Structuring World State Knowledge for Multi-UAV Automated Planning

Kai Sommer, Jane Jean Kiam

University of the Bundeswehr Munich
Department of Aerospace Engineering
Neubiberg, Germany
kai.sommer@unibw.de, jane.kiam@unibw.de

Abstract

World-state representation is a fundamental requirement for robotic agents operating under automated planning frameworks. However, existing planning frameworks typically assume a fully symbolic discrete world state, whereas other components of real robotic systems operate on sensor-derived continuous state representations. The gap between sensor-derived state representations and symbolic planning state remains largely unaddressed by existing knowledge bases, storing either only shallow symbolic facts or rich semantic representations. This paper presents a modular knowledge base approach that offers both: a structured continuous world state representation and a symbolic world model usable by automated planning. We introduce a fluent registry in which planning fluents are declaratively defined as conditions over a persistent knowledge base. The proposed knowledge base is integrated into the ROS 2-based AUSPEX multi-UAV framework and its applicability in a real-world Search and Rescue (SAR) scenario is demonstrated.

Code — https://github.com/UniBwM-IFS-AILab/AUSPEX-KNOW

Introduction

Automated planning enables robotic systems to perform high-level decision making that goes beyond navigation, allowing robots to reason at the task level to accomplish complex missions (Ghallab, Nau, and Traverso 2016). In its most general form, decision making can be formulated as finding a sequence of actions that leads from a given initial state to a desired goal state. This allows for strategic, task-based reasoning that is closer to human reasoning, enabling cognitive robots to be deployed more effectively in cooperative missions with human operators, as their actions become transparent and their plans interpretable and explainable.

Typically, automated AI planning techniques separate an abstract domain model from a specific problem instance containing initial state and goal conditions. The initial state is represented symbolically as a set of object instances and predicates describing object properties and relations. Problem models are commonly formalized using languages such as Planning Domain Definition Language (PDDL) (Ghallab et al. 1998).

Programmatic interfaces for PDDL modeling, such as Tarski (Francés and Ramirez 2018), have simplified the application of automated planning in robotic systems by eliminating the need to manually design PDDL files. Unlike Tarski, which focuses on programmatic PDDL modeling, the Unified Planning (UP) framework (Micheli et al. 2025) provides a planner-agnostic interface that unifies access to multiple planning engines through a common API, facilitating the integration of automated planning into robotic applications. In addition, a wrapper layer for integrating UP into ROS is available in the form of UP4ROS[1] and a dedicated bridge has been proposed for integrating UP into embedded systems to support plan execution and monitoring (Sadanandam et al. 2023). Together, these approaches lower the barrier for integrating automated planning into robotic systems by simplifying problem modeling, planner access, and plan execution. However, they assume a given initial state that represents the world in a discrete symbolic manner.

In contrast, robotic systems such as a multi-UAV system operating in real-world environments maintain continuous state representations to support perception, motion planning, and action execution (Siciliano, Khatib, and Kröger 2008). This creates a gap between the discrete symbolic world model required by automated planning and the continuous sensory and telemetry representations maintained within a robotic system. To generate valid plans, an automated planner needs a hybrid knowledge base where symbolic world states are grounded in their corresponding continuous state representations of the physical environment.

The main contribution of this work is the proposal of a knowledge base module for multi-UAV systems that maintains a global, persistent world state from continuously updated sensor-derived information, while enabling automated planning modules to derive a symbolic state representation for high-level mission planning. Our work focuses on:

- introducing a knowledge base module that structures both dynamic knowledge, such as sensor-derived information, and static knowledge, such as mission-specific information, thereby maintaining a centralized world state.
- proposing an architecture in which planning fluents are derived centrally from the knowledge base via a specified set of rules, rather than being asserted in a distributed and implicit manner across procedural code.

[1] https://github.com/aiplan4eu/UP4ROS

- proposing a modular system design that offers interfaces for integration into ROS 2-based multi-UAV systems.

The rest of this paper is structured as follows. We first present the motivation and review related work, followed by an overview of the system and its knowledge representations. Next, we describe the implementation. We then evaluate the module's performance and demonstrate its applicability in a real-world SAR mission through integration into the AUSPEX decision-making framework for UAVs.

Motivation

In a coordinated multi-UAV system (Chung et al. 2018), in addition to static information such as predefined platform capabilities and constraints, as well as mission-related information (e.g. goals and operational constraints), a substantial volume of continuously incoming data is essential for decision-making. This includes telemetry data describing the internal state of the UAVs, as well as perception-derived information (Alqudsi and Makaraci 2025).

However, these heterogeneous sources of information are not directly suitable when decision making is formulated as an automated planning problem (Ghallab, Nau, and Traverso 2016), which requires a planning instance defined by a set of typed objects, an initial state given by currently true predicates, and a set of goal conditions.

Deriving such a predicate-based world state from asynchronous data and semantic information sources poses several challenges:

- **Continuous versus discrete representations**: Robotic subsystems typically maintain real-valued, continuous state estimates, whereas automated planners require a discrete symbolic state.
- **Distributed and heterogeneous information sources**: Relevant knowledge is produced by multiple components, often in different formats and at different abstraction levels.
- **Conflicting observations**: Different UAVs or perception modules may produce inconsistent observations about the same entity, requiring the system to represent and manage multiple perspectives.
- **Dynamic validity of information**: Some knowledge, such as platform state or object observations, becomes outdated, whereas other knowledge, such as mission waypoints or platform capabilities, remains valid throughout the mission.
- **Symbol grounding problem** (Harnad 1990): Symbols used by the planner have no inherent meaning in the underlying physical world and require explicit grounding.

In addition to challenges inherent to the representations, *information acquisition and retrieval in coordinated multi-UAV systems are asynchronous*, due to varying observation update rates and communication constraints.

To address these challenges, two fundamental approaches can be distinguished (Levesque and Lakemeyer 2001): a *knowledge-based approach*, in which the world state is explicitly represented by a collection of symbolic structures maintained in a centralized knowledge base, and a *procedural approach*, in which distributed system components instead independently process information and assert world knowledge locally, from which the world state emerges implicitly.

A key advantage of the knowledge-based approach is that the structures describing the world state are centrally maintained and explicitly represented, enabling external inspection and interpretation. In contrast, procedural approaches lack a unified representation, as the world state is implicitly encoded and distributed across the codebase.

Building on the advantages of a knowledge-based approach, this work addresses the challenges outlined above. The proposed knowledge module provides a centralized representation for integrating heterogeneous information sources, supports handling of conflicting observations and dynamic validity, and defines an interpretable mapping from continuous state estimates to discrete symbolic predicates for planning.

Related Work

Automated Planning Frameworks for Robotic Systems

A number of approaches have successfully integrated automated planning into robotic systems, including frameworks such as ROSPlan (Cashmore et al. 2015) and PlanSys2 (Martín et al. 2021).

ROSPlan is a framework for integrating automated planning into ROS-based systems by providing interfaces for symbolic planning, execution, and monitoring components. A centralized knowledge base stores and manages plain symbolic planning facts, but the grounding of these facts is delegated to components that assert the facts via the knowledge base API rather than being represented as explicit grounding rules within the knowledge base itself. Furthermore, ROSPlan is coupled to ROS 1, which has been superseded by ROS 2 as the current standard for robotic software development.

Inspired by ROSPlan, PlanSys2 (Martín et al. 2021) provides an integration of automated planning into modern ROS 2-based systems. In PlanSys2, symbolic planning fluents are managed by the "Problem Expert" module and updated procedurally by user-defined components.

In summary, the knowledge bases in these frameworks store only symbolic facts that are asserted by external components, rather than maintaining a persistent sensor-derived state from which planning fluents are derived through explicit abstraction rules.

Knowledge Bases and Representations in Robotics

In parallel to advances in automated planning frameworks, there has been extensive research on how robotic systems can represent, store, and reason about knowledge of the world. Knowledge bases populated with semantic representations aim to capture objects, relations, and contextual information in a structured and reusable form, enabling higher-level reasoning beyond raw sensor data. Early robotic

systems, such as Shakey (Nilsson 1984), relied on an internal representation of the world as the basis for planning and task execution. This world model consisted of a set of symbolic facts that were explicitly asserted by system components.

More recent approaches have focused on developing expressive knowledge bases for robotic systems, most prominently ontology-based frameworks such as KnowRob (Beetz et al. 2018) and ORO (Lemaignan et al. 2010). These frameworks provide rich semantic models of robotic environments and support logical reasoning over world knowledge. However, they are often developed and evaluated in the context of service and manipulation robotics, and their underlying assumptions do not readily generalize to other domains such as UAV missions. Other ontology-based approaches rely on large-scale knowledge bases such as OpenCyc (Lenat 1995) or on hand-crafted, robot-specific knowledge graphs like RoboBrain (Saxena et al. 2014).

While these knowledge bases have convincingly demonstrated the value of explicit semantic world models and reasoning capabilities, they typically lack a systematic connection to automated planning frameworks. In particular, they do not specify how symbolic planning fluents required by planners are derived from knowledge base internal world-state representations. Other works, such as PlanOWL (Adamik and Forte 2025), generate PDDL domains from OWL ontologies, while our work assumes a given domain and focuses on grounding symbolic planning problem instances (i.e. current state) from sensor-derived world knowledge.

Bridging Sensor-Derived and Symbolic Representations

Several approaches have addressed the challenge of connecting symbolic planning with continuous representations of the world. Semantic maps enrich metric or topological maps with semantic information to support task planning. Galindo et al. show how such representations can be exploited by a planner to reason about spatial structure and object semantics, yet do not provide an explicit, persistent knowledge base (Galindo et al. 2008).

Another related line of work introduces *semantic attachments* to connect symbolic planners with external computations (Dornhege et al. 2012). Semantic attachments allow planners to invoke continuous or geometric checks during planning, but require these evaluations to be embedded directly within the planning process.

In contrast, our approach employs a decoupled knowledge base that provides a symbolic world-state independently of the planning process.

A Knowledge Module For Multi-UAV Automated Planning

The presented approach is two-fold:

- First, it includes a knowledge base that maintains a globally consistent world state that integrates real-valued and categorical information across all UAVs and the mission environment.
- Second, it provides a mechanism to derive a symbolic representation from the structured, continuous world state maintained by the knowledge base to enable automated planning.

System Overview

To this end, the approach introduces a *Knowledge Module*, which includes two components:

- A *Knowledge Base*, which persistently maintains the current real-valued and categorical structured global world state together with static mission knowledge in a frame-based representation.
- A *Fluent Registry*, which allows a planning module to register fluents as declarative conditions over the structured world representations and grounds them based on the current state of the knowledge base, thereby providing the symbolic representation required by the planning module.

A typical multi-UAV system operates across multiple levels of abstraction. The lowest level of abstraction corresponds to the raw sensor data layer, which comprises continuous-valued measurements acquired directly from onboard sensors (e.g. an IMU, position estimates from a GNSS module, and image streams from payload cameras) without semantic interpretation.

Above this layer, perception and state estimation modules process raw sensor data through filtering and sensor fusion (e.g., using Kalman filters) or through higher-level interpretation, such as object detection algorithms (Varghese and M. 2024).

The proposed knowledge module is positioned at this abstraction level and structures the received information into a globally consistent world state. In addition to the continuously evolving state vectors derived from sensor measurements, the system maintains static mission knowledge describing objects that remain constant throughout mission execution. This includes platform model information for UAVs and sensors, such as technical specifications; and it encompasses environmental data, such as points of interest or search areas, which can be derived from a map or defined by a human operator.

Knowledge Base Knowledge in the knowledge base is represented in a frame-based manner (Minsky 1974). In this object-oriented representation, a *frame* is a structure that describes an entity through *slots*, where each slot can hold a *value* and optionally a default value. Additionally, frames can include procedural attachments that associate callbacks with specific slots, which are automatically triggered when the slot value changes.

While a frame defines the generic schema, a *frame instance* represents a concrete realization of that schema by filling the slots with specific values. Each instance is distinguished from other instances by a unique *instance ID*. In some cases, this unique identifier of an instance is known *a priori*, for example for UAV platforms or predefined waypoints. In other cases, such as for objects detected on-the-fly, a unique identifier can be generated in the perception module by tracking algorithms (Wojke, Bewley, and Paulus 2017).

General Structure

```
Frame
    |-- Instance ID
        |-- Subframe
        |--- [Mode]
        |--- [TTL]
            |-- Variant Key [if Mode=multiple]
                |-- Slot : Value
                |--- [Default]
                |--- [Timestamp]
```

Figure 1: General structure of the frame-based knowledge representation.

Example

```
Object
    |-- object23
        |-- observation
        |--- [Mode=multiple]
        |--- [TTL=30]
            |-- milan1
                |-- detection_class : "bicycle"
                |--- [Default="object"]
                |--- [Timestamp=123456]
                |-- confidence : 0.56
                |--- [Default=None]
                |--- [Timestamp=123456]
            |-- hawk2
                |-- detection_class : "person"
                |--- [Default="object"]
                |--- [Timestamp=234567]
                |-- confidence : 0.79
                |--- [Default=None]
                |--- [Timestamp=234567]
```

Figure 2: Example of a frame instance with multiple variants for an observed object.

Figure 1 illustrates the general structure of the frame-based knowledge representation. In a UAV mission domain, typical frames are UAV platforms, waypoints, and (observed) objects.

Frames contain *subframes*, which group semantically related slots into a named structure within a frame. This allows, for example, the slots representing the platform state of a UAV to be separated from those describing its platform capabilities, even though both belong to the same frame.

Subframes can be conceptually categorized into *static* and *dynamic* subframes. Slots of static subframes do not originate from measurements and are initialized once during mission setup. In contrast, dynamic subframes represent attributes of entities that evolve over time and are continuously updated during mission execution. Their values are derived from perception or state estimation modules and may be continuous (e.g., position estimates) or categorical (e.g., detection classes such as `"person"`).

In addition to their values, slots are associated with metadata that govern their validity and interpretation. In particular, slots of dynamic subframes are annotated with a time-to-live (TTL) value (see Figure 1), which defines their temporal validity: if the TTL expires without an update, the corresponding slot value is automatically invalidated. Static subframes do not carry a TTL, as their values are assumed to remain valid throughout the mission. Further metadata include a timestamp of the last update and an optional default value. Despite these differences, static and dynamic subframes are handled uniformly through the same internal representation and interfaces.

Furthermore, we distinguish between subframes with a *single* variant and those with *multiple* variants. A slot with a *single* variant holds exactly one value at a time, which is overwritten upon each update—for example, the current position of a UAV or the coordinates of a waypoint. In contrast, a slot with *multiple* variants can store several values simultaneously, each associated with a *variant key* that distinguishes it from other variants. This enables, for example, different UAVs to maintain independent observations of the same detected object, as illustrated in Figure 2. Multiple variant values can optionally be merged into a single fused value via a procedural attachment.

Fluent Registry The purpose of the *Fluent Registry* is to derive a symbolic representation of the world state from the object-oriented representations stored in the knowledge base.

Let $\mathcal{O}_t$ denote a finite set of object instances at time step t and $\mathcal{P}$ denote the set of predicates defined in the planning domain. A symbolic world state S_t at time step t is defined as a finite set of grounded predicates: $S_t \subseteq \{\, p(o_1, \ldots, o_k) \mid p \in \mathcal{P},\ o_i \in \mathcal{O}_t \,\}$, where $p(o_1, \ldots, o_k)$ denotes a predicate that evaluates to true at time step t. The value k is called the arity of a predicate.

Examples of unary and binary predicates in the UAV mission domain would be: `isPerson(obj23)`, `isAt(uav1, waypoint45)`. In more recent frameworks, predicates are often referred to as fluents, emphasizing that their values may change over time and, in some formalisms, may take non-Boolean values. In our architecture, however, we restrict ourselves to classical planning (Ghallab, Nau, and Traverso 2016) and therefore consider only Boolean-valued fluents.

The Fluent Registry allows a planning module to register fluents as conditions over slot values of frame instances.

A fluent is registered in the Fluent Registry as a tuple:

$$f := \langle name, frames, slots, condition, aggregation \rangle$$

where $name$ is a unique fluent identifier, and $frames$ and $slots$ denote ordered lists of frames and corresponding slots over which the fluent is evaluated by $condition$. The aggregation parameter specifies how multiple slot values are combined during evaluation, supporting the operators *any* (true if at least one value satisfies the condition) and *all* (true only if all values satisfy the condition), with *any* used as the default. As a simplified example, the unary fluent `isLanded(uav)` can be defined as:

$$\langle isLanded, uav, platform\_status.altitude, v < 0.1 \rangle$$

where v denotes the value of the referenced slot. To evaluate a fluent for a given frame instance, the procedure shown in Algorithm 1 is applied. It first retrieves all variants (see Line 2) for the referenced subframe and iterates over them, reading the corresponding slot values (see Line 4) and evaluating the fluent condition (see Line 6) for each variant. The results are then aggregated according to the specified aggregation function (see Line 9). If the aggregated result evaluates to true, the grounded fluent for the given frame instance is added to the knowledge base (see Line 11); otherwise, it is removed (see Line 13).

Algorithm 1: Evaluate a unary fluent for a given instance and store the result in the knowledge base.

Require: Unary fluent f, Knowledge Base KB, Instance ID id

1: $R \leftarrow [\,]$
2: Variants $K \leftarrow KB.\text{GETVARIANTS}(f.frame, id, f.slots)$
3: **for** each $key \in K$ **do**
4: $\quad V \leftarrow KB.\text{READSLOTS}(f.frame, id, f.slots, key)$
5: $\quad (v_1, \ldots, v_n) \leftarrow V$
6: $\quad result \leftarrow f.condition(v_1, \ldots, v_n)$
7: $\quad$ append $result$ to R
8: **end for**
9: $is\_true \leftarrow \text{AGGREGATE}(R, f.aggregation)$
10: **if** $is\_true =$ **true then**
11: $\quad KB.\text{SETFLUENT}(f.name, id)$
12: **else**
13: $\quad KB.\text{DELFLUENT}(f.name, id)$
14: **end if**

The Fluent Registry stores only *lifted* fluents, that is, fluents defined at the schema level as specified by the tuple above and not evaluated for specific frame instances. The fluents are grounded on demand and the Fluent Registry performs grounding by identifying all frame instances to which the fluent applies, see Algorithm 2. Specifically, the registry queries the knowledge base for all frame instance IDs of the given frame (see Line 1). For each such instance, the fluent's condition is evaluated using Algorithm 1 (see Line 3). As a result, the knowledge base maintains the set of instance IDs for which the fluent holds.

Algorithm 2: Compute groundings of a unary fluent.

Require: Unary fluent f, Knowledge Base KB, Fluent Registry FR

1: Instance IDs $I \leftarrow KB.\text{GETINSTANCES}(f.frame)$
2: **for** each $id \in I$ **do**
3: $\quad FR.\text{EVALUATE}(f.name, id)$
4: **end for**

The approach naturally extends to binary fluents (see Algorithm 3). However, the computational complexity increases to $\mathcal{O}(n^2)$, as all pairs of instances from the referenced frames must be considered.

In addition to evaluating and grounding registered fluents, the Fluent Registry provides functionality that allows the planning module to observe fluents. To this end, the planning module is notified whenever the truth value of a registered fluent changes or when a new grounding of that fluent becomes available. Whenever a slot value relevant to a fluent changes, the corresponding fluent instance is re-evaluated. This observation mechanism can be used to trigger replanning strategies in response to changes in the world state.

Algorithm 3: Compute groundings of a binary fluent.

Require: Binary fluent f, Knowledge Base KB, Fluent Registry FR

1: Instance IDs $I_1 \leftarrow KB.\text{GETINSTANCES}(f.frame[0])$
2: Instance IDs $I_2 \leftarrow KB.\text{GETINSTANCES}(f.frame[1])$
3: **for** each $i \in I_1$ **do**
4: $\quad$ **for** each $j \in I_2$ **do**
5: $\quad\quad FR.\text{EVALUATE}(f.name, i, j)$
6: $\quad$ **end for**
7: **end for**

Implementation

The implementation of the proposed knowledge module includes the following components, see Figure 3:

- A *Knowledge Collector* that collects relevant ROS 2 messages from perception and state estimation modules.
- A *Knowledge Server* that provides ROS 2 services for insertions, updates, queries and deletion of knowledge base items.
- A *Fluent Registry* for the proposed fluent registration, evaluation and observation mechanism.
- A *Valkey Client*, a client for the database.
- A *Valkey Server*, as database backbone.

The complete module is ROS 2-based and it is intended to be integrated into a ROS 2-based planning and acting framework that comprises at least the following components:

- A centralized *Planning Module*, which utilizes an automated planner to generate plans based on the initial state retrieved from the knowledge module.
- An *Execution Module*, which is responsible for executing the generated plans.
- and n *Offboard Control Modules* and m *Perception Modules* (where n denotes the number of UAVs and m the number of sensors connected to the system), which publish their state estimates and their observations as ROS 2 messages.

It is assumed that the knowledge and planning modules run on a centralized ground station, while the offboard control modules run on the companion computers of the UAVs, and the perception and execution modules may run on either the ground station or the UAVs, all interconnected within a common ROS 2 network.

Database A database is used for storing the world state using the representation described above in Section Knowledge Base. This implementation leverages Valkey[2], which

[2]Valkey is an open-source community fork of Redis: https://github.com/valkey-io

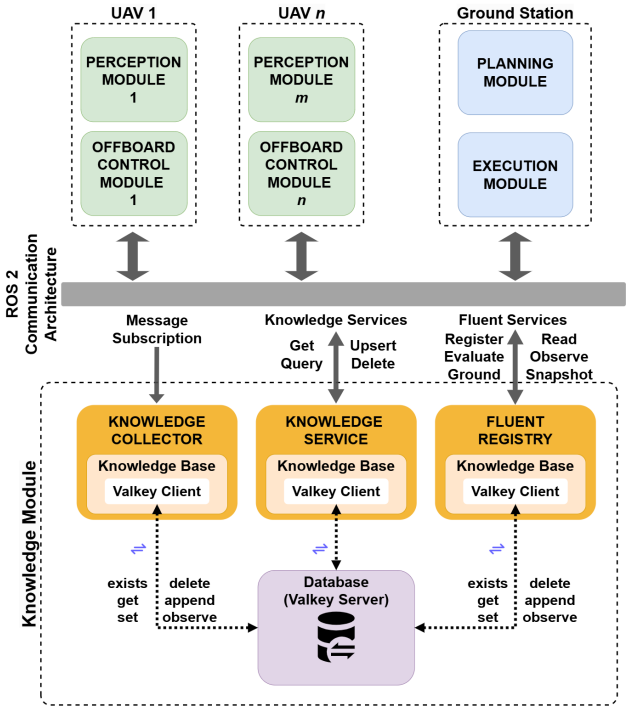


Figure 3: Overview of software components and interfaces of the proposed knowledge module.

is primarily an in-memory key-value store. Valkey can be extended through plug-ins to support document-like JSON data, offering a convenient way to store frame-like entities with slot values.

In this implementation, keys are composed by using the frame as a prefix, the instance ID as a unique identifier, and the subframe as a namespace for the slot values. If a slot has multiple variants, an additional suffix is appended to distinguish between them. The value associated with a key can be either a primitive value or a JSON document. Table 1 summarizes how the frame representation is mapped to the database layer. The primary benefit of a Valkey-based core is its in-memory data storage, which enables low-latency read and write access, making the system suitable for time-sensitive real-world UAV missions. Furthermore, Valkey's NoSQL design and support for different data structures provide flexibility in schema evolution, which is beneficial for evolving knowledge representations.

Any component on the same network that needs to access the database for basic atomic operations can instantiate a Valkey client component.

Knowledge Base Interfaces Two components are responsible for updating the maintained world state.

The first component, the *Knowledge Collector* node, is responsible for updating entities whose slots represent information derived from ROS 2 messages. To this end, it subscribes to a configurable set of ROS 2 topics. Incoming ROS 2 messages are interpreted as subframes, whose fields are mapped to slots of the corresponding frame instance. Upon reception, the respective slots of an existing frame instance are updated, or a new frame instance is created if none exists. The system maintains a configurable mapping that associates ROS 2 message types with subframes and specifies which message field serves as the instance ID. Table 2 illustrates how the different representation levels in the proposed architecture relate to each other. Since no schema is enforced at the database level, ROS 2 message definitions effectively serve as the schema for subframes.

The *Knowledge Server* is a node that provides ROS 2 services for reading, setting or updating, and deleting slot values, as well as creating and deleting frame instances or retrieving a list of instances for a given frame. These services are primarily used to load static knowledge into the knowledge base. In addition, the execution component can use the Knowledge Server to resolve symbolic references into numerical values, such as retrieving the GNSS coordinates of a waypoint for action execution.

Fluent Registry The *Fluent Registry* node provides the following ROS 2 services for the planning module:

- Register/Unregister: Upon registration of a new fluent, a tuple $\langle name, frames, slots, condition, aggregation \rangle$ is inserted into or removed from an internally maintained dictionary indexed by $name$.
- Evaluate: Evaluates the fluent condition for a registered fluent with respect to a given instance ID (see Algorithm 1).
- Ground: Computes all groundings of a fluent and evaluates the corresponding conditions (see Algorithm 2).
- Read: A read request is parameterized by an instance ID. It returns true if the instance ID is contained in the set of grounded fluents, since only fluents that evaluate to true are stored. A variant of this operation re-evaluates the condition for the given instance and returns the resulting truth value.
- Observe: Upon subscribing to a fluent, a `FluentChange` message containing the instance ID is published whenever the set of grounded fluents changes. At the database level, this is implemented using Valkey's publish/subscribe mechanism to detect changes in slot values relevant to the observed fluent, followed by re-evaluation of the condition.
- Snapshot: Computes the current symbolic world state by grounding all registered fluents and returning, for each fluent, the set of instance IDs (or tuples thereof) for which the fluent evaluates to true. The resulting snapshot corresponds to the set of grounded predicates defining S_t. Additionally, it includes, for each frame, the set of corresponding instance IDs.

Evaluation

Validation on Hardware

The experiments were conducted on a laptop, which can typically act as a ground station in a multi-UAV system (equipped with 32 GB RAM and an Intel Core i9-11900H CPU).

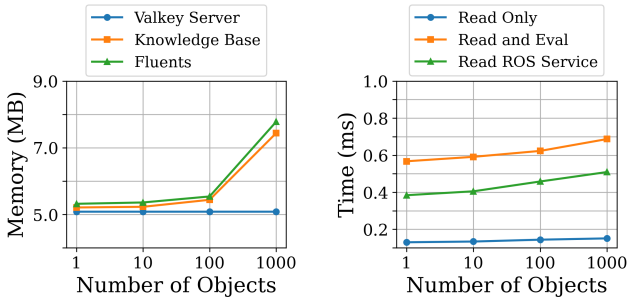
Table 1: Mapping of frame-based representation to Valkey key structure

Frame Representation	**Valkey Key Structure**
Frame	`uav`
Instance ID	`uav:hawk2`
Subframe	`uav:hawk2:platform_state`
Slot	`uav:hawk2:platform_state:gps_position`
Variant (optional)	`uav:hawk2:platform_state:gps_position:sensor1`
Full key schema	`frame:instance:subframe:slot[:variant]`
Example key	`uav:hawk2:platform_state:gps_position:sensor1`
Value	Primitive value or JSON document

Table 2: Representation levels in the proposed system

ROS 2 Level	**Knowledge Base Level**	**Symbolic Level**
–	Frame	Type
–	Frame instance	Object
Message type	Subframe	–
Message instance	Subframe instance	–
Instance ID	Instance ID	Object name
Message field	Slot	–
Message field value	Slot value	–
–	Condition over slot values	Fluent

– indicates no direct correspondence.



(a) Memory Usage (b) Read Performance

Figure 4: Memory usage and read performance of the knowledge module.

The experiments included three types of frames: a `uav` frame with a dynamic single-variant subframe containing platform state information (telemetry) and a static single-variant subframe containing platform capabilities; an `object` frame with a multiple-variant subframe containing observations that may originate from different UAVs; and a static `waypoint` frame. ROS 2 messages corresponding to the dynamic subframes were published at 10 Hz. The number of frame instances was increased in powers of ten.

Figure 4a shows that the memory footprint of the Valkey storage remains low even with 1000 objects and the storage of true fluents adds only a small overhead, while Figure 4b shows that basic read operations as well as ROS 2 service read calls and read combined with fluent evaluation remain in the sub-millisecond range.

These results suggest that the knowledge module is lightweight enough to be deployed not only on a ground station but also on the onboard computer of a UAV, indicating that the approach is feasible for execution on embedded hardware within the communication constraints in real multi-UAV systems.

Example of Real-World Application

For validation in a real-world mission, the presented knowledge module was integrated into AUSPEX (Döschl, Sommer, and Kiam 2025), a ROS 2-based decision-making framework for UAVs. Through its open-source availability[3] and modular architecture, AUSPEX provides a ready-to-use infrastructure for deploying the proposed knowledge module in a real-world UAV mission. Notably, AUSPEX includes a planning module that integrates the UP library, thereby enabling automated planners to interface with the symbolic representation provided by the knowledge module.

In addition, AUSPEX comprises a plan execution module, a perception module that analyzes UAV camera streams using an object detector (Varghese and M. 2024), and an offboard control module that executes actions and publishes platform state messages. All AUSPEX modules, together with the knowledge module, were deployed on a centralized ground station, except for the offboard control modules, which were deployed on the UAV companion computers. For this mission, three UAVs were employed: one Holybro X500v2 equipped with an optical onboard camera for object detection, a second Holybro X500v2 equipped with a first-aid delivery mechanism, and a Multikopter MK-U20 with thermal and optical cameras for object detection. Both Holybro UAVs were equipped with a Raspberry Pi 5 as a companion computer for the offboard control module, and the MK-U20 with an NVIDIA Jetson Xavier NX. The three UAVs and the ground station were interconnected via a shared VPN network over a cellular network.

In the example SAR scenario, the objective was to locate a missing person in two search areas. The mission was considered complete when all UAVs had landed and either both search areas had been fully searched or a person had been detected. In case a person was detected, first aid had to be delivered. Two of the UAVs were equipped with a camera, which was a precondition for the search action, and one UAV was equipped with a first aid kit, which was a precondition for the delivery action. This planning problem was

[3]https://github.com/UniBwM-IFS-AILab/AUSPEX

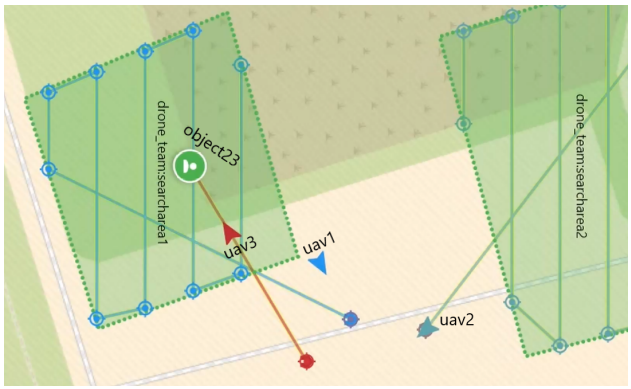


Figure 5: Mission GUI showing the process during execution of the search mission, including UAV trajectories, search areas, and the position of detected objects.

modeled using the UP library by registering fluents such as `isLanded(uav)`, `hasOpticalCamera(uav)`, `hasFirstAid(uav)`, and `isPerson(object)` at the Fluent Registry.

The problem instance was generated using the UP library and initialized with the current world state obtained from the snapshot of the knowledge module. Planning was performed using the Fast Downward planner (Helmert 2006), integrated via UP.

The search areas, UAV specifications, and equipped payloads were inserted into the knowledge module during mission initialization. The Knowledge Collector was configured to subscribe to `platform_state` messages of the UAVs and `object_observation` messages of the perception modules. Replanning for the delivery task was triggered by observing the fluent `isPerson`, as soon as it became true for any detected object.

The execution component further utilizes the knowledge module to execute the generated plans by resolving symbolic references into concrete values required for action execution — for example, resolving the symbolic argument of an action such as `flyTo(object23)` to concrete GNSS coordinates. Figure 5 shows the mission GUI, which queries the knowledge module to visualize the progress of the mission and represent symbolic entities in their corresponding spatial context.

During execution, the knowledge base had an average input rate of 13 kB/s. The memory footprint was approximately 6 MB. These values remain well below the capacity limits of the available VPN network bandwidth. A more systematic quantitative evaluation of read/write latencies and symbolic world-state generation is left for future work.

Conclusion

In this work, we present an approach for structuring information arising in a multi-UAV system into knowledge that can be utilized for automated planning.

In contrast to existing approaches that store solely shallow symbolic facts, our approach maintains an underlying layer of detailed real-valued and categorical information. As a result, the derivation of fluents from the knowledge base is not distributed across perception or state estimation modules. This makes the fluent grounding process more inspectable and debuggable.

Beyond that, symbolic knowledge can be resolved to more fine-grained information, which can be utilized by execution or visualization components.

The proposed knowledge module is centralized and forms a collective world state from different observation perspectives. This allows for a consistent symbolic world state for the planning module. Its Valkey-based implementation provides fast in-memory read/write access and a low memory footprint through its lightweight frame-based knowledge representation, making it suitable for real-world missions. In addition, the client–server architecture supports modular system design and remote access to a centralized knowledge base.

Moreover, the fluent observation mechanism that enables observers to track changes in the symbolic world state makes this approach interesting for replanning or plan repair strategies.

Finally, the source code of the presented knowledge module is publicly available under an open-source license. Its modular ROS 2-based design facilitates integration into ROS 2-based frameworks, as demonstrated with AUSPEX (Döschl, Sommer, and Kiam 2025).

Future Work

Future work will focus on the following directions:

- Investigating distributed synchronization strategies to maintain a consistent world state across multiple instances of the knowledge module — deployed both on the ground station and on individual UAVs — under intermittent connectivity.
- Exploring probabilistic approaches to model, represent, and resolve conflicting knowledge, building on the frame-based representation with variants introduced in this work.
- Learning fluent conditions from data, replacing manually handcrafted rules.
- Introducing an ontology on top of the frame-based representation to support more expressive typing and semantic reasoning within the knowledge base.

Acknowledgments

This research was supported by the Bundesministerium für Forschung, Technologie und Raumfahrt (BMFTR) through Project MENTHON and by the DFG-funded Project CHIP-GT. Partial hardware resources were provided by dtec.bw – Digitalization and Technology Research Center of the Bundeswehr MissionLab, which we gratefully acknowledge. dtec.bw is funded by the European Union – NextGenerationEU. The scenario modeling benefited from valuable input provided by the Mountain Rescuers of Penzberg (Bergwacht Penzberg).

References

Adamik, M.; and Forte, P. 2025. PlanOwl: Automated PDDL Files Generation from OWL Ontologies and Visual Language Models. In *Proceedings of the AAAI Symposium Series*, volume 7, 634–643.

Alqudsi, Y.; and Makaraci, M. 2025. UAV swarms: research, challenges, and future directions. *Journal of Engineering and Applied Science*, 72(1): 12.

Beetz, M.; Beßler, D.; Haidu, A.; Pomarlan, M.; Bozcuoğlu, A. K.; and Bartels, G. 2018. Know rob 2.0—a 2nd generation knowledge processing framework for cognition-enabled robotic agents. In *2018 IEEE international conference on robotics and automation (ICRA)*, 512–519. IEEE.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtos, N.; and Carreras, M. 2015. Rosplan: Planning in the robot operating system. In *Proceedings of the international conference on automated planning and scheduling*, volume 25, 333–341.

Chung, S.-J.; Paranjape, A. A.; Dames, P.; Shen, S.; and Kumar, V. 2018. A survey on aerial swarm robotics. *IEEE Transactions on robotics*, 34(4): 837–855.

Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2012. Semantic attachments for domain-independent planning systems. In *Towards Service Robots for Everyday Environments: Recent Advances in Designing Service Robots for Complex Tasks in Everyday Environments*, 99–115. Springer.

Döschl, B.; Sommer, K.; and Kiam, J. J. 2025. AUSPEX: An integrated open-source decision-making framework for UAVs in rescue missions. *Frontiers in Robotics and AI*, 12: 1583479.

Francés, G.; and Ramirez, M. 2018. Tarski: An AI Planning Modeling Framework. https://github.com/aig-upf/tarski.

Galindo, C.; Fernández-Madrigal, J.-A.; González, J.; and Saffiotti, A. 2008. Robot task planning using semantic maps. *Robotics and autonomous systems*, 56(11): 955–966.

Ghallab, M.; Knoblock, C.; Wilkins, D.; Barrett, A.; Christianson, D.; Friedman, M.; Kwok, C.; Golden, K.; Penberthy, S.; Smith, D.; Sun, Y.; and Weld, D. 1998. PDDL - The Planning Domain Definition Language.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated planning and acting*. Cambridge University Press.

Harnad, S. 1990. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1-3): 335–346.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246.

Lemaignan, S.; Ros, R.; Mösenlechner, L.; Alami, R.; and Beetz, M. 2010. ORO, a knowledge management platform for cognitive architectures in robotics. In *2010 IEEE/RSJ International conference on intelligent robots and systems*, 3548–3553. IEEE.

Lenat, D. B. 1995. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11): 33–38.

Levesque, H. J.; and Lakemeyer, G. 2001. *The logic of knowledge bases*. Mit Press.

Martín, F.; Clavero, J. G.; Matellán, V.; and Rodríguez, F. J. 2021. Plansys2: A planning system framework for ros2. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 9742–9749. IEEE.

Micheli, A.; Bit-Monnot, A.; Röger, G.; Scala, E.; Valentini, A.; Framba, L.; Rovetta, A.; Trapasso, A.; Bonassi, L.; Gerevini, A. E.; Iocchi, L.; Ingrand, F.; Köckemann, U.; Patrizi, F.; Saetti, A.; Serina, I.; and Stock, S. 2025. Unified Planning: Modeling, Manipulating and Solving AI Planning Problems in Python. *SoftwareX*, 29: 102012.

Minsky, M. 1974. *A framework for representing knowledge*. USA: Massachusetts Institute of Technology.

Nilsson, N. J. 1984. Shakey the robot.

Sadanandam, S. H. S. S.; Stock, S.; Sung, A.; Ingrand, F.; Lima, O.; Vinci, M.; and Hertzberg, J. 2023. A Closed-Loop Framework-Independent Bridge from AIPlan4EU's Unified Planning Platform to Embedded Systems. In *33rd International Conference on Automated Planning and Scheduling (ICAPS 2023) PlanRob Workshop*.

Saxena, A.; Jain, A.; Sener, O.; Jami, A.; Misra, D. K.; and Koppula, H. S. 2014. Robobrain: Large-scale knowledge engine for robots. *arXiv preprint arXiv:1412.0691*.

Siciliano, B.; Khatib, O.; and Kröger, T. 2008. *Springer handbook of robotics*, volume 200. Springer.

Varghese, R.; and M., S. 2024. YOLOv8: A Novel Object Detection Algorithm with Enhanced Performance and Robustness. In *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*, 1–6.

Wojke, N.; Bewley, A.; and Paulus, D. 2017. Simple Online and Realtime Tracking with a Deep Association Metric. In *2017 IEEE International Conference on Image Processing (ICIP)*, 3645–3649. IEEE.