorithms that adjust the UAVs' positions in real-time. This not only includes adjustments due to movements but also adapting to changes in the formation structure. Additionally, depending on the mission requirements, the formation might need to remain rigid or could be allowed to deform to a certain extent. All these require complex inter-UAV communication, precise control mechanisms, and accurate real-time positioning information.
    - Cooperative Sensing and Data Sharing: When operating as a multi-UAV system, UAVs can benefit immensely from sharing sensor data and other mission-relevant information. For example, a UAV that has detected an obstacle can share this information with others, allowing them to adjust their paths even before they come close to the obstacle. Moreover, by sharing data from their individual perspectives, UAVs can create a more comprehensive and accurate 3D map of the environment, improving path planning, and obstacle detection. However, effective data sharing requires a robust and reliable communication system to handle potentially large volumes of data and ensure synchronization.
    - Task Allocation: In a multi-UAV system, tasks must be efficiently distributed among the UAVs to maximize overall performance. This involves complex decision-making processes, considering factors such as the capabilities of each UAV, the importance and requirements of each task, and the status of the mission. Some tasks might be best handled by a single UAV, while others could benefit from the coordinated efforts of multiple UAVs. Task allocation strategies need to be adaptive to respond to real-time changes in the mission or the environment. If a UAV becomes unavailable or if a new task emerges, the system needs to reallocate tasks on the fly.
    - Intra-collision Avoidance: While each UAV in a multi-UAV system uses its own collision avoidance system to avoid environmental obstacles, they also need to avoid collisions with each other. This requires an additional layer of collision avoidance that considers the positions, directions, and speeds of all UAVs in the system. Intra collision avoidance strategies can range from simple rules (like maintaining a minimum distance from each other) to sophisticated algorithms that predict potential collisions based on current trajectories and adjust the UAVs' paths to avoid them. It's also crucial to

coordinate these adjustments to prevent conflicting actions (like two UAVs trying to avoid each other by moving in the same direction).

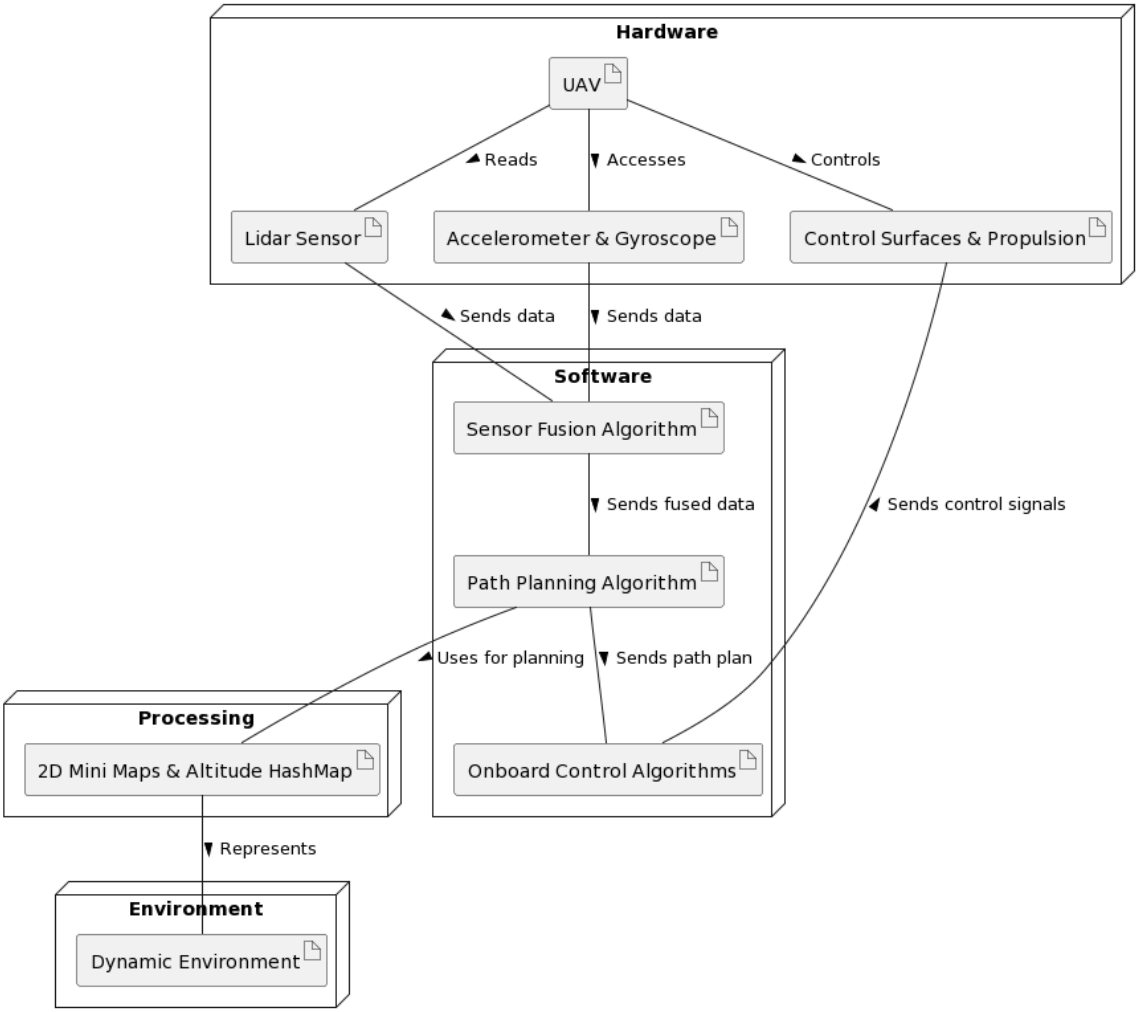


Figure 5. Online Path Planning Deployment Diagram

4.16. Known Uses

Instead of listing known uses of the pattern, we identify similar applications and potential users who could use the same approach. The pattern has a broad range of applications and is already being utilized in several domains. Its primary utility lies in enhancing the adaptability and efficiency of path planning and motion control for autonomous systems navigating in complex and dynamic environments. Below are some specific instances where this pattern is known to be employed.

1. Search and Rescue Operations: These missions often take place in dynamic and unpredictable environments. The drones involved in these operations need to be able to adapt their paths in real-time, avoiding obstacles and efficiently navigating through the disaster-stricken areas. The pattern has been used to enhance the efficiency and safety of these operations, allowing drones to adapt to real-time changes in the environment and to their own orientation and direction [53][54].
2. Surveillance and Monitoring: In surveillance and monitoring applications, drones are often required to navigate through complex environments like forests or urban landscapes. The pattern has been employed in these contexts to enable drones to avoid collisions and adapt their paths in real-time, making the surveillance process more efficient and accurate [55].
3. Autonomous Vehicles: Autonomous vehicles operate in highly dynamic environments, with other vehicles, pedestrians, and various obstacles constantly moving around them. The Online Path Planning System pattern can be applied to these vehicles to enhance their path planning and motion control capabilities, allowing them to navigate safely and efficiently through their environments [56].

4. Indoor Robots: Indoor robots, such as those used in warehouses or factories, often need to navigate through cluttered spaces with various obstacles. The pattern can be utilized in these situations to improve the robots' path planning and motion control abilities, enabling them to move more efficiently and adaptively in their environments [57]. In each of these uses, the pattern enhances the autonomous system's ability to navigate efficiently and safely through complex, dynamic environments, considering real-time changes in the environment and the system's orientation and direction. However, the implementation of the pattern may require additional computational resources due to the integration of more sophisticated algorithms and additional sensor data.

4.17. Consequences

The implementation of the pattern carries several significant consequences, both positive and negative. Below are the detailed explanations.

- Positive: Improved Efficiency, Accuracy, and Adaptability The primary consequence of implementing this pattern is a marked improvement in the efficiency, accuracy, and adaptability of the path planning process. By integrating a direction-aware A* algorithm, the pattern enables the UAV to consider its current orientation when determining the next move. This results in more efficient navigation, reducing unnecessary movements and conserving energy. In terms of accuracy, the integration of a Lidar-generated mini 2D map and altitude information stored in a hash map allows the UAV to have a more accurate understanding of its immediate environment. This improved environmental awareness enhances the accuracy of the path planning and reduces the likelihood of collisions with unforeseen obstacles. The pattern also boosts the adaptability of the UAV in dynamic environments. The UAV becomes capable of adjusting its path in real time in response to changes in the environment. This adaptability is crucial for UAV operations in unpredictable and changing environments like disaster-stricken areas or forests with moving obstacles.
- Negative: Increased Computational Resource Requirements On the downside, the implementation of this pattern may lead to an increase in the computational resources required. The addition of a modified A* algorithm, Lidar-generated mini 2D maps, and altitude data stored in a hash map means that the UAV's onboard computer needs to process more data in real time. This could necessitate more powerful computing hardware, more sophisticated software, or both. Depending on the specific UAV platform, this could result in increased costs, higher energy consumption, or additional weight from the onboard computer. These factors should be considered and appropriately managed during the implementation of the pattern.

4.18. Related Patterns

The pattern is related to several other patterns that deal with the navigation, control, and coordination of autonomous systems. Below is a definition of related patterns.

1. Traditional Path Planning Algorithms: These algorithms, such as the A* algorithm, Dijkstra's algorithm, or RRT, form the basis for many navigation systems. They allow for efficient path planning in known environments but often struggle in dynamic or complex environments. The pattern extends these traditional algorithms by incorporating real-time data and adjusting for changes in the UAV's orientation and direction [58].
2. SLAM (Simultaneous Localization and Mapping): SLAM is a computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. It's a fundamental problem solved in the field of robotics and is essential for autonomous robots like self-driving cars, drones, and rovers. SLAM could be complementary to the Online Path Planning System for UAV pattern, providing real-time updates about the environment which could be used for more informed path planning [59].
3. Sensor Fusion Algorithms: Sensor fusion is the use of sensory data from diverse sources in a complementary manner to provide robust and more accurate information than would be possible by using individual sensors alone. Sensor fusion techniques can be used to enhance the input data for the Online Path Planning System pattern, providing a more accurate picture of the dynamic environment [60].
4. Control Theory Applications: Control theory is an interdisciplinary branch of engineering and mathematics that deals with the behavior of dynamical systems with inputs. In the context of UAV, it's used to maintain stability and to control the flight. Control theory can be applied to improve the motion system's response to the path planning algorithm's output [61].
5. Swarm Intelligence: Swarm intelligence involves the collective behavior of decentralized, self-organized systems, natural or artificial. In the context of UAV, it's used to coordinate multiple drones to complete

tasks more efficiently. While this pattern focuses on individual drone navigation, it can be combined with swarm intelligence for tasks requiring coordination among multiple drones. These related patterns can be used in conjunction with online path planning system patterns to create an efficient, and adaptable navigation and control system for autonomous vehicles operating in dynamic environments [62].

5. CONCLUSION

Our research embarked on an ambitious journey to redefine the path planning systems of Unmanned Aerial Vehicles (UAVs) in dynamic environments. The cornerstone of this venture was the development of an innovative software pattern design, inspired by the paradigms of the Software Pattern Community (PLOP). This design serves as a blueprint for creating adaptable, efficient, and robust UAV path planning systems capable of navigating the complexities of unpredictable environments. The theoretical exploration presented in this study underscores the potential of integrating diverse software design patterns into UAV path planning. The proposed framework, a tapestry of behavioral, structural, and creational patterns, offers a versatile solution to the challenges of real-time adaptability and decision-making in UAV operations. It highlights the feasibility and benefits of employing a pattern-oriented approach to address the intricacies of UAV navigation, such as dynamic obstacle avoidance, energy optimization, and data management. While our study does not include empirical validation, its theoretical contributions lay the groundwork for future research. It opens possibilities for practical implementation and empirical testing of the proposed design in real-world scenarios. The next steps could involve adapting the design for specific UAV models and mission types, integrating advanced technologies like artificial intelligence and machine learning to further enhance its capabilities. In conclusion, this research not only advances the field of UAV path planning but also sets a precedent for the integration of software engineering principles into UAV system design. It offers a comprehensive framework that can be further explored and refined, potentially leading to groundbreaking advancements in UAV technology and its applications in diverse and dynamic environments.

REFERENCES

[1] T. Lai and F. Ramos, "LTR*: Rapid Replanning in Executing Consecutive Tasks with Lazy Experience Graph," *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8784-8790, 2022, https://doi.org/10.1109/IROS47612.2022.9982237.

[2] S. Lin, A. Liu, J. Wang, and X. Kong, "A Review of Path-Planning Approaches for Multiple Mobile Robots," *Machines*, vol. 10, no. 9, p. 773, 2022, https://doi.org/10.3390/machines10090773.

[3] C. Jose and K. S. Vijula Grace, "Optimization based routing model for the dynamic path planning of emergency vehicles," *Evol. Intel.*, vol. 15, pp. 1425–1439, 2022, https://doi.org/10.1007/s12065-020-00448-y.

[4] S. Abdallaoui, E.-H. Aglzim, A. Chaibet, and A. Kribèche, "Thorough Review Analysis of Safe Control of Autonomous Vehicles: Path Planning and Navigation Techniques," *Energies*, vol. 15, no. 4, p. 1358, 2022, https://doi.org/10.3390/en15041358.

[5] J. Dai, J. Qiu, H. Yu, C. Zhang, Z. Wu, and Q. Gao, "Robot Static Path Planning Method Based on Deterministic Annealing," *Machines*, vol. 10, no. 8, p. 600, 2022, https://doi.org/10.3390/machines10080600.

[6] S. Chakraborty, D. Elangovan, P. L. Govindarajan, M. F. ELnaggar, M. M. Alrashed, and S. Kamel, "A Comprehensive Review of Path Planning for Agricultural Ground Robots," *Sustainability*, vol. 14, no. 15, p. 9156, 2022, https://doi.org/10.3390/su14159156.

[7] M. Ramezani, H. Habibi, J. L. Sanchez-Lopez and H. Voos, "UAV Path Planning Employing MPC-Reinforcement Learning Method Considering Collision Avoidance," *2023 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 507-514, 2023, https://doi.org/10.1109/ICUAS57906.2023.10156232.

[8] A. Israr, Z. A. Ali, E. H. Alkhammash, and J. J. Jussila, "Optimization Methods Applied to Motion Planning of Unmanned Aerial Vehicles: A Review," *Drones*, vol. 6, no. 5, p. 126, 2022, https://doi.org/10.3390/drones6050126.

[9] S. Maadi, S. Stein, J. Hong, and R. Murray-Smith, "Real-Time Adaptive Traffic Signal Control in a Connected and Automated Vehicle Environment: Optimisation of Signal Planning with Reinforcement Learning under Vehicle Speed Guidance," *Sensors*, vol. 22, no. 19, p. 7501, 2022, https://doi.org/10.3390/s22197501.

[10] P. Yu, Y. Zhou, X. Sun, H. Sang, and S. Zhang, "Adaptive path following control for wave gliders in ocean currents and waves," *Ocean Eng.*, vol. 284, p. 115251, 2023, https://doi.org/10.1016/j.oceaneng.2023.115251.

[11] M. Kamezaki, A. Kobayashi, R. Kono, M. Hirayama and S. Sugano, "Dynamic Waypoint Navigation: Model-Based Adaptive Trajectory Planner for Human-Symbiotic Mobile Robots," in *IEEE Access*, vol. 10, pp. 81546-81555, 2022, https://doi.org/10.1109/ACCESS.2022.3194146.

[12] C. Zhang, P. Cheng, B. Du, B. Dong, and W. Zhang, "AUV path tracking with real-time obstacle avoidance via reinforcement learning under adaptive constraints," *Ocean Eng.*, vol. 256, p. 111453, 2022, https://doi.org/10.1016/j.oceaneng.2022.111453.

[13] P. Balasubramanian, "Automation in Data Science, Software, and Information Services," in Springer Handbook of Automation, S. Y. Nof, Ed. Cham, Switzerland: Springer, 2023, doi: https://doi.org/10.1007/978-3-030-96729-1_46.

[14] Z. Pei, L. Liu, C. Wang and J. Wang, "Requirements Engineering for Machine Learning: A Review and Reflection," *2022 IEEE 30th International Requirements Engineering Conference Workshops (REW)*, pp. 166-175, 2022, https://doi.org/10.1109/REW56159.2022.00039.

[15] Z. Chen, J. Liu, Y. Su, H. Zhang, X. Ling, Y. Yang, and M. R. Lyu, "Adaptive Performance Anomaly Detection for Online Service Systems via Pattern Sketching," in *Proc. of the 44th International Conference on Software Engineering (ICSE '22)*, pp. 61–72, 2022, https://doi.org/10.1145/3510003.3510085.
[16] A. R. Kunduru, "Cloud BPM Application (Appian) Robotic Process Automation Capabilities," *Asian J. Res. Comput. Sci.*, vol. 16, no. 3, pp. 267-280, 2023, https://doi.org/10.9734/ajrcos/2023/v16i3361.
[17] N. E. Akrami, M. Hanine, E. S. Flores, D. G. Aray and I. Ashraf, "Unleashing the Potential of Blockchain and Machine Learning: Insights and Emerging Trends From Bibliometric Analysis," in *IEEE Access*, vol. 11, pp. 78879-78903, 2023, https://doi.org/10.1109/ACCESS.2023.3298371.
[18] S. M. Srinivasan, S. Mahbub, R. S. Sangwan, Y. Badr, and P. Mukherjee, "Pattern Language for Designing Distributed AI Systems," In *INFORMS International Conference on Service Science*, pp. 467-477, 2022, https://doi.org/10.1007/978-3-031-15644-1_34.
[19] Shumba, A. T., Montanaro, T., Sergi, I., Fachechi, L., De Vittorio, M., & Patrono, L. , "Leveraging IOT-aware technologies and AI techniques for real-time critical healthcare applications," Sensors, vol. 22, no. 19, p. 7675, 2022, https://doi.org/10.3390/s22197675.
[20] F. Al-Doghman, N. Moustafa, I. Khalil, N. Sohrabi, Z. Tari and A. Y. Zomaya, "AI-Enabled Secure Microservices in Edge Computing: Opportunities and Challenges," in *IEEE Transactions on Services Computing*, vol. 16, no. 2, pp. 1485-1504, 2023, https://doi.org/10.1109/TSC.2022.3155447.
[21] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Sci. Robotics*, vol. 7, no. 66, eabm6074, 2022, https://doi.org/10.1126/scirobotics.abm6074.
[22] N. S. P. Peraka and K. P. Biligiri, "Pavement asset management systems and technologies: A review," *Automation in Construction*, vol. 119, p. 103336, 2020, https://doi.org/10.1016/j.autcon.2020.103336.
[23] K. Telli, O. Kraa, Y. Himeur, A. Ouamane, M. Boumehraz, S. Atalla, and W. Mansoor, "A Comprehensive Review of Recent Research Trends on Unmanned Aerial Vehicles (UAVs)," *Systems*, vol. 11, no. 8, p. 400, 2023, https://doi.org/10.3390/systems11080400.
[24] J. Li, G. Zhang, C. Jiang, and W. Zhang, "A survey of maritime unmanned search system: theory, applications and future directions," *Ocean Eng.*, vol. 285, 2023, p. 115359, https://doi.org/10.1016/j.oceaneng.2023.115359.
[25] H. S. Hewawasam, M. Y. Ibrahim and G. K. Appuhamillage, "Past, Present and Future of Path-Planning Algorithms for Mobile Robot Navigation in Dynamic Environments," in *IEEE Open Journal of the Industrial Electronics Society*, vol. 3, pp. 353-365, 2022, https://doi.org/10.1109/OJIES.2022.3179617.
[26] M. Kobayashi and N. Motoi, "Local Path Planning: Dynamic Window Approach With Virtual Manipulators Considering Dynamic Obstacles," in *IEEE Access*, vol. 10, pp. 17018-17029, 2022, https://doi.org/10.1109/ACCESS.2022.3150036.
[27] J. A. Abdulsaheb and D. J. Kadhim, "Classical and Heuristic Approaches for Mobile Robot Path Planning: A Survey," *Robotics*, vol. 12, no. 4, p. 93, 2023, https://doi.org/10.3390/robotics12040093.
[28] Z. Zhang, S. Wang, J. Chen and Y. Han, "A Bionic Dynamic Path Planning Algorithm of the Micro UAV Based on the Fusion of Deep Neural Network Optimization/Filtering and Hawk-Eye Vision," in *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 53, no. 6, pp. 3728-3740, 2023, https://doi.org/10.1109/TSMC.2023.3233965.
[29] M. Jones, S. Djahel, and K. Welsh, "Path-Planning for Unmanned Aerial Vehicles with Environment Complexity Considerations: A Survey," *ACM Comput. Surv.*, vol. 55, no. 11, p. 234, 2023, https://doi.org/10.1145/3570723.
[30] A. Khan, J. Zhang, S. Ahmad, S. Memon, H. A. Qureshi, and M. Ishfaq, "Dynamic Positioning and Energy-Efficient Path Planning for Disaster Scenarios in 5G-Assisted Multi-UAV Environments," *Electronics*, vol. 11, no. 14, p. 2197, 2022, https://doi.org/10.3390/electronics11142197.
[31] A. Serban and J. Visser, "Adapting Software Architectures to Machine Learning Challenges," *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 152-163, 2022, https://doi.org/10.1109/SANER53432.2022.00029.
[32] O. Chenaru, S. Mocanu, R. Dobrescu, and M. Nicolae, "Enhancing Antifragile Performance of Manufacturing Systems through Predictive Maintenance," *Applied Sciences*, vol. 12, no. 23, p. 11958, 2022, https://doi.org/10.3390/app122311958.
[33] G. Baptista and F. Abbruzzese. *Software Architecture with C# 10 and .NET 6: Develop Software Solutions Using Microservices, DevOps, EF Core, and Design Patterns for Azure*. Packt Publishing Ltd. 2022. https://books.google.co.id/books?hl=id&lr=&id=6tlhEAAAQBAJ.
[34] M. Pau, M. Mirz, J. Dinkelbach, P. Mckeever, F. Ponci and A. Monti, "A Service Oriented Architecture for the Digitalization and Automation of Distribution Grids," in *IEEE Access*, vol. 10, pp. 37050-37063, 2022, https://doi.org/10.1109/ACCESS.2022.3164393.
[35] V. Velepucha and P. Flores, "A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges," in *IEEE Access*, vol. 11, pp. 88339-88358, 2023, https://doi.org/10.1109/ACCESS.2023.3305687.
[36] A.-T. Shumba, T. Montanaro, I. Sergi, L. Fachechi, M. De Vittorio, and L. Patrono, "Leveraging IoT-Aware Technologies and AI Techniques for Real-Time Critical Healthcare Applications," *Sensors*, vol. 22, no. 19, p. 7675, 2022, https://doi.org/10.3390/s22197675.
[37] A. Puente-Castro, D. Rivero, A. Pazos, and E. Fernandez-Blanco, "A review of artificial intelligence applied to path planning in UAV swarms," *Neural Comput. Appl.*, vol. 34, no. 1, pp. 153-170, 2022, https://doi.org/10.1007/s00521-021-06569-4.
[38] N. Bashir, S. Boudjit, G. Dauphin, and S. Zeadally, "An obstacle avoidance approach for UAV path planning," *Simul. Model. Pract. Theory*, vol. 129, 2023, p. 102815, https://doi.org/10.1016/j.simpat.2023.102815.

[39] P. Galle, "Christopher Alexander's Battle for Beauty in a World Turning Ugly: The Inception of a Science of Architecture?," *She Ji: The Journal of Design, Economics, and Innovation*, vol 6, no. 3, pp. 345-375, 2020, https://doi.org/10.1016/j.sheji.2020.03.002.
[40] F. Mo, M. U. Querejeta, J. Hellewell, H. U. Rehman, M. I. Rezabal, J. C. Chaplin, D. Sanderson, and S. Ratchev, "PLC orchestration automation to enhance human–machine integration in adaptive manufacturing systems," *J. Manuf. Syst.*, vol. 71, pp. 172-187, 2023, https://doi.org/10.1016/j.jmsy.2023.07.015.
[41] J. Axelsson, "Systems-of-Systems Design Patterns: A Systematic Literature Review and Synthesis," *2022 17th Annual System of Systems Engineering Conference (SOSE)*, pp. 171-176, 2022, https://doi.org/10.1109/SOSE55472.2022.9812681.
[42] I. A. Buckley and E. B. Fernandez, "Dependability Patterns: A Survey," *Computers*, vol. 12, no. 10, p. 214, 2023, https://doi.org/10.3390/computers12100214.
[43] A. Alami and O. Krancher, "How Scrum adds value to achieving software quality?" *Empirical Software Engineering*, vol. 27, no. 7, pp. 165, 2022, https://doi.org/10.1007/s10664-022-10208-4.
[44] I. Arnold. *Enterprise Architecture Function: A Pattern Language for Planning, Design and Execution*. Springer Nature. 2022. https://books.google.co.id/books?hl=id&lr=&id=FJtXEAAAQBAJ.
[45] S. Waseeb and V. Vranić, "Toward Organizational Pattern Ontology," in *Proc. of the 27th European Conference on Pattern Languages of Programs (EuroPLop '22)*, p. 20, 2023, https://doi.org/10.1145/3551902.3551983.
[46] O. Zimmermann, M. Stocker, D. Lubke, U. Zdun, and C. Pautasso, *Patterns for API Design: Simplifying Integration with Loosely Coupled Message Exchanges*. Addison-Wesley Professional. 2022. https://books.google.co.id/books?hl=id&lr=&id=2tnPEAAAQBAJ.
[47] V. M. Romero and E. B. Fernandez, "Towards a Reference Architecture for Cargo Ports," *Future Internet*, vol. 15, no. 4, p. 139, 2023, https://doi.org/10.3390/fi15040139.
[48] B. Thapa, E. B. Fernandez, I. Cardei, and M. M. Larrondo-Petrie, "Abstract Entity Patterns for Sensors and Actuators," *Computers*, vol. 12, p. 93, 2023, https://doi.org/10.3390/computers12050093.
[49] L. Li, L. F. Herrera, L. Liang, and N. Law, "An outcome-oriented pattern-based model to support teaching as a design science," *Instructional Science*, pp. 1-32, 2022, https://doi.org/10.1007/s11251-021-09563-4.
[50] H. Washizaki, S. Ogata, A. Hazeyama, T. Okubo, E. B. Fernandez and N. Yoshioka, "Landscape of Architecture and Design Patterns for IoT Systems," in *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10091-10101, 2020, https://doi.org/10.1109/JIOT.2020.3003528.
[51] O. I. Abiodun *et al.*, "Comprehensive Review of Artificial Neural Network Applications to Pattern Recognition," in *IEEE Access*, vol. 7, pp. 158820-158846, 2019, https://doi.org/10.1109/ACCESS.2019.2945545.
[52] A. T. Azar, A. Koubaa, N. A. Mohamed, H. A. Ibrahim, Z. F. Ibrahim, M. Kazim, A. Ammar et al., "Drone deep reinforcement learning: A review," *Electronics*, vol. 10, no. 9, p. 999, 2021, https://doi.org/10.3390/electronics10090999.
[53] P. K. Muthusamy, M. Garratt, H. Pota and R. Muthusamy, "Real-Time Adaptive Intelligent Control System for Quadcopter Unmanned Aerial Vehicles With Payload Uncertainties," in *IEEE Transactions on Industrial Electronics*, vol. 69, no. 2, pp. 1641-1653, 2022, https://doi.org/10.1109/TIE.2021.3055170.
[54] V. Kangunde, R. S. Jamisola, and E. K. Theophilus, "A review on drones controlled in real-time," *Int. J. Dyn. Control*, vol. 9, no. 4, pp. 1832-1846, Dec. 2021, https://doi.org/10.1007/s40435-020-00737-5.
[55] S. Hu, W. Ni, X. Wang, A. Jamalipour and D. Ta, "Joint Optimization of Trajectory, Propulsion, and Thrust Powers for Covert UAV-on-UAV Video Tracking and Surveillance," in *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1959-1972, 2021, https://doi.org/10.1109/TIFS.2020.3047758.
[56] M. Aloqaily, R. Hussain, D. Khalaf, D. Slehat and A. Oracevic, "On the Role of Futuristic Technologies in Securing UAV-Supported Autonomous Vehicles," in *IEEE Consumer Electronics Magazine*, vol. 11, no. 6, pp. 93-105, 1 Nov. 2022, https://doi.org/10.1109/MCE.2022.3141065.
[57] L. Zhao, L. Yan, X. Hu, J. Yuan, and Z. Liu, "Efficient and High Path Quality Autonomous Exploration and Trajectory Planning of UAV in an Unknown Environment," *ISPRS Int. J. Geo-Inf.*, vol. 10, no. 10, p. 631, 2021, https://doi.org/10.3390/ijgi10100631.
[58] A. Ait Saadi, A. Soukane, Y. Meraihi, A. Benmessaoud Gabis, S. Mirjalili, and A. Ramdane-Cherif, "UAV Path Planning Using Optimization Approaches: A Survey," *Arch. Comput. Methods Eng.*, vol. 29, no. 6, pp. 4233-4284, 2022, https://doi.org/10.1007/s11831-022-09742-7.
[59] O. Saha, P. Dasgupta, and B. Woosley, "Real-time robot path planning from simple to complex obstacle patterns via transfer learning of options," *Autonomous Robots*, vol. 43, no. 8, pp. 2071-2093, 2019, https://doi.org/10.1007/s10514-019-09852-5.
[60] H. Yu, F. Zhang, P. Huang, C. Wang and L. Yuanhao, "Autonomous Obstacle Avoidance for UAV based on Fusion of Radar and Monocular Camera," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5954-5961, 2020, https://doi.org/10.1109/IROS45743.2020.9341432.
[61] X. Huang et al., "The Improved A* Obstacle Avoidance Algorithm for the Plant Protection UAV with Millimeter Wave Radar and Monocular Camera Data Fusion," *Remote Sensing*, vol. 13, no. 17, p. 3364, 2021, https://doi.org/10.3390/rs13173364.
[62] Y. Zhou, B. Rao and W. Wang, "UAV Swarm Intelligence: Recent Advances and Future Trends," in *IEEE Access*, vol. 8, pp. 183856-183878, 2020, https://doi.org/10.1109/ACCESS.2020.3028865.

AUTHOR BIOGRAPHY

Gregorius Airlangga received the B.S. degree in information system from the Yos Sudarso Higher School of Computer Science, Purwokerto, Indonesia, in 2014, and the M.Eng. degree in informatics from Atma Jaya Yogyakarta University, Yogyakarta, Indonesia, in 2016. He got Ph.D. degree with the Department of Electrical Engineering, National Chung Cheng University, Taiwan. He is also an Assistant Professor with the Department of Information System, Atma Jaya Catholic University of Indonesia, Jakarta, Indonesia. His research interests include artificial intelligence and software engineering include path planning, machine learning, natural language processing, deep learning, software requirements, software design pattern and software architecture.