A Reference Ontology for Collision Avoidance Systems and Accountability Integrated with DAIDALUS

David Martín-Lammerding
Department of Stats. Comput. Sci. Math
Public University of Navarre (UPNA)
Pamplona, Spain
email:david.martin@unavarra.es

José Javier Astrain
Department of Stats. Comput. Sci. Math
Public University of Navarre (UPNA)
Pamplona, Spain
email:josej.astrain@unavarra.es

Alberto Córdoba
Department of Stats. Comput. Sci. Math
Public University of Navarre (UPNA)
Pamplona, Spain
email:alberto.cordoba@unavarra.es

***Abstract*—Unmanned Aerial Systems (UASs) will be deployed massively in urban areas to implement logistic and security applications, with lower cost and more flexibility than manned aircraft. An increasing number of UAS missions requires an improvement of their safety by equipping UASs with Collision Avoidance Systems (CASs). CAS implementations require data from the UAS environment to identify conflicts and to perform an avoidance maneuver if required. UAS generates heterogeneous data from multiple sources, like the Flight Control Unit (FCU), the Global Navigation Satellite System (GNSS), a radio receiver, an onboard-camera, etc. However, each CAS implementation represents, processes and stores conflict data in a different way. Therefore, there is a lack of standards that simplify their development and homologation. To solve this situation, we present a reference knowledge model for any CAS for UAS implemented as a novel application ontology, called Dronetology-cas, and its integration with DAIDALUS, a CAS developed by NASA. Dronetology-cas provides a unified semantic representation within an ontology-based triplet store designed to run in a Single Board Computer (SBC). Its semantic model provides advantages, such as interoperability between systems, machine-processable data and the ability to infer new knowledge. It is implemented using semantic web standards, which contribute to simplify an operational safety audit. Additionally, we integrate Dronetology-cas with DAIDALUS and verify it with two scenarios where conflict data from external sources are considered to improve UAS safety.**

***Index Terms*—Semantic reasoning; ontology; DAIDALUS; UAS; knowledge; conflicts; anti-collision; sensor; embedded; air traffic.**

I. Introduction

The use of Unmanned Aerial Systems (UASs) improves efficiency in logistics applications, infrastructure inspection, emergency situations, etc. as it avoids pilot risk. However, their flights are limited to certain areas of the airspace to avoid encountering other aircrafts. Therefore, each UAS must be equipped with new safety systems to fly safely in a shared airspace, like Collision Avoidance Systems (CASs). CASs detect aircraft, discover potential collision hazards and decide maneuvers to avoid collisions that may change the flight plan initially configured.

A massive use of UASs will imply an increasing collision risk that involves ensuring accountability for all UASs missions. The accountability principle requires UAS operators to take responsibility for what their UAS do in a mission and how they comply with traffic management authorities. UAS operators must have appropriate records to be able to demonstrate their compliance. The accountability of an UAS flight must be ensured because any incident or accident must be able to be investigated by surveyors or authorities. In the worst case, a collision may occur, which must be investigated to determine the cause and to improve CAS.

In our previous work [1] published at SEMAPRO, we present a novel ontology, Dronetology-cas, for CAS for UAS. From those results, we develop an integration with a reference CAS implementation, called DAIDALUS (Detect and AvoID Alerting Logic for Unmanned Systems [2]), as it implements the functional requirements specified in DO-365, the Minimum Operational Performance Standards (MOPS) for UAS developed by RTCA (Radio Technical Commission for Aeronautics) Special Committee 228 (SC-228). DAIDALUS avoidance does not depend on communications with centralized systems, as any delay in making a decision increases the risk of collision. Our based knowledge application also requires to be executed onboard to reduce communication latency. So, we deploy Dronetology-cas in a Single Board Computer (SBC), suitable for mounting on an UAS, to verify its capabilities of collision avoidance, collision investigation and if response time is reasonable. Dronetology-cas includes a Knowledge Base (KB), which consists of triplets of data collected by onboard sensors, external conflict data and inferred knowledge during the UAS mission. Dronetology-cas uses software components to interact with the CAS, onboard sensors and external data sources.

The investigation of aviation incidents and accidents is today recognized as a fundamentally important element of improving safety. The International Civil Aviation Organization (ICAO) releases Annex 13 [3] that requires various States to establish and maintain an accident and incident database to provide an effective analysis of information on actual or potential safety deficiencies. UASs traffic management will be probably inspired by actual commercial aviation recommendations so similar requirements are expected. However,

the increasing number of UASs and the autonomy required for their operation forces process standardization and automation. Ontologies unify schemes for exchange of information and provides access to stakeholders and provide a common format for identifying actual or potential safety deficiencies. Therefore, Dronetology-cas may be integrated with any CAS typology, whose design factors are depicted in Figure 1.

There are external air-traffic data that can be used in collision avoidance. As an example, we consider data from Notice To Airmen (NOTAM), provided by an aviation authority. It alerts aircraft pilots of potential hazards along a flight route or at a location that could affect the safety of the flight. We insert NOTAM data in the KB during a simulated UAS flight, when connectivity allows to download it from an external data source, to consider conflicts that may not be detected with onboard sensors.

ADS-B (Automatic Dependent Surveillance - Broadcast) [4] is a surveillance system that replaces the information currently obtained from radars. It allows to broadcast to other aircrafts the position, obtained from a Global Navigation Satellite System (GNSS), and other flight data. These signals are received by ground or onboard aircraft receivers. UASs can deploy an ADS-B transceiver as part of its integration into the common airspace. Therefore, we also consider ADS-B data in the KB during the simulation performed to improve situational awareness and minimize collisions.

The rest of the paper is structured as follows. Section II presents the state of the art of CAS and accountability systems, Section III defines the problem statement and Section IV describes our contribution. The ontology design is presented in Section V. Section VI is devoted to the integration architecture. Section VII formulates ontology Competency Questions (CQs) and Section VIII summarizes experimental simulations results. Section IX presents the conclusions and references end the paper.

II. Related Work

We review ontology applications, autonomous driving and CAS implementations to identify different approaches. Among them, autonomous driving applications shares some requirements with autonomous UAS and CAS. CAS for autonomous driving requires an accurate localization of the car provided by multiple sensors, as described in [6]. This can also be applied to UASs localization, as in landing maneuvers that require precision, using our proposed ontology-driven system, as it can deal with any precision available. Therefore, a knowledge layer, like we propose, allows to apply it easily to other systems and vehicles.

Ontologies are commonly used to store knowledge in domains related with UAS and CAS, like sensors and air traffic. For the former, we review the Semantic Sensor Network (SSN) [7] ontology that can be used to model UAS as sensors, as it is one of the most widespread application of UAS is data gathering. However, the SSN ontology lacks concepts to model the UAS mission and the CAS, so an extension is required which implies a larger ontology. A large ontology increases memory consumption and we want to run the system in a SBC with limited resources.

Regarding the air traffic domain, we review the propose ontology[8] that applies semantic technologies to air traffic in order to unify heterogeneous data from multiple sources. The ontology implementation presented is performed centralized. However, our proposal is a decentralized ontology implemented in a SBC mounted on each UAS to serve as a knowledge base for the CAS.

Ontology performance is another issue that we review as we want to improve CAS performance using a SBC with limited computing resources. [9] presents a *light-weight* ontology for embedded systems whose design reduces concepts, complexity and query times, compared to the SSN ontology. It is intended for the sensor domain and, therefore, it has limitations for modeling a CAS for UAS, as we want to achieve a knowledge repository to improve CAS performance. However, our proposal also limits the number of classes and the relations for just the necessary.

There are multiple CAS implementations for UAS, but we only consider ACAS-Xu [10] and DAIDALUS [2], as both CAS have their source code available and are two reference CAS implementations. Both CASs require a specific configuration for considering the same avoidance maneuvers and conflict scenario. Given the same scenario, their output formats are different as shown in [11]. A limitation of both CASs is that they do not share a common conceptual model that simplify the usage of knowledge, like the solution we propose.

Multi-Task Learning is an approach for autonomous driving applied to obtain more interpretable results from raw sensor data, thanks to human-readable intermediary representation, as presented in [12]. Our ontological approach can also be considered an intermediate representation, but we also want to facilitate its reusability in other systems, as one key feature of our proposed solution is to improve interoperability. Therefore, we consider that semantic technologies will achieve both objectives.

The accountability of an UAS flight must be ensured because any incident or accident should be able to be investigated by surveyors or authorities. There are systems similar to black boxes for UAS [13]–[15]. They store the UAS's route and the CAS's status. However, the decision-making process prior to a maneuver is complex and its recording is not provided in these systems, therefore, we improve the accountability of a CAS recording decisions and maneuvers in the KB with a common vocabulary that facilitates interoperability.

III. Problem statement

The data required by a CAS depends on how the main design factors are combined. The main concepts of CAS used in the design of Dronetology-cas are described below.

A conflict between two UAS occurs when minimum separation between them is lost. Figure 2 shows a conflict between *local UAS* and *remote UAS* and the protection distance between them, d_p. A loss of separation does not always imply a future

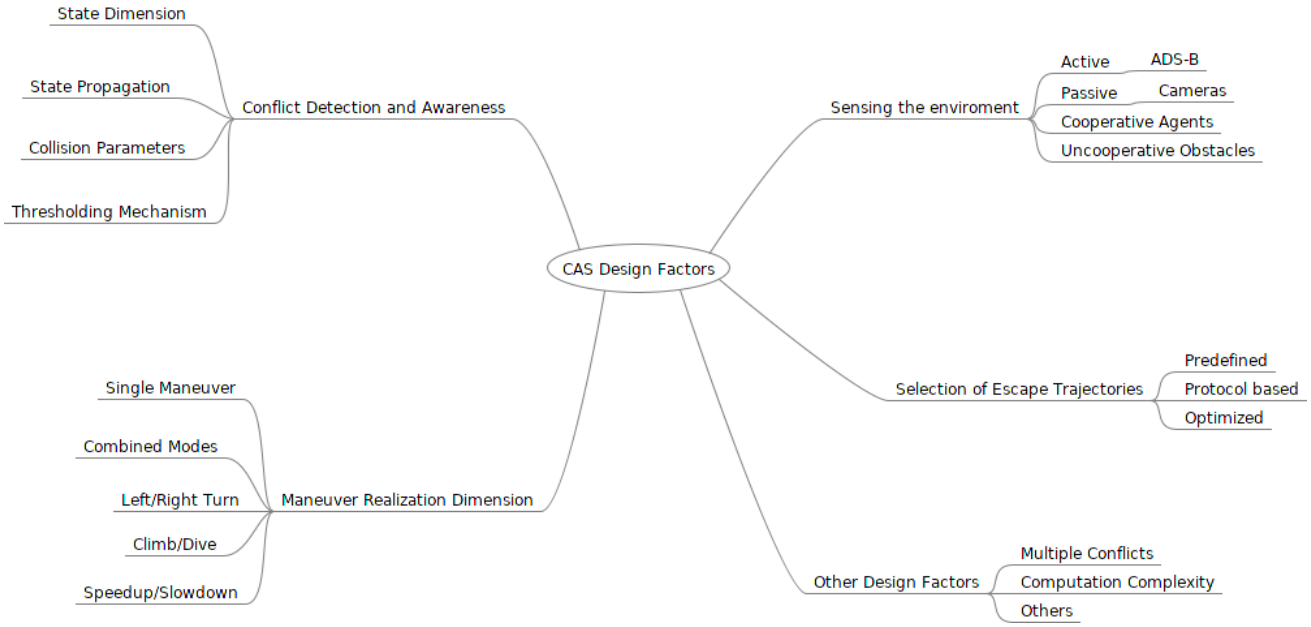

Fig. 1. CAS design factors. Based on [5].

collision, but it is a key safety indicator. A CAS deployed in an UAS allows to maintain a minimum safe separation between UASs. Once a conflict is detected, the onboard CAS deviates the UAS to a safer path. The number of simultaneous conflicts are denoted as N_C. Time to collision t_{tc} is the time required to collide two UAS if both UASs continue at their current speed and on the same path. Lower t_{tc} values correspond to higher risk of collision and it is considered to prioritize conflicts. Very Low Level airspace (VLL) is the space below 500 ft, measured Above Ground Level (AGL). It is the part of the airspace intended for most of the new UAS applications and it will concentrate the largest number of UAS conflicts.

CASs are based on different technologies that collect data from the UAS environment using sensors and/or collaborative elements based on radio receivers/transmitters. UAS can deploy collaborative elements and non-collaborative sensors. A collaborative element broadcast its position and bearing within its coverage and receives from other aircrafts. ADS-B is the most common standard applied in collaborative systems. A non-collaborative sensor detects obstacles and conflicts without requiring other to implement the same system. Technologies applied to non-collaborative systems are vision cameras [16], LIDAR [17], SONAR [18], Radar [19], etc. [20] presents a complete survey of the main technologies applied to sensors for conflict detection.

Most CASs for UASs are distributed, so they run in a SBC mounted on each UAS. However, the size of the UAS limits the weight of the payload, which limits the type and power of processor that can be used. Any software component used in a distributed CAS implementation should be non-compute-intensive to ensure a reasonable response time.

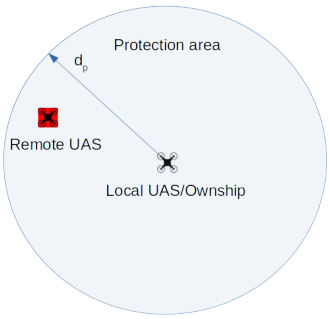

Fig. 2. Conflict between a local UAS (ownship) and a remote UAS.

IV. Contribution

In this paper, we develop the integration of Dronetology-cas with DAIDALUS and verify the integration with two scenarios where knowledge provided by Dronetology-cas improves DAIDALUS collision avoidance. We also describe the ontology Dronetology-cas and its main characteristics. Dronetology-cas improves DAIDALUS as it minimizes the processing of unnecessary conflicts and increases the situational awareness of the UAS translating NOTAMs to conflicts.

Dronetology-cas provides key advantages over other repositories or log storage implementations, as it is ontology based and can be queried using SPARQL (SPARQL Protocol and RDF, Resource Description Framework, Query Language). Another Dronetology-cas key features that we verify with the simulations performed are reasonable response time, modifiability, ease of maintenance, built-in inference capabilities and potential for reuse.

V. Dronetology-cas: The application ontology

Dronetology-cas is an application ontology derived from the domain ontology Dronetology [21]. The domain of Dronetology is UASs. Dronetology-cas formal specification is based on the design factors of a CAS. Next, we review Dronetology before describing Dronetology-cas.

A. Dronetology: The domain ontology

The purpose of Dronetology is to describe concepts that define the components of any UAS, the missions it performs and the environment that surrounds it. Its main applications are the management of bill of materials, the improvement of flight efficiency and autonomous decision making. Dronetology imports external ontologies to avoid redefining concepts from other domains, like SSN for onboard UAS sensors. Another advantage of importing widespread ontologies is that available data from other applications can be integrated easily.

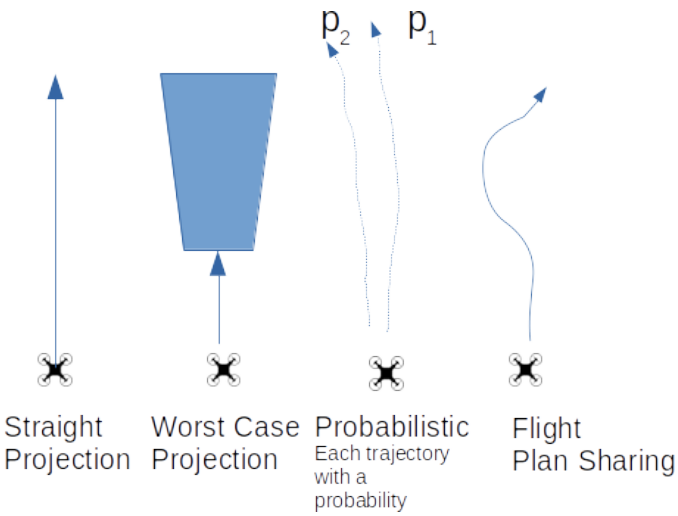

Fig. 3. Methods used for projecting current encounter's information. Based on [5].

B. Dronetology-cas description

We derive the Dronetology-cas application ontology from the Dronetology domain ontology adding concepts from the CAS domain. Its design use abstract concepts that can be used in any CAS integration. The ontology is accessible in [22].

Dronetology-cas allows auditing the CAS decision making process, as the KB stores the planned flight plan, the CAS status and the real flight path of the mission. Conflicts and their temporal variations are detected by onboard sensors and stored in the KB. Differences between planned and real flight can be retrieved using SPARQL queries, as flight plan changes usually are related to conflicts.

Knowledge is obtained from data collected from sensor systems and collaborative elements, and stored in the KB. Data sources are sensors, the Flight Control Unit (FCU) and the GNSS. Inference improves the CAS decisions thanks to knowledge derived from the data. Dronetology-cas has an inference engine that generates new knowledge by applying semantic rules to the KB. The rules are expressed in SPARQL statements [23], [24]. Defined rules inference a conflict's attribute, an evasive trajectory method, a maneuver attribute, etc.

A common feature of a CAS is that it usually runs in a loop with an operation frequency in order to update the internal representation of conflicts, the situation awareness and the current maneuver. This is modeled in Dronetology-cas with the concept of *Iteration*. Dronetology-cas stores CAS status, UAS telemetry and conflicts for each *Iteration* to audit the system. Data collected from sensors are also related to the *Iteration* to provide a complete picture of the environment and the CAS. Dronetology-cas simplifies the integration of data from different sources. It integrates data from any sensor system by defining generic classes, which are not directly dependent on the technology and the implementation. These classes are *NoCollaborativeData* and *CollaborativeData* and both extend *InputData*.

Another common feature of a CAS is to estimate future positions of conflicts, called a projection, in order to obtain a maneuver that will avoid a future collision. As an example of the above, Dronetology-cas allows to choose the method to estimate the future position from multiple options available, as depicted in Figure 3. If the conflict has been detected only through a vision camera, the uncertainty about the heading of the conflict is higher, so the most appropriate method of estimating may be the *Worst Case* method. On the other hand, if the conflict has been detected by a collaborative element, the heading is known and there is less uncertainty. In this case, the *Straight Projection* method is the most appropriate. Both decisions can be derived easily using Dronetology-cas with a rule that relates the conflict data source with the most appropriate estimated method for the projection.

When the CAS decides a maneuver to avoid a collision, Dronetology-cas stores every UAS position and groups them with a individual of class *Maneuver*. Thus, Dronetology-cas relates multiple specific-maneuvers concepts, like *left-turn*, with a set of positions, which allows any combination of trajectories, altitudes and speeds. Similarly, the dynamics of conflicts in the 3D space are stored in Dronetology-cas as different positions at different times. On the contrary, full trajectory prediction made by the CAS are not stored in Dronetology-cas as it is a highly variable data.

C. Dronetology-cas design

The CAS design factors, presented in previous section, are used in the ontology design. Concepts defined in Dronetology-cas are abstractly modeled to fulfill any CAS requirement and simplify the integration. The relevant information that enables making a CAS accountable is identified and annotated with adequate ontology terms. Next, we review each design factor and how Dronetology-cas implements each one.

The first design factor considered in the design is the type of onboard sensors. Dronetology-cas models every onboard

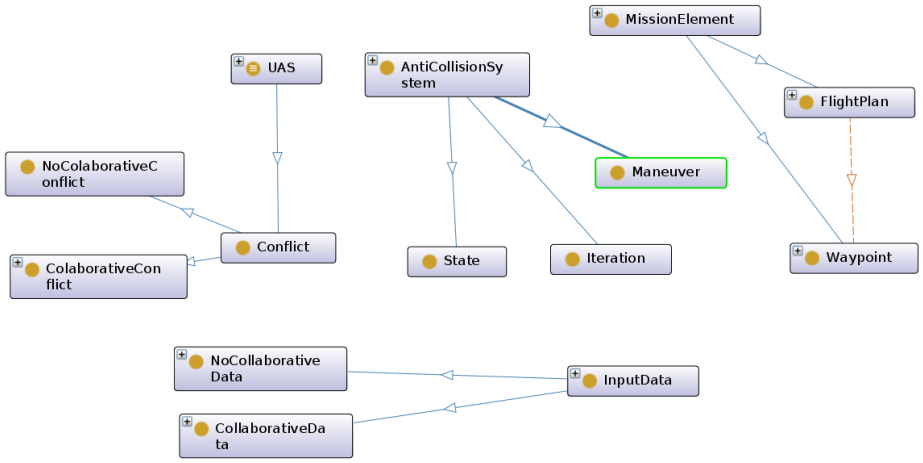

Fig. 4. Dronetology-cas main classes

sensor as an abstract data source so it reduce classes and simplifies the design, avoiding to define specific technologies and attributes for each sensor system. Therefore, sensors are classified between collaborative and non-collaborative.

Dronetology-cas stores conflict locations in every *Iteration* (defined by the period of the CAS) in order to store data for ensuring the system accountability. The ownship location is also periodically stored in Dronetology-cas, distinguishing planned location from locations derived from an evasive maneuvering obtained from the CAS.

The Dronetology-cas design implements knowledge inference to improves the CAS response. For example, inference can classify conflicts, aggregate data from multiple sensors or dismiss a conflict. Inference from historical data are also supported. For example, when a conflict's attribute are not available, like speed, it can be inferred from the conflict past locations. The method to calculate an evasive trajectory and the associated maneuver depends on the CAS implementation.

Dronetology-cas has been designed considering the computational limitations of SBC. Thus, memory usage has been reduced by limiting the number of classes in the model and avoiding importing auxiliary ontologies.

The main classes of Dronetology-cas are *UAS*, *MissionElement*, *InputData*, *AntiCollisionSystem* and *Conflict*. Figure 4 shows the main Dronetology-cas classes.

The class *UAS* describes unmanned aircraft including the communication systems and the ground base. The class *Conflict* is a subclass of *UAS* so in our model only UAS can be conflicts. *MissionElement* is a class that enclose all the elements of a mission. The classes *Waypoint* and *FlightPlan* derive from *MissionElement*.

The class *InputData* represents any data collected from a sensor (non-collaborative), from a collaborative element (radio receiver), from the GNSS or from the FCU. The concepts *NoColaborativeData* and *ColaborativeData* are derived from *InputData* to identify a conflict and its source type. The property *drone:detect* is an object property that relates individuals of *NoColaborativeData* or *ColaborativeData* with individuals of class *Conflict*.

Some classes in Dronetology-cas have geographic data defined as datatype properties. The latitude and longitude are relative to the World Geodetic System 1984 (WGS84) coordinate system. The altitude is relative to Mean Sea-Level (MSL). To improve interoperability, the *Conflict* class uses *geo:wktLiteral* datatype with a WGS 84 geodetic latitude-longitude. This allows Dronetology-cas to implement a geospatial web service that could be reused and recombined to fulfill a user query using a common standard.

The class *AntiCollisionSystem* groups elements of any CAS. The classes *State*, *Maneuver*, *NextIterationLocation* and *Iteration* are derived from it. The state of the CAS are represented as instances of the class *State* with an attribute that codifies it. A class *Iteration* instance relates all the knowledge stored in the KB at an instant of time through the object-property *hasIteration*.

The class *Maneuver* defines a set of locations of the UAS when the CAS is active. CAS may calculate one or multiple location alternatives for the UAS to avoid the collision, stored as instances of the class *NextIterationLocalUASLocation*, grouped by an instance of the class *Maneuver* through the object-property *hasManeuver*. In every iteration, at least a new instance of *NextIterationLocalUASLocation* is stored in the KB and sent to the FCU by the CAS.

Flight safety compliance are implemented in Dronetology-

cas defining attributes of the class *Conflict* to translate the severity classification requirements defined in ESARR2 [25].

VI. System Architecture

Dronetology-cas integration with a CAS can be implemented in two ways: *repository-mode* or *knowledge-mode*. The *repository-mode* of Dronetology-cas is a data sink, intended for flight telemetry storage. The *knowledge-mode* extends the *repository-mode* adding a connector or an endpoint to provide knowledge. Figure 5 shows the system architecture for each integration mode.

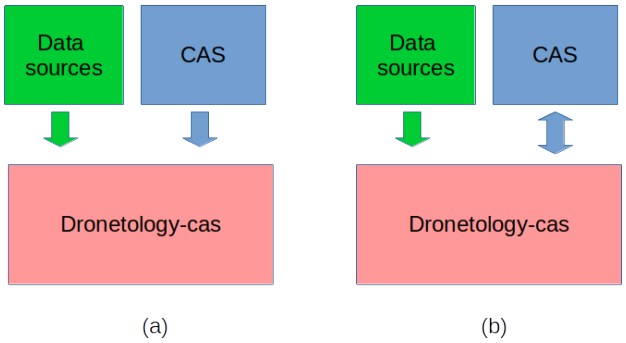

Fig. 5. Dronetology-cas integration alternatives: (a) *repository-mode* (b) *knowledge-mode*

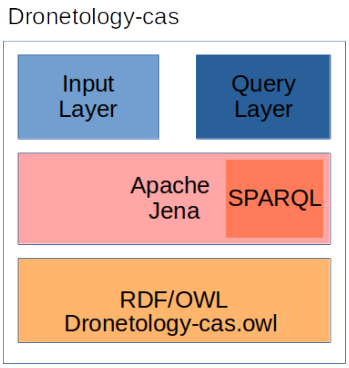

Fig. 6. Dronetology-cas system architecture.

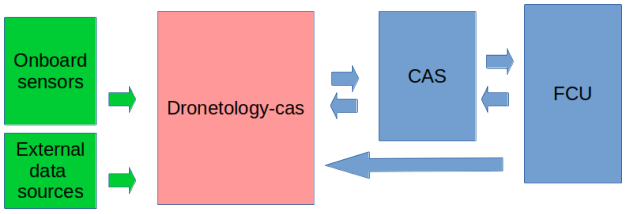

Fig. 7. Dronetology-cas *knowledge-mode* integration architecture detailed.

Dronetology-cas system architecture defines software components that allow its integration with other UAS subsystems, like a CAS, onboard sensor systems, the FCU and external data sources. The components of the Dronetology-cas architecture consists of an input data layer and a query layer, as shown in Figure 6. The input layer, implemented for the *repository-mode*, is a software layer that insert into the KB collected sensor data, CAS status, GNSS locations, FCU telemetry and safety information form external providers.

The *knowledge-mode* architecture is detailed in Figure 7. The query layer, implemented for the *knowledge-mode*, is a SPARQL adapter that executes sentences and return values. The query layer provides a knowledge base criteria to select conflict detected or received by the onboard sensor. It adds implicit knowledge inference and reasoning capabilities to some CAS functions, such as conflict detection or new path selection. It can be configured as a SPARQL endpoint if HTTP interconnection is required for incident investigation, as federated SPARQL may be convenient to access other endpoints.

External conflict data sources relies on available connectivity to insert data in the KB. NOTAM providers and Ground Surveillance Radars (GSRs) are examples of external data sources.

Dronetology-cas is developed using the Web Ontology Language (OWL) language [26]. The main languages used to develop CAS (C, C++, Python) have implementations to process RDF triplets [27] and ontologies in OWL format.

Dronetology-cas architecture requires multi-platform compatibility to run in different hardware. Different software components are available to implement a semantic web stack, and among them, we select Apache Jena as it is the most common stack. Apache Jena is an open source Java framework for building Semantic Web and Linked Data applications. Additionally, its only dependencies are the Java Runtime Enviroment (JRE) so it can be deployed in every SBC that has a JRE implementation.

TABLE I
Dronetology-cas competency questions

CQ_1	How many conflicts are detected?
CQ_2	Which UAS has the highest priority among the UAS in conflict?
CQ_3	Which conflict has the shortest time to collision?
CQ_4	Has the number of conflicts increased or decreased?
CQ_5	How has been detected the conflict with a given UAS?
CQ_6	How long it has taken to resolve a conflict?
CQ_7	Has the distance flown been increased with respect to the flight plan?
CQ_8	In which locations have there been conflicts?
CQ_9	Where and when was the collision?
CQ_{10}	How many UAS were in conflict before the collision?
CQ_{11}	What UAS has it collided with?
CQ_{12}	What maneuver was the UAS performing before the collision?

VII. Competency Questions

We define a set of Competency questions (CQs) to define the knowledge that has to be entailed in Dronetology-cas. These questions, listed in Table I, has been used to validate Dronetology-cas. Some CQs are suitable for an UAS mission audit process. Others can assist the CAS in a decision making process, when Dronetology-cas is integrated in *knowledge-mode*. There are CQs that are intended to find out how the conflict has been resolved, e.g., CQ_6, CQ_7 and CQ_8. Some

CQs help to find out what happened and how when a collision happens, e.g., CQ_9, CQ_{10}, CQ_{11} and CQ_{12}.

In a *knowledge-mode* integration, the CAS uses the results of some CQs to make decisions. Continuing the previous example, the CQ *What type of conflict is X?* allows the CAS to select the most appropriate way of calculating the future position of the conflict. Other CQs are intended for a security audit of the CAS. An example of this is the CQ used to check when and where a collision occurred.

VIII. Simulation

Dronetology-cas verification is performed using simulations to asses its response time and its integration with DAIDALUS. Testing CAS with live-fly field experiments are costly and highly time consuming. Therefore, simulations are more convenient thank to their flexibility and easy of configuration.

The first simulation is a performance evaluation of the KB of Dronetology-cas when queried with some selected CQs translated to SPARQL sentences. We develop a *simulated-CAS* that loads data in the KB during the simulation time to test the effect of the growth of the number of triplets stored in the KB and to measure the performance when executing SPARQL queries.

The second simulation performed is a Dronetology-cas *knowledge-mode* integration with DAIDALUS. A custom FCU simulator is developed in Java 8 to generate UAS flight positions (expressed in WGS-84 coordinates) from an initial flight plan. Multiple UASs can be simulated by the FCU simulator as it can generate positions of the ownship and the conflicting UASs sharing a common time reference.

We consider for the simulations the well known SBC Raspberry Pi 3 Model B+ (Pi3) [28], as it is shipped in large numbers and it has a huge user base. Its processor is a 1.2 GHz 64-bit quad core ARM Cortex-A53. The operative system installed to perform the simulations is the Raspberry Pi OS 32 bits. The FCU simulator runs in an external desktop computer that is connected with the Pi3 through a network connection.

A. Performance evaluation

Dronetology-cas performance is tested in both modes, *repository-mode* and *knowledge-mode*. The performance evaluation is executed using the *simulated-CAS* that loads the KB dynamically with data from conflicts and queries the KB using CQs translated to SPARQL sentences.

The most generic CQs have been selected to measure response times and memory footprint, as they are the most likely to be used in any integration mode. CQ_1 and CQ_3 are necessary for any auditing process to review conflicts and their status. CQ_5 and CQ_6 provide knowledge that the CAS can use to modify its response to conflicts. One hundred repetitions of each case were performed to calculate the mean and the standard deviation. The results obtained from the response times and memory footprint are shown in Table II. CQs considered are translated to SPARQL, available in [29].

Response time and memory footprint are measured with different number of triplets stored in the KB. Memory footprint has been measured using the Java 8 API. The number of triplets with conflicts and CAS data grows as the UAS flies. Therefore, the flight duration determines the number of triplets stored in the KB. In our tests, we have simulated up to 10000 triplets that corresponds to 15 minutes of flight by inserting an average of 10 triplets per second.

The response time affects the CAS depending on the integration type chosen. In *repository-mode*, there are no strict response time requirements as it is not required a short response time. However, in *knowledge-mode*, the response time delays the CAS decisions. For our purpose, a suitable response time should allow to take a decision with the most recent data, before new data is available, that is, the response time should be below the refreshing rate of incoming data. Each sensor system has its refreshing rate ranging from 1 Hz of ADS-B until 20 Hz of a vision camera [30]. The response time of CQ_5 and CQ_6 obtained complies with the previous criteria as long as the number of triplets are below approximately 1000 triplets.

Figure 8 shows that Dronetology-cas response time increases when the number of triplets increases. Memory consumption grows as the UAS flies as well. That is, the duration of the UAS flight increases the response time. The worst response time is at the end of a flight. This result is due to our limited implementation of the software components that instantiates and queries the KB. An option to scale up is to have two instances of Dronetology-cas model, each with a different purpose, one instance for the *repository-mode* and the other for the *knowledge-mode*. The instance for the *repository-mode* should store all triplets, but the instance for the *knowledge-mode* should keep only recent triplets needed for the inference process.

B. Dronetology-cas knowledge-mode integration with DAIDALUS

DAIDALUS is a reference implementation of the algorithm for the Phase 1 Minimum Operational Performance Standards (MOPS) for Detect and Avoid (DAA). It is available a prototype implementations written in Java. It serves as a reference of a MOPS-compliant DAA algorithm so we select it to verify Dronetology-cas *knowledge-mode* integration.

Dronetology-cas integration with DAIDALUS v2.0.2b is developed in Java, using a knowledge layer that provides data location from the ownship and the remote conflicting aircraft to DAIDALUS.

The integrated system is simulated using the FCU simulator to generate the ownship location and other conflicting aircrafts when necessary. The simulation is a continuous execution that inserts locations in the KB while flying a defined flight plan. In each iteration, Dronetology-cas provides DAIDALUS the current position of the conflicting-UAS and the ownship. DAIDALUS calculates a response in form of a new location for the ownship that it is inserted in the KB. The simulations performed are autonomous flights where the ownship follows the corrective guidance provided by DAIDALUS without pilot interaction.

TABLE II
RESPONSE TIME (IN MILLISECONDS) AND MEMORY FOOTPRINT (IN KILOBYTES) OF *repository-mode* AND *knowledge-mode* IMPLEMENTED IN APACHE JENA RUNNING IN A PI3.

No triplets	*Repository-mode*								*Knowledge-mode*							
	CQ_1				CQ_3				CQ_5				CQ_6			
	Response time		Memory footprint		Response time		Memory footprint		Response time		Memory footprint		Response time		Memory footprint	
	mean	sdev	mean	sdev	mean	sdev	mean	sdev	mean	sdev	mean	sdev	mean	sdev	mean	sdev
100	18.95	4.13	5025.28	1418.13	26.68	7.83	5101.67	1420.78	19.98	5.91	5104.45	1420.83	18.21	7.45	5100.81	1421.06
250	23.54	2.26	5200.98	1308.69	23.38	2.17	5241.25	1308.82	24.12	4.18	5211.77	1308.21	24.03	2.23	5233.40	1305.41
500	38.10	2.97	5441.26	1327.53	38.14	3.10	5441.51	1322.02	52.37	18.03	5377.53	1327.33	39.05	3.56	5491.02	1322.82
1000	74.42	17.78	5986.61	1332.65	67.47	3.10	5983.91	1336.00	68.49	3.18	6061.40	1331.63	68.74	3.10	5969.27	1319.01
2500	165.62	36.42	6997.64	1342.78	171.69	46.86	3362.77	1727.86	173.87	55.31	3875.86	1768.38	160.60	6.81	6972.95	1315.36
5000	320.98	64.36	6004.34	1718.44	320.30	63.49	5391.63	1587.92	355.39	100.29	5309.72	1510.33	324.63	65.44	5935.41	1724.54
10000	662.00	162.01	9545.00	1744.46	662.24	164.92	8440.54	2154.39	654.65	151.11	8560.70	2140.84	670.81	162.15	9595.25	1752.67

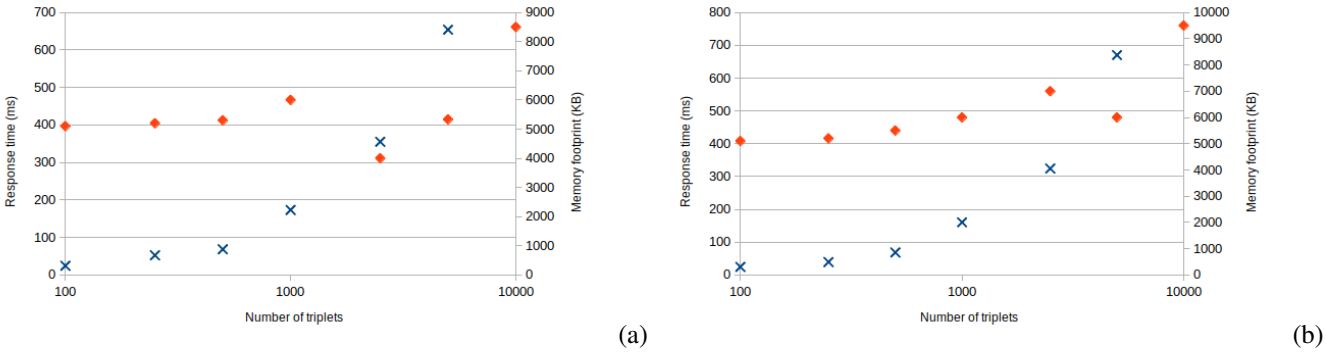

Fig. 8. Response time (x) and memory footprint (♦) for *knowledge-mode* for CQ_5(a) and CQ_6(b).

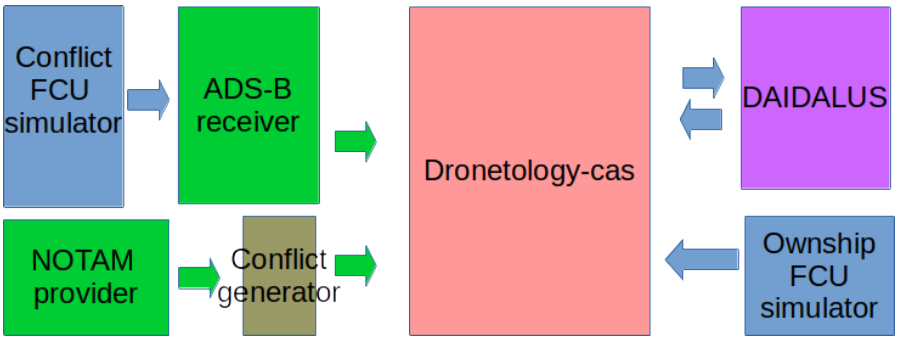

Fig. 9. Simulation architecture implemented to verify Dronetology-cas *knowledege mode* integration.

Next, we present two simulated scenarios for the *knowledge-mode* integration where a modification of the architecture depicted in Figure 9 is developed, whose main difference is that the KB is connected to the onboard sensors. The simulation of both scenarios are performed in a Pi3.

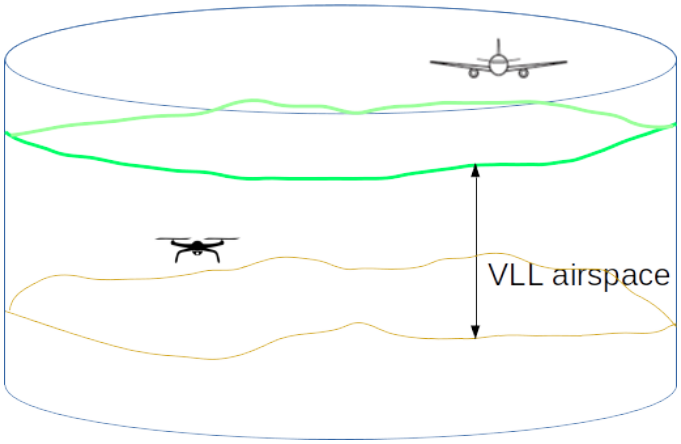

Fig. 10. Conflict not considered by Dronetology-cas as the flying airspace class of the aircraft and the UAS are not the same. Based on [31]

1) Scenario 1: Using ADSB knowledge: ADS-B is a common surveillance technology that increases situational awareness between both manned and unmanned operations. The availability of ADS-B equipment and its decreasing cost have contributed to mount it in UASs. Nowadays, the majority of aircraft broadcast ADS-B messages constantly. Starting from the year 2020, civil aviation aircraft in Europe and United States are required to be ADS-B compliant. Airspace is divided in classes specified by the International Civil Aviation Organization (ICAO) [32]. In our simulated scenario, depicted in Figure 10, manned aircraft and UASs do not share same airspace class. The developed FCU simulator inserts ADS-B data in the KB, as *ADBSConflict* class instances, that refers to a remote conflicting aircraft. Altitude is an attribute of the *ADBSConflict* class. It is obtained from the aircraft airborne position ADS-B message that is used to broadcast the position and altitude of the aircraft. It has the Type Code 9–18 and 20–22. When Type Code is from 9 to 18, the encoded altitude represents the barometric altitude of the aircraft. When the Type Code is from 20 to 22, the encoded altitude contains the GNSS altitude of the aircraft.

In our proposed scenario, a small UAS, equipped with an ADS-B transceiver, flies in VLL and receives ADS-B messages from an aircraft flying in the class A airspace. In a common configuration, every ADS-B message received are transformed into a conflict that are processed by DAIDALUS. However, DAIDALUS does not provide an evasive maneuver in this case because vertical separation is larger than the minimum configured, but this processing requires some time. When the number of conflicts flying in a different airspace class increases the delay may affect response time of the overall system. To see this effect we simulate multiple conflicts and the response time of DAIDALUS. Results, that are depicted in Table III, show that DAIDALUS Response Time (RT) increases when the number of conflicts increases. The mean of the RT has been measured repeating DAIDALUS execution 100 times for each number of conflicts.

TABLE III
DAIDALUS RESPONSE TIME CHANGING THE NUMBER OF SIMULTANEOUS CONFLICTS.

Number of conflicts	0	1	2	3	4
DAIDALUS RT (ms)	4	157	175	174	185

Dronetology-cas integration in *knowledge-mode* avoids this delay due to conflicts flying in different airspace classes, as the criteria defined with a SPARQL sentence provides DAIDALUS only conflicts in the same airspace zone, avoiding aircraft that fly out of VLL airspace. The SPARQL sentence, depicted in Listing 1, defines a criteria that selects conflicts received by an ADS-B transceiver, that fly in VLL (below 500 ft) and its ADS-B type is UAV. When repeating the previous simulation with the Dronetology-cas integration, no conflicts are returned to DAIDALUS.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX drone: <http://www.dronetology.net/dronetology-cas.owl#>

SELECT ?latitude ?longitude ?altitude
WHERE
{
        ?conflict a drone:Conflict.
        ?conflict drone:hasLatitude ?latitude.
        ?conflict drone:hasLatitude ?longitude.
        ?conflict drone:hasLatitude ?altitude.
        ?conflict drone:hasLatitude ?type.
        ?conflict rdf:type ?ADSBConflict.
        FILTER(?altitude<500)
        FILTER(?type="UAV"^^xsd:string)
}
```

Listing 1. SPARQL statement to retrieve conflicts received by ADS-B and flying in VLL

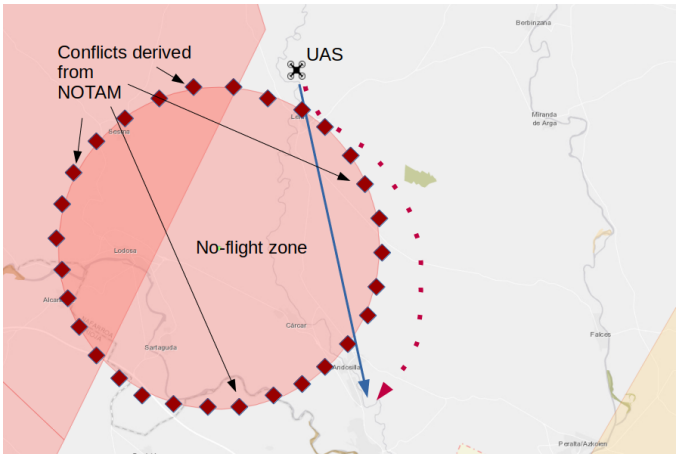

Fig. 11. A no-flight zone defined in a NOTAM is translated to multiple conflicts in the no-flight zone boundary. UAS planned flight (blue) and evasive path (red) are represented.

Therefore, Dronetology-cas integration avoids unnecessary processing delays, as shown in the simulation results obtained, and also it simplifies maneuvers when one or more conflicts are dismissed.

2) Scenario 2: Using NOTAM knowledge: A CAS usually considers conflicts detected by collaborative or/and non-collaborative onboard transceivers or sensing devices. However other risks are known such as those listed in NOTAMs. NOTAMs are usually considered when planning an UAS flight but they may change so any subsequent update may not be considered. In the simulated scenario proposed, Dronetology-cas translates external data from a NOTAM provider to standard conflicts triplets that can be used by DAIDALUS during the UAS flight. The download of NOTAMs requires connectivity to external providers. As NOTAMs refresh rate is unknown we configure a periodical update of the NOTAM data stored in the KB.

The simulation performed consist of an UAS crossing a no-flight zone that it is defined in a NOTAM published during the flight so it has not been taken into account to plan the UAS flight, as depicted in Figure 11. DAIDALUS configuration for this simulation avoids altitude changes in any of the corrective

guidance generated, therefore, only horizontal maneuvers are considered. A *conflict-generator* software is developed to translate any NOTAM into multiple conflicts in the boundary of the no-flight zone. Conflicts generated are inserted in the KB. To avoid unnecessary processing, the *conflict-generator* only translates NOTAMs that are near the waypoints of the flight plan of the UAS.

The number of conflicts derived from a NOTAM and their positions are estimated using the default configuration of DAIDALUS (the CD3D parameters). The main parameter used is the Horizontal Miss Distance threshold (HMD) to separate generated conflicts so the *conflict-generator* avoids creating an excessive number of conflicts, so the no-flight zone boundary is partially covered.

During the simulated flight, the *conflict-generator* inserts conflicts considering the new NOTAN published and the KB provides DAIDALUS conflicts in the boundary of the no-flight zone, so DAIDALUS calculates a maneuver that changes the path of the UAS, avoiding the no-flight zone, as shown in the red dotted line in Figure 11. Therefore, Dronetology-cas integration increases situational awareness of the UAS as it allows to translate hazards, like no-flight zones, to conflicts that can be managed by DAIDALUS.

Listing 2 depicts the SPARQL query used to retrieve conflicts, including the ones generated from the NOTAM, in order to modify the UAS path and bypass a no-flight zone.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX drone: <http://www.dronetology.net/dronetology-cas.owl#>


SELECT ?latitude ?longitude ?altitude
WHERE
{
        ?conflict a drone:Conflict.
        ?conflict drone:hasLatitude ?latitude.
        ?conflict drone:hasLongitude ?longitude.
        ?conflict drone:hasAltitude ?altitude.
        FILTER(?altitude<500)
}
```

Listing 2. SPARQL statement to retrieve conflict data

IX. Conclusion

In this paper, we describe the Dronetology-cas ontology, as a value-added component for any CAS, and its integration with a specific one, DAIDALUS. Dronetology-cas integration modes allows multiple configurations for different purposes, such as being a black box or a conflict knowledge manager. Self developed software components integrate Dronetology-cas with DAIDALUS to provide knowldege. Results obtained from simulations performed show that knowledge available in the KB improves DAIDALUS capabilities and performance, reducing unnecessary processing and increasing safety. This was achieved inserting ADS-B data and NOTAM information as conflicts in the KB and defining SPARQL sentences to fetch conflicts. The installation of Dronetology-cas in a production-ready integration on a SBC should take into a account the performance results and the integration mode required to balance response time and memory consumption as obtained in the simulations performed.

Future work will be focused on improving performance, testing different SBC implementations, customizing the default linux-based OS, considering others, like Microsoft Windows IoT Core, and building a specific OS distribution devoted for CAS applications.

Another line of work is to create a *dataset* with semantic mission data to be used for research of UAS air traffic. Software components developed for the simulation could be evoluted to a generic software simulation for UASs. Further developments of this work will develop an extension of Dronetology-cas to manage other UAS subsystems, like the path planner. These developments have the potential to expand the initial Dronetology-cas domain and to achieve an ontology standard for autonomous UAS.

References

[1] D. Martín-Lammerding, J. J. Astrain, and A. Cordoba, "A reference ontology for collision avoidance systems and accountability," in *SEMAPRO 2021, The Fifteenth International Conference on Advances in Semantic Processing*, IARIA, 2021, pp. 22–28.

[2] C. Muñoz, A. Narkawicz, G. Hagen, J. Upchurch, A. Dutle, M. Consiglio, and J. Chamberlain, "Daidalus: Detect and avoid alerting logic for unmanned systems," in *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, IEEE, 2015, 5A1–1.

[3] ICAO. (2021). Annex 13 — aircraft accident and incident investigation, [Online]. Available: https://www.icao.int/safety/airnavigation/aig/pages/documents.aspx (visited on 05/30/2022).

[4] C. Rekkas and M. Rees, "Towards ads-b implementation in europe," in *2008 Tyrrhenian International Workshop on Digital Communications-Enhanced Surveillance of Aircraft and Vehicles*, IEEE, 2008, pp. 1–4.

[5] B. Albaker and N. Rahim, "A survey of collision avoidance approaches for unmanned aerial vehicles," in *2009 international conference for technical postgraduates (TECHPOS)*, IEEE, 2009, pp. 1–7.

[6] J. Phillips, J. Martinez, I. A. Bârsan, S. Casas, A. Sadat, and R. Urtasun, "Deep multi-task learning for joint localization, perception, and prediction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, 2021, pp. 4679–4689.

[7] W. W. Group. (2021). SSN, Semantic Sensor Network Ontology, https://www.w3.org/tr/vocab-ssn/, [Online]. Available: https://www.w3.org/TR/vocab-ssn/ (visited on 05/30/2022).

[8] R. M. Keller, S. Ranjan, M. Y. Wei, and M. M. Eshow, "Semantic representation and scale-up of integrated air traffic management data," in *Proceedings of the International Workshop on Semantic Big Data*, 2016, pp. 1–6.

[9] H. Rahman and M. I. Hussain, "A light-weight dynamic ontology for internet of things using machine learning technique," *ICT Express*, 2020.

[10] M. P. Owen, A. Panken, R. Moss, L. Alvarez, and C. Leeper, "Acas xu: Integrated collision avoidance and detect and avoid capability for uas," in *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*, IEEE, 2019, pp. 1–10.

[11] J. T. Davies and M. G. Wu, "Comparative analysis of acas-xu and daidalus detect-and-avoid systems," *National Aeronautics and Space Administration NASA Ames Research Center; Moffett Field CA United States Technical Report NASA/TM-2018-219773 ARC-E-DAA-TN50499*, 2018.

[12] B. Zhou, P. Krähenbühl, and V. Koltun, "Does computer vision matter for action?" *Science Robotics*, 2019.

[13] Redcat Holdings. (2021). Drone Box, https://www.redcatholdings.com/drone-box, [Online]. Available: https://www.redcatholdings.com/drone-box (visited on 05/30/2022).

[14] Tl-Elektronic. (2021). Black box, https://www.tl-elektronic.com/, [Online]. Available: https://www.tl-elektronic.com/index.php?page=uav&p_id=40&lang=en (visited on 05/30/2022).

[15] UAV Navigation. (2021). Black Box https://www.uavnavigation.com/, [Online]. Available: https://www.uavnavigation.com/sites/default/files/docs/2021-03/UAV%20Navigation%20FDR01%20Brochure.pdf (visited on 05/30/2022).

[16] D. Zuehlke, N. Prabhakar, M. Clark, T. Henderson, and R. J. Prazenica, "Vision-based object detection and proportional navigation for uas collision avoidance," in *AIAA Scitech 2019 Forum*, 2019, p. 0960.

[17] U. Papa, G. Ariante, and G. Del Core, "Uas aided landing and obstacle detection through lidar-sonar data," in *2018 5th IEEE International Workshop on Metrology for AeroSpace (MetroAeroSpace)*, IEEE, 2018, pp. 478–483.

[18] U. Papa, "Sonar sensor model for safe landing and obstacle detection," in *Embedded Platforms for UAS Landing Path and Obstacle Detection*, Springer, 2018, pp. 13–28.

[19] N. Gellerman, M. Mullins, K. Foerster, and N. Kaabouch, "Integration of a radar sensor into a sense-and-avoid payload for small uas," in *2018 IEEE Aerospace Conference*, IEEE, 2018, pp. 1–9.

[20] A. Muraru, "A critical analysis of sense and avoid technologies for modern uavs," *Advances in Mechanical Engineering ISSN: 2160-0619*, vol. 2, Mar. 2012. DOI: 10.5729/ame.vol2.issue1.23.

[21] D. Martín-Lammerding. (2021). Dronetology, the UAS Ontology, https://dronetology.net/dronetology, [Online]. Available: https://dronetology.net/dronetology (visited on 05/30/2022).

[22] ——, (2021). Dronetology-cas, the anti-collision ontology, https://dronetology.net/dronetology-cas, [Online]. Available: https://dronetology.net/dronetology-cas (visited on 05/30/2022).

[23] Web Working Group. (2021). SPARQL Query Language for RDF, https://www.w3.org/2001/sw/wiki/sparql, [Online]. Available: https://www.w3.org/TR/sparql11-query/ (visited on 05/30/2022).

[24] ——, (2021). Spin Working Group, Rules for SPARQL, https://www.w3.org/submission/spin-sparql/, [Online]. Available: https://www.w3.org/Submission/spin-sparql/ (visited on 05/30/2022).

[25] T. Licu, F. Cioran, B. Hayward, and A. Lowe, "Eurocontrol—systemic occurrence analysis methodology (soam)—a "reason"-based organisational methodology for analysing incidents and accidents," *Reliability Engineering & System Safety*, vol. 92, no. 9, pp. 1162–1169, 2007.

[26] Web Working Group. (2021). Web Ontology Language (OWL), https://www.w3.org/owl/, [Online]. Available: https://www.w3.org/owl/ (visited on 05/30/2022).

[27] ——, (2021). Resource Description Framework (RDF), https://www.w3.org/2001/sw/wiki/rdf, [Online]. Available: https://www.w3.org/rdf/ (visited on 05/30/2022).

[28] Raspberry Pi Foundation. (2021). Raspberry Pi 3, https://www.raspberrypi.org/, [Online]. Available: https://www.raspberrypi.org/ (visited on 05/30/2022).

[29] D. Martín-Lammerding. (2021). Competency questions in sparql, https://dronetology.net/sim/competency-questions.zip, [Online]. Available: https://dronetology.net/sim/competency-questions.zip (visited on 08/02/2021).

[30] S. Graham, J. De Luca, W.-z. Chen, J. Kay, M. Deschenes, N. Weingarten, V. Raska, and X. Lee, "Multiple intruder autonomous avoidance flight test," in *Infotech@ Aerospace 2011*, 2011, p. 1420.

[31] UAVionix. (2021). Inert and alert: Intelligent ads-b for uas nas integration, [Online]. Available: https://uavionix.com/downloads/whitepapers/Inert-and-Alert-CONOPS.pdf (visited on 05/30/2022).

[32] I. Frolow and J. H. Sinnott, "National airspace system demand and capacity modeling," *Proceedings of the IEEE*, vol. 77, no. 11, pp. 1618–1624, 1989.