OGC Testbed-16

Full Motion Video to Moving Features Engineering Report

Publication Date: 2021-01-13

Approval Date: 2020-12-15

Submission Date: 2020-11-20

Reference number of this document: OGC 20-036

Reference URL for this document: http://www.opengis.net/doc/PER/t16-D021

Category: OGC Public Engineering Report

Editor: Emeric Beaufays, C.J. Stanbridge, Rob Smith

Title: OGC Testbed-16: Full Motion Video to Moving Features Engineering Report

OGC Public Engineering Report

Table of Contents

Chapter 1. Subject

This OGC Testbed-16 Engineering Report (ER) evaluates the suitability of existing OGC standards for the generation of Moving Features from Full Motion Video (FMV) that has an embedded stream of detected moving objects.

This ER presents several proof of concept applications that accept FMVs, with multiple encoded Video Moving Target Indicators (VMTI), and combines the VMTIs into separate tracks that are then encoded to OGC Moving Features.

In addition, the ER explores the generation of records encoded according to OGC Sensor Model Language (SensorML) 2.0 standard describing the collection platform and relevant telemetry information from the key-value stream content encoded according to the MISB 0601 and 0903 specifications of the Motion Imagery Standards Board (MISB).

Chapter 2. Executive Summary

This OGC ER documents work completed in the OGC Testbed-16 Full Motion Video (FMV) thread.

Unmanned Aerial Vehicles (UAVs) are increasingly used in military and civil applications. A video camera is often the primary sensor on a UAV, providing a torrent of imagery of the overflown ground. From this imagery, detecting the presence of moving objects on the ground, such as vehicles and people, can be of strategic importance.

Modern Motion Imagery platforms are capable of extracting a considerable body of information from the raw collection and then streaming that information in-band with the video stream. However, this information comes in a form which is not readily exploitable except by specialized systems.

STANAG 4609, MISB Std. 0601 (UAS metadata) and MISB Std. 0903 (VMTI) define encoding a video with frame by frame metadata that may contain any number of VMTIs. However, the individual moving object detections can be made more useful when combined into moving features or "tracks". This process is summarized in the tracking algorithm chapter. This process is a non-trivial problem. The experimental methods and results are further described in this document.

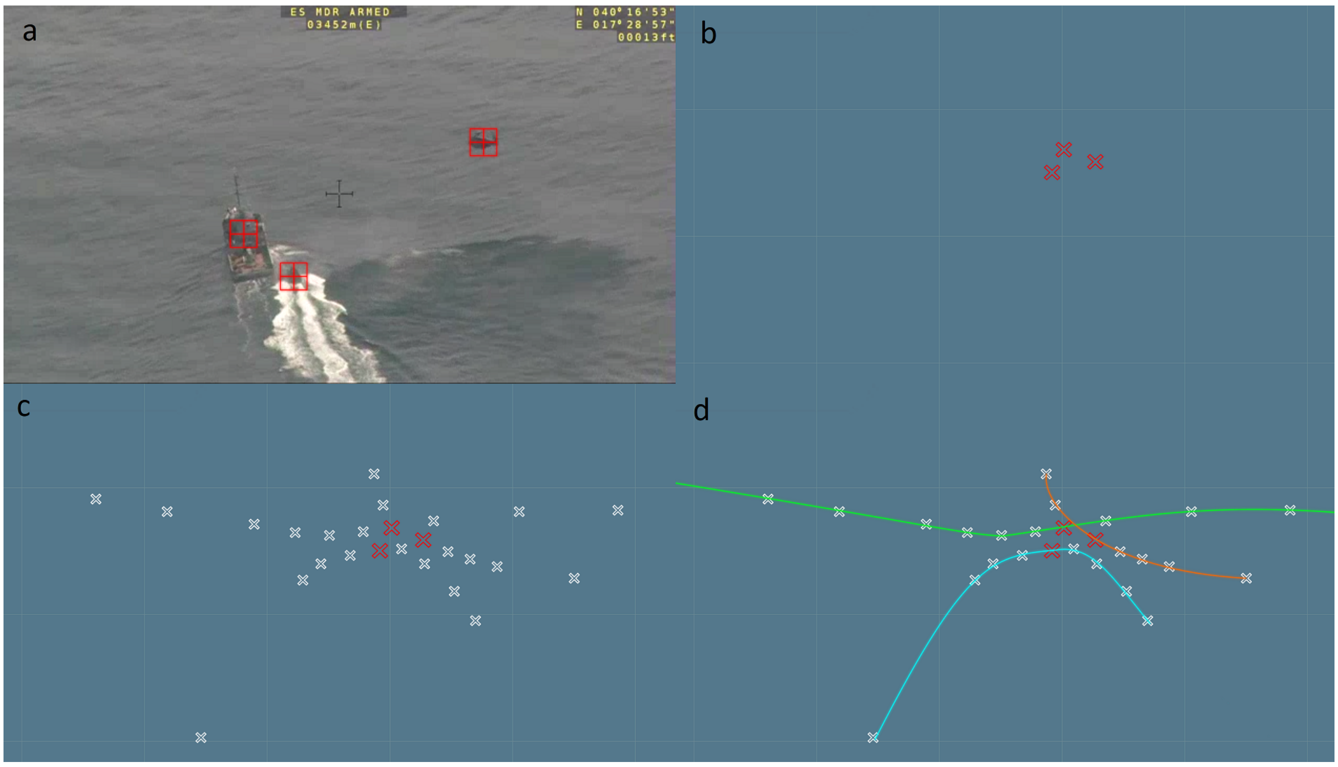


Figure 1. This figure outlines the processing steps that lead to the production of Moving Features from VMTIs.

(a) Each frame of the video can contain several detections. (b) The detections are decoded and referenced according to the camera telemetry. (c) The detections throughout the video are accumulated. (d) The different tracks are inferred.

In addition, the OGC has developed several standards to serve as the interoperability medium for the outcome and description of the above process which are assessed in their suitability to the FMV to moving feature scenario.

This report explores the following topics:

1. Assessment of the suitability of OGC Moving Features and OGC O&M as interoperability formats for moving features inferred from VMTI detections.
2. Implementation of a process to generate tracks from VMTI detections.
3. Encoding to OGC Moving Features, OGC O&M, and WebVMT [https://w3c.github.io/sdw/proposals/geotagging/webvmt/].
4. Generation of OGC SensorML 2.0 records describing the collection platform and relevant telemetry information from the key-value stream (0601 and 0903 content).
5. Implementation of demonstrators for the generation of OGC Moving Features, Observations, and SensorML documents from sample Full Motion Video streams.
6. Subsequent conversion of OGC Moving Features data into the SensorThings API, permitting access to JSON-encoded Moving Features observations through SensorThings-based sites and architectures.

2.1. Key findings and results

OGC Moving Features is an easy to read and write format with csv, json and xml implementations. The standrd specifies how to encode the result of a tracking algorithm in a simple and readable fashion and enables a one to one mapping of all the metadata from the original MISB 0903 VMTIs. However, the mapping of certain properties like embedded Geography Markup Language (GML) and Web Ontology Language (OWL) classes can be complex and will be difficult to define an encoding for those properties that can be interpreted by a generic decoder.

OGC O&M offers similar capabilities relative to OGC Moving Features with the difference that specific attention is given to map ontologies. This is an advantage of OGC O&M over OGC Moving Features but comes at a cost of complexity and is only useful for MISB 0903 data that carries relevant metadata.

SensorThings is a useful API in which to encode results from Full-Motion Video and Moving Features sources due to its versatile support for displaying readings from varied sensors. The toolset that converts FMV sources to Moving Features and SensorThings (ST) follows a consistent high-level workflow for conversion the outputs to be accessible via the ST API and convey them to dashboards for user observation.

Web Video Map Tracks (WebVMT) [https://w3c.github.io/sdw/proposals/geotagging/webvmt/] is an open web format based on JavaScript Object Notation (JSON) and W3C Web Video Text Tracks (WebVTT). WebVMT offers some similarities to its OGC counterparts for moving objects and sensors with mapping to MISB 0903 VMTI metadata and the additional benefit of encapsulation to support any format that can be encoded as a json object. Generic decoders can identify data by type and seamless integration with HTML DataCue enables metadata access in web browsers. The DataCue API enables web pages to associate arbitrary timed data with audio or video media resources, and for exposing timed data from media resources to web pages.

2.2. Business value

This testbed work contributed business value by providing a semantic model which integrates MISB, National Imagery Transmission Format (NITF), Sensor Web Enablement (SWE), Semantic Sensor Network (SSN) ontology, Moving Features, STANAG 4676, SensorThings, WebVMT, and other sensor models.

2.3. Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Contacts

Name	Organization	Role
Emeric Beaufays	Hexagon	Editor
Frederic Houbie	Hexagon	Editor
C. J. Stanbridge	Compusult	Contributor
Rob Smith	Away Team Software	Contributor

2.4. Foreword

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

Chapter 3. References

The following normative documents are referenced in this document.

MISB: MISB ST0903.5, Video Moving Target Indicator and Track Metadata (2020) [https://gwg.nga.mil/misb/docs/standards/ST0903.5.pdf]

MISB: MISB ST0601.15, UAS Datalink Local Set (2019) [https://gwg.nga.mil/misb/docs/standards/ST0601.15.pdf]

OGC: OGC 18-075, OGC® Moving Features Encoding Part I: XML Core (2019) [++http://docs.opengeospatial.org/is/18-075/18-075.html]

OGC: OGC 16-140r1, OGC® Moving Features Encoding Extension – JSON (2020) [++https://docs.opengeospatial.org/bp/16-140r1/16-140r1.html]

OGC: OGC 10-025r1, Observations and Measurements - XML Implementation 2.0 (2011) [++http://portal.opengeospatial.org/files/?artifact_id=41510]

OGC: OGC 12-000, OGC® SensorML: Model and XML Encoding Standard (2014) [++https://portal.opengeospatial.org/files/?artifact_id=55939]

OGC: OGC 15-078r6, SensorThings API Part 1: Sensing (2016) [++http://docs.opengeospatial.org/is/15-078r6/15-078r6.html]

Chapter 4. Terms and definitions

- **FMV|Full-Motion Video**

  high-fidelity digitally encoded video, defined as that whose data is stored at a rate of 25 or more frames per second

- **MF|MovingFeatures**

  OGC API for defining the paths of moving points, lines, and solid geometrical entities as well as the static and dynamic properties of sensors that are located on those moving objects. MovingFeatures data can be expressed as either XML or JSON.

- **MISB|Motion Imagery Standards Board**

  agency established by directive of the United States Department of Defense "to formulate, review, and recommend standards for Motion Imagery"

4.1. Abbreviated terms

API	Application Programming Interface
BER	Basic Encoding Rules
BER-OID	Basic Encoding Rules, Object IDentifiers
ER	Engineering Report
FMV	Full-Motion Video
FOI	Feature of Interest
GML	Geography Markup Language
HTML	HyperText Markup Language
IMAPB	[floating-point to] Integer MAPping using starting point B (defined by MISB 1201)
I/O	Input/Output
IoT	Internet of Things
ISO	International Organization for Standardization
JSIL	Joint System Integration Laboratory (United States Department of Defense)

JSON	JavaScript Object Notation
JSP, .jsp	Jakarta Server Pages (formerly JavaServer Pages)
KLV	Key-Length-Value
LDS	Local Data Set
LS	Local Set
MF	MovingFeatures
MIME	Multipurpose Internet Mail Extension
MISB	Motion Imagery Standards Board
MISP	Motion Imagery Standards Profile
MPEG-2	Moving Picture Experts Group standard 2
MQTT	Message Queuing Telemetry Transport
NITF	National Imagery Transmission Format
OGC	Open Geospatial Consortium
O&M	Observations and Measurements
PAT	Program Association Table
PES	Program Elementary Stream
PMT	Program Map Table
POC	Proof of Concept
REST	REpresentational State Transfer
SDO	Standards Development Organization
SensorML	Sensor Model Language
SMPTE	Society of Motion Picture and Television Engineers

SSN	Semantic Sensor Network
STA	SensorThings API
STANAG	STANdardization AGreement
SWE	Sensor Web Enablement
TCP	Transmission Control Protocol
TS	Transport Stream
UAS	Unmanned Air System
UDP	User Datagram Protocol
UDS	Universal Data Set
UI	User Interface
UML	Unified Modeling Language
UOM, uom	Unit of Measure
URI	Uniform Resource Identifier
VMTI	Video Moving Target Indicator
WebVMT	Web Video Map Tracks
XML	eXtensible Markup Language

Chapter 5. Description of standards

5.1. Overview

This section provides a brief description of the standards that are used and evaluated both as input and output to the proof of concept.

5.2. STANAG 4609

STANAG 4609 [https://gwg.nga.mil/misb/docs/nato_docs/STANAG_4609_Ed3.pdf] describes an exchange format for motion imagery. It is the official format for motion imagery (video data, image sequences, FMV - full motion videos) exchange within the NATO nations. Motion imagery is defined by MISB to be video of at least 1 Hz image frequency together with metadata. STANAG 4609 describes the encoding of the video and the metadata (geographical data) for different usages. This includes the supported video codecs, bit rates, frame rates, container formats, metadata content, metadata encoding and hardware to distribute the motion imagery.

The standards which make television and cable networks possible are established and maintained by the Society of Motion Picture and Television Engineers (SMPTE).

5.3. SMPTE ST 336

SMPTE ST 336 [https://ieeexplore.ieee.org/document/8019807] defines a byte-level data encoding protocol for which can be multiplexed with a video stream. Synchronization between the key-value pairs and the associated video frames is maintained using the same mechanism as is used to synchronize the audio and video streams. SMPTE also defines a small set of Keys in SMPTE ST 335.

The Motion Imagery Standards Board (MISB) has built on those standards to address additional requirements identified by the Defense and Intelligence communities. Those standards are codified in the Motion Imagery Standards Profile (MISP) and STANAG 4609. The MISB standards most relevant to this effort are MISB 0601 and MISB 0903. MISB 0601 defines the basic set of keys for use by UAS systems. MISB 0903 defines additional keys for Video Moving Target Indicators (VMTI). Moving Target Indicators are reports on detections of objects in a FMV frame which appear to be moving, along with any additional descriptive information that the detector can provide.

5.4. OGC Moving Features

This OGC Standard [https://www.opengeospatial.org/standards/movingfeatures] specifies standard encoding representations of movement of geographic features. The primary use case is information exchange. The encodings specified in the OGC® Moving Features suite of standards conform to the ISO 19141:2008, Geographic information – Schema for moving features standard.

A feature is an abstraction of a real-world phenomenon. A geographic feature if it is associated with a location relative to the Earth is a geographic feature. ISO 19141:2008 represents moving features by defining a geometry for a feature that moves as a rigid body. This allows moving and stationary features to be analyzed and exploited using the same algorithms and software.

5.5. OGC SensorML

Sensor Model Language (SensorML) [https://portal.opengeospatial.org/files/?artifact_id=55939] provides a robust and semantically tied means of defining processes and processing components associated with the measurement and post-measurement transformation of observations. This includes sensors and actuators as well as computational processes applied pre- and post-measurement. The main objective is to enable interoperability, first at the syntactic level and later at the semantic level (by using ontologies and semantic mediation). This is so sensors and processes can be better understood by machines, utilized automatically in complex workflows, and easily shared between intelligent sensor web nodes.

5.6. OGC Observations and Measurements

The Observations and Measurements (O&M) [https://www.opengeospatial.org/standards/om] Standard defines a conceptual schema for sensor observations, and sampling features produces when making observations. These provide models for the exchange of information describing observation acts and their results, both within and between different scientific and technical communities. Observations commonly involve sampling of an ultimate feature of interest. O&M defines a common set of sampling feature types classified primarily by topological dimension, as well as samples for ex-situ observations. The schema includes relationships between sampling features (sub-sampling, derived samples).

5.7. OGC SensorThings API

The SensorThings API (STA) [https://www.ogc.org/standards/sensorthings] OGC standard defines the interconnection and communication of Internet of Things (IoT) devices. It is comprised of a sensing part, which allows data observed by multiple IoT sources to be conveyed using a standard JSON-based format, and a tasking part, which allows events to be activated based on the values of these observations. All observations, metadata, and messages generated in this API can be forwarded to MQTT or to RESTful endpoints for later use by clients.

Chapter 6. Tracking Algorithm

6.1. Overview

This section provides a general description of a Multiple Object Tracking algorithm that computes tracks from the individual MISB 0903 Video Moving Target Indicators (VMTI). The algorithm uses a global nearest neighbor approach that takes into account a speed and bearing estimate of the moving object. This algorithm has the advantage of being able to run on a video stream, rather than needing to collect all the detections first, as well as being fast enough to run in real-time.

MISB 0903 concentrates on video tracking as a VMTI is essentially a location and bounding box. Tracking through Lidar sensors is not prohibited. The standard is not meant for radar tracking. The standard for Radar moving target detections is STANAG 4607. The algorithm makes the assumption that a single object is described by a single target and that a single target describes a single object (Point Object Tracking). Missing detections and false positives are dealt with as corner cases.

6.1.1. Initialization

The first phase of the algorithm is the initialization or "seeding" of tracks where new detections are used to instantiate tracks.

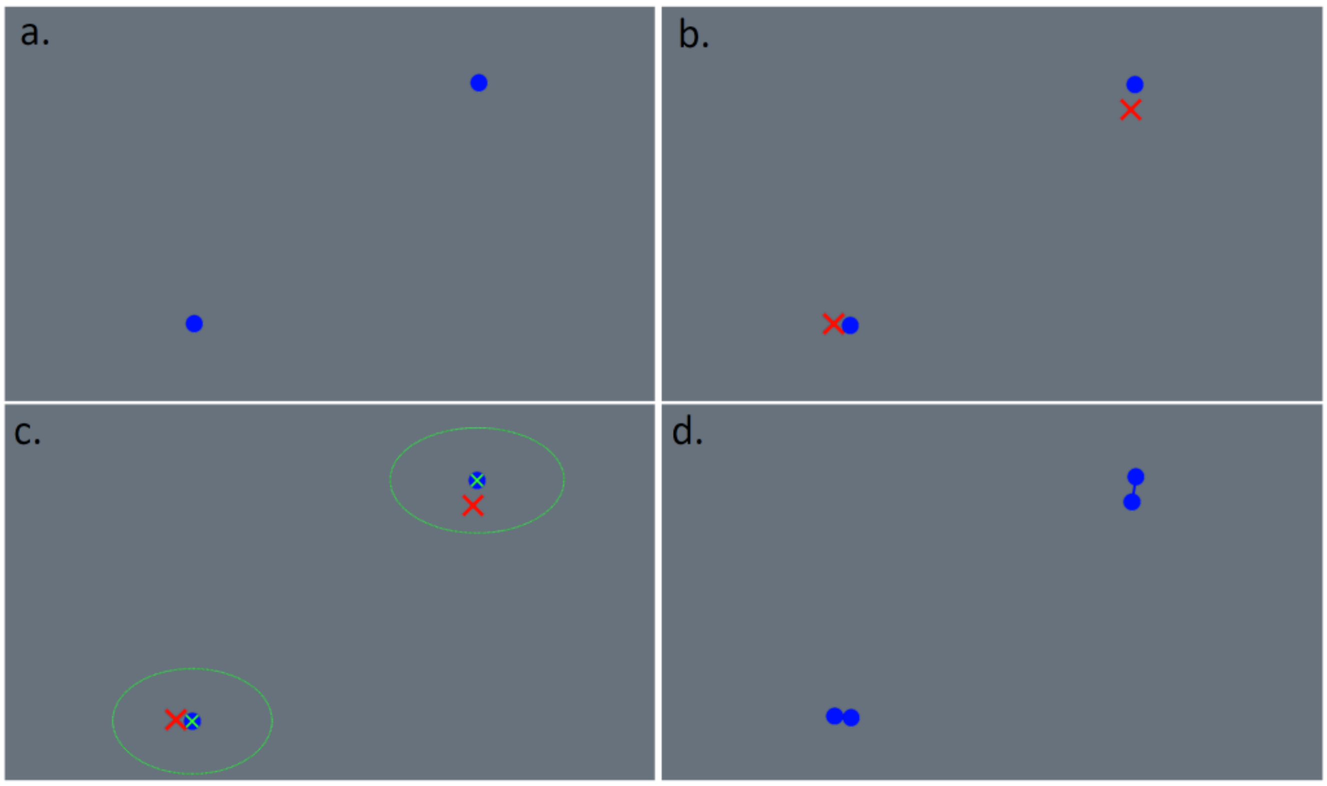


Figure 2. Initialization of new tracks.

VMTIs from the first frame or VMTIs that cannot be matched to an existing track each begin a new track. The VMTIs from the following frames are loaded. Each track looks for VMTIs within a given radius of the last known position. The new VMTIs that are matched to existing tracks are used to extend them. VMTIs that don’t match to any existing track start a new track

6.1.2. Expansion

Existing tracks are expanded by matching detections to the track's predicted location based on speed and bearing.

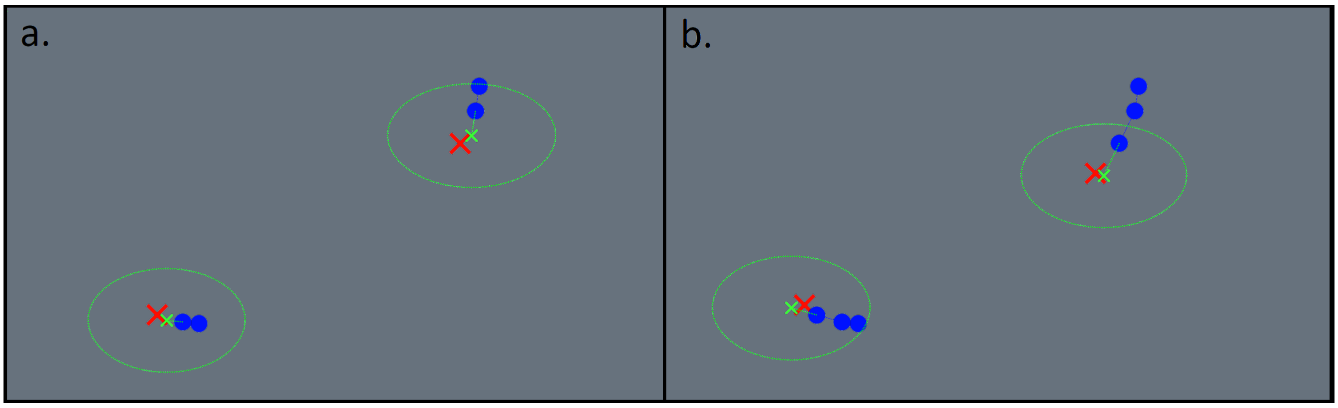


Figure 3. Expansion of existing tracks.

Based on the direction and speed of the moving object, their predicted location is computed. The algorithm looks for the next track location within a given radius of the predicted location and expands the track. If no VMTI is found within the radius, the track is finalized after some time.

6.1.3. Resolving ambiguous situations

During track expansion, detections may come close to the predicted location of several tracks.

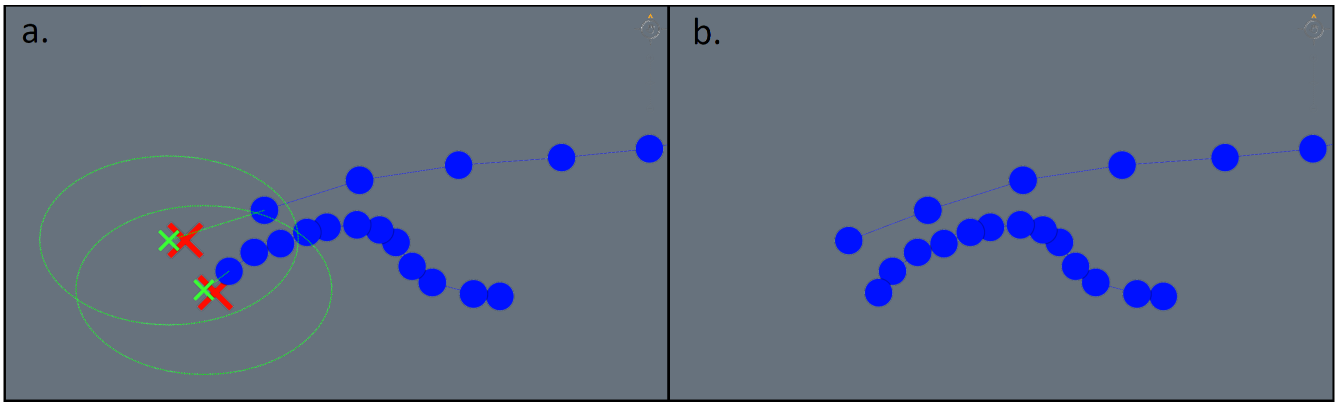


Figure 4. Resolving ambiguities.

In the situation above, both VMTIs fall into the radius of the predicted location of two tracks. This is resolved by matching the VMTIs to the predicted locations in order to minimize the overall predicted to true location distance.

6.1.4. Missing detections

Detections may be missing because the target was obscured, because the camera looks away or because the detection algorithm fails. Missing detections may be accounted for by keeping tracks alive for a certain amount of time in the hope of matching it with a detection at a later time.

This solution works well as long as the tracked object keeps the same speed and direction. Increasing the radius within which to search for matching detections helps to account for acceleration and heading changes but runs the risk of mixing up the tracks. This solution works well once the search radius and keep-alive time are optimized for a certain type of moving object.

6.1.5. False positives

False positives are detections that do not actually correspond to a moving object. They may be due to many things such as video glitches that the detection algorithm interprets as an actual object. Removing outliers is tempting but an outlier is not necessarily a false detection and it may help expand tracks with few detections. Another approach is to filter out unusable tracks after the tracking algorithm is done. These are tracks made up of just a few detections which could not be matched with any existing tracks and therefore generated tracks of their own.

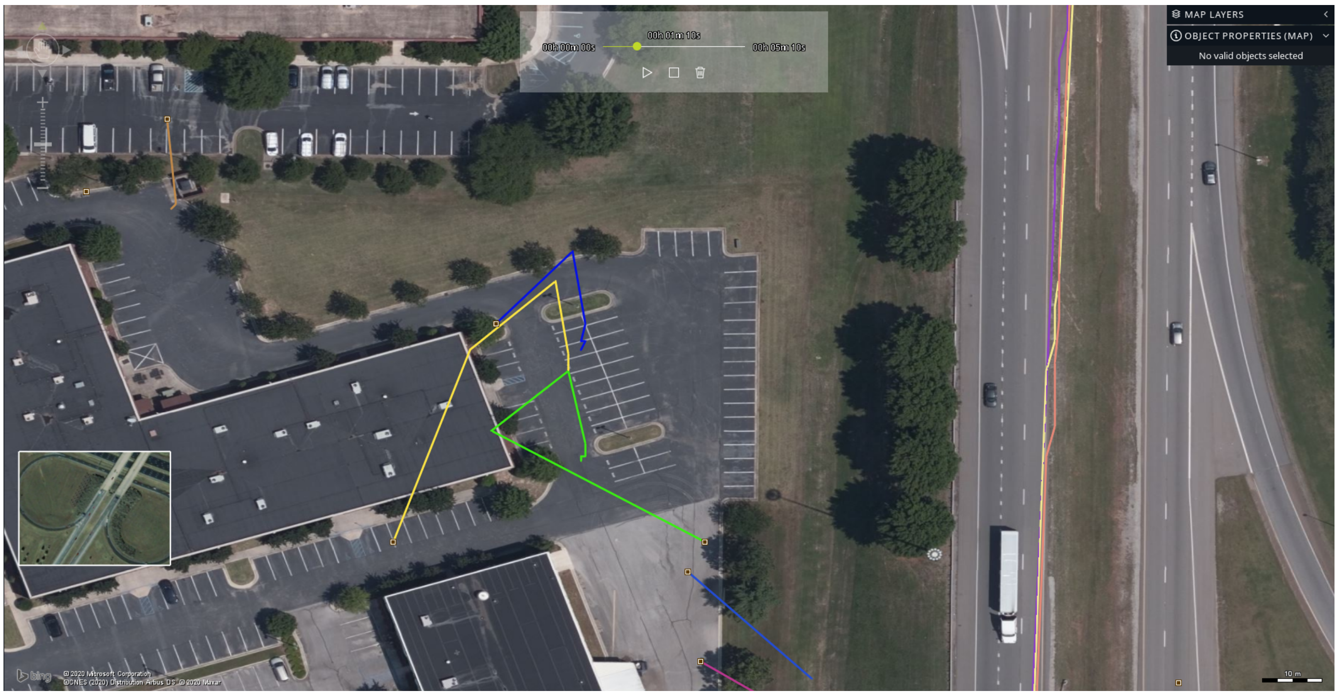


Figure 5. False positive tracks.

The image above shows the long tracks on the right that follow the roads and the short jittery tracks made up of false positives on the left.

6.1.6. Smoothing and interpolation

The raw tracks can be jittery and lack resolution. A simple method to smooth and interpolate the tracks can be used. In this implementation, the location of a track at a given time is calculated based on a weighted average of the forces pulling the track to the next known true locations.

6.1.7. Improvements

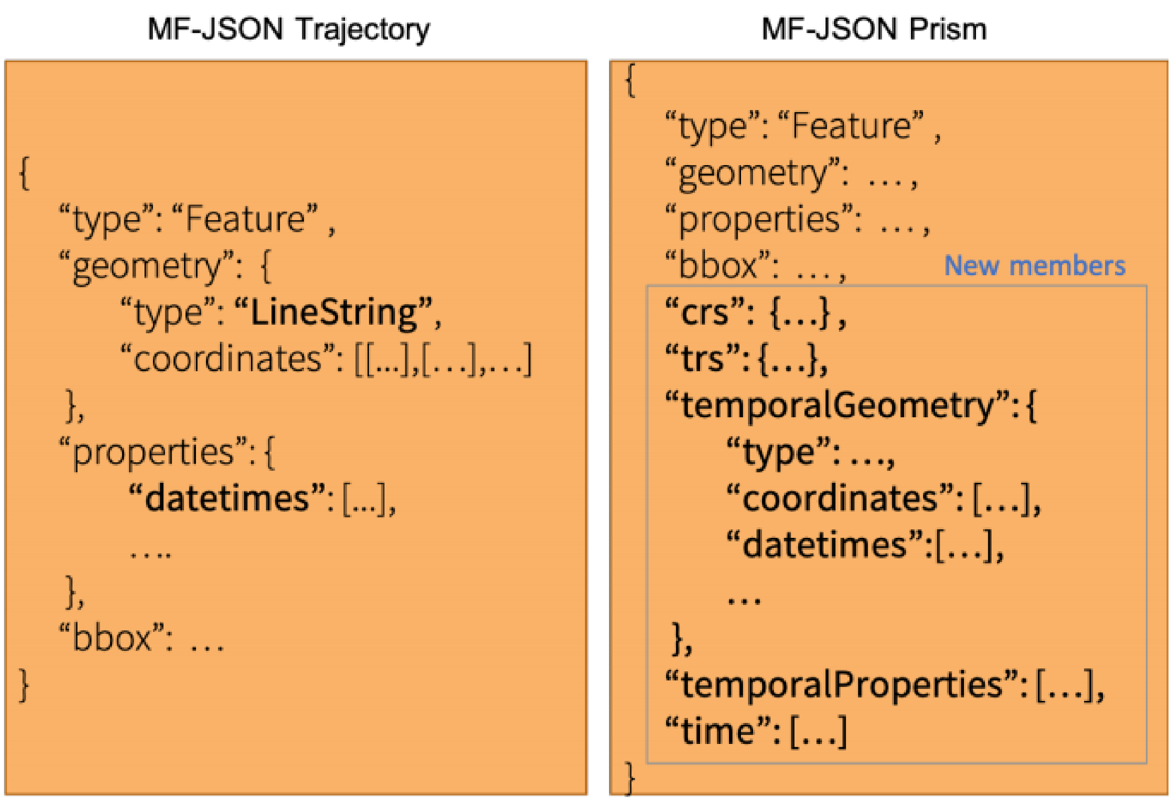
The algorithm can be made probabilistic by considering information such as class, color, intensity and target confidence level that are sometimes present in MISB 0903 VMTIs. This information would weigh-in to resolve some ambiguous situations.

Chapter 7. Encoding to OGC Moving Features

7.1. Overview

This section covers how the elements of the MISB 0903 and the result of the Tracking Algorithm can be mapped to OGC Moving Features.

Moving Features with JSON encoding (OGC 19-045r3) was used to demonstrate the mapping of MISB 0903. Below is a quick recap of the 2 allowed structures. MF_JSON Trajectory can be used for very simple use-cases while the MF-JSON Prism gives more flexibility.



```
{
   "type": "Feature" ,
   "geometry": {
        "type": "LineString",
        "coordinates": [[...],[...],...]
    },
   "properties": {
        "datetimes": [...],
        ....
    },
   "bbox": ...
}
```

```
{
   "type": "Feature" ,
   "geometry": ... ,
   "properties": ... ,
   "bbox": ... ,
   "crs": {...} ,
   "trs": {...},
   "temporalGeometry": {
        "type": ...,
        "coordinates": [...],
        "datetimes": [...],
        ...
    },
   "temporalProperties": [...],
   "time": [...]
}
```

Figure 6. MF_JSON Trajectory vs MF_JSON Prism

7.2. Tracks

The tracking algorithm or alternatively, the tracker information directly encoded in MISB 0903, creates groups of points with time-stamps (tracks). The simplest way to encode this information is to use a **Feature Collection** with a separate **Feature** for each track.

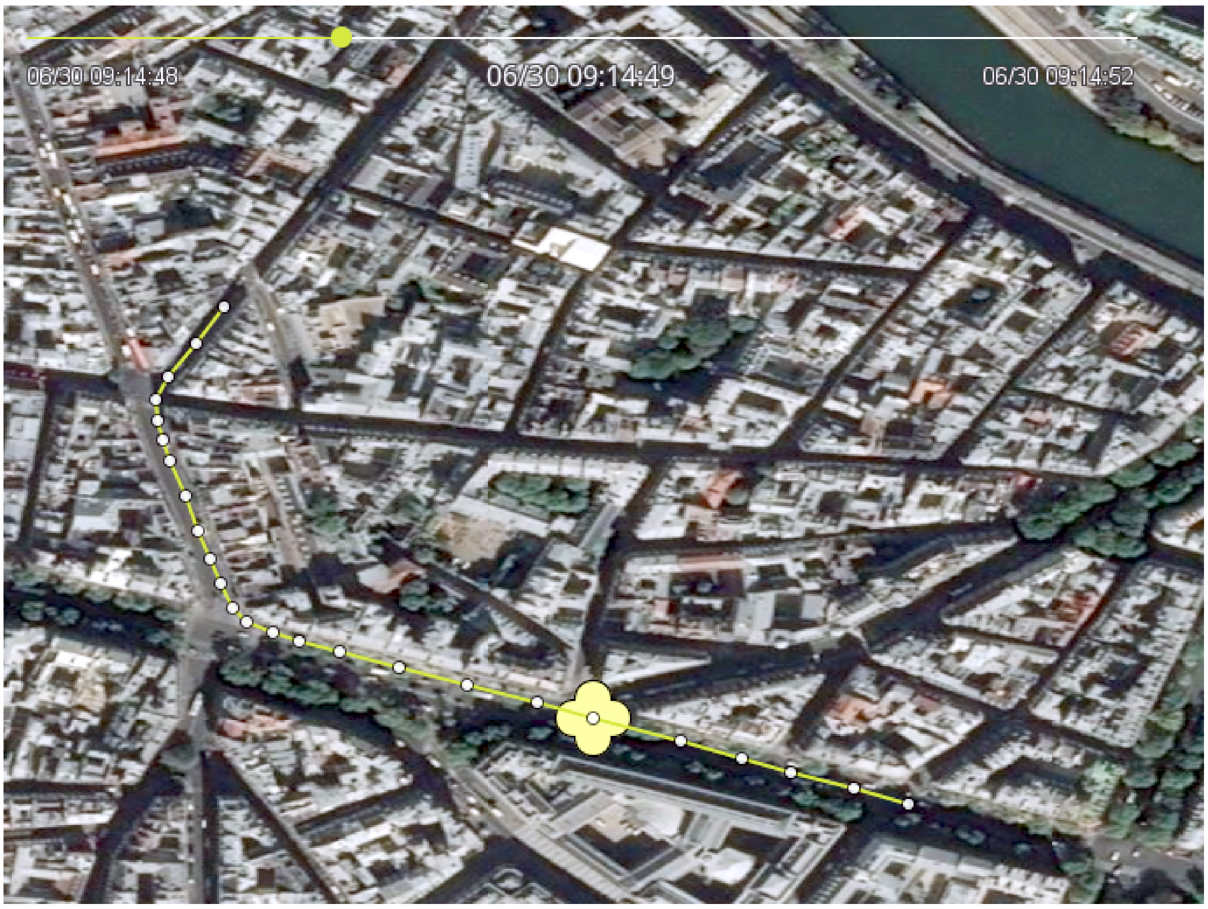


Figure 7. Example OGC Moving Feature (JSON encoding)

Example of a simple OGC Moving Feature (JSON encoding)

```
{
  ...
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "temporalGeometry": {
        "type": "MovingPoint",
        "coordinates": [
          [
            0.3666873337477613,
            0.33452775033244386
          ],
          [
            0.5194375435035997,
            0.14164689242465356
          ]
        ],
        "datetimes": [
          "1970-01-01T00:00:00Z",
          "1970-01-01T00:00:00.001Z"
        ],
        "interpolation": "Linear"
      }
    },
    {
      "type": "Feature",
      "temporalGeometry": {
        "type": "MovingPoint",
        "coordinates": [
          [
            0.7076755252993154,
            0.37470246099128324
          ],
          [
            0.6273214391327322,
            0.3127051674314162
          ]
        ],
        "datetimes": [
          "1970-01-01T00:00:00Z",
          "1970-01-01T00:00:00.001Z"
        ],
        "interpolation": "Linear"
      }
    }
  ]
}
```

7.3. Bounds

OGC Moving Features specifies a predefined field to encode individual track bounds and overall bounds of a dataset in terms of space and time. The VTracker Local Data Set(LDS) may define the bounds of a track but they may also be calculated based on the result of the Tracking Algorithm.

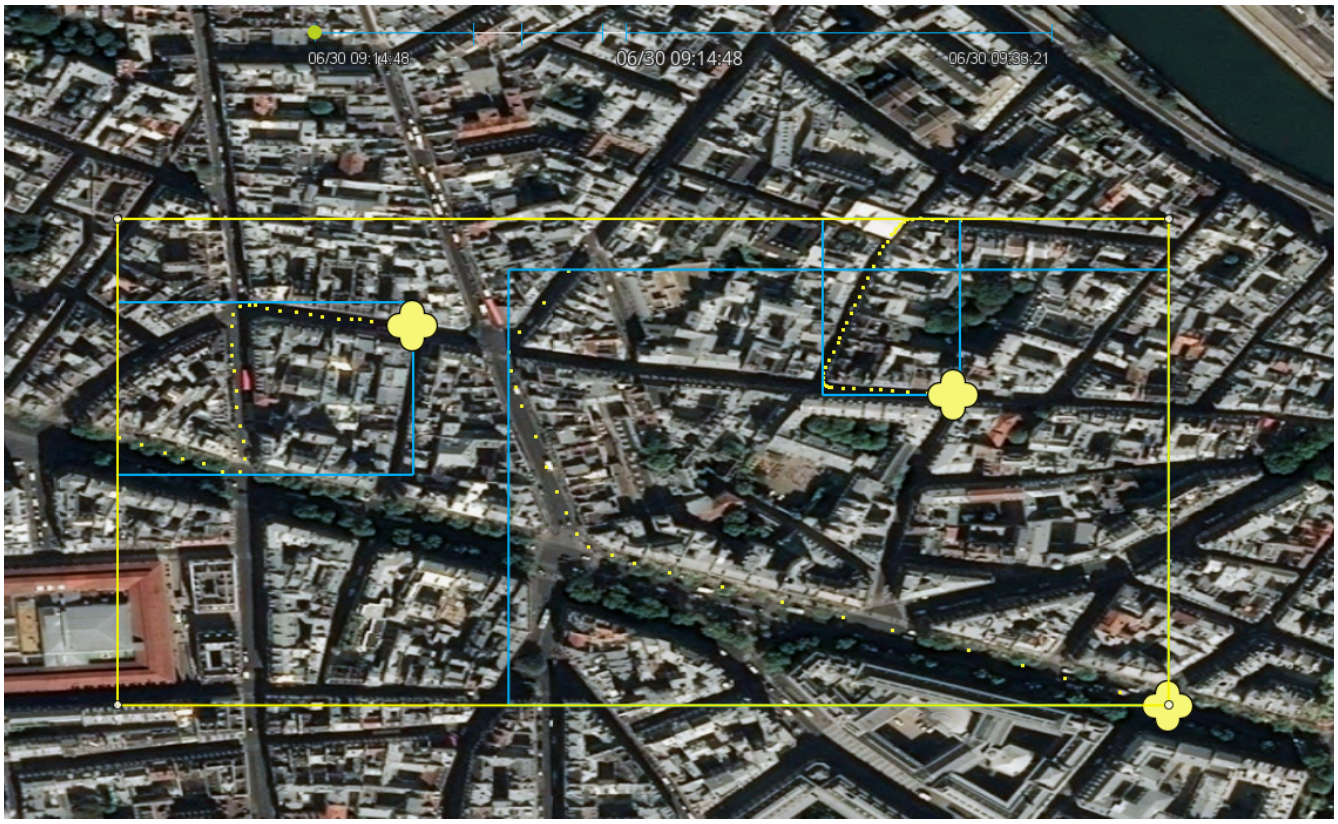


Figure 8. example OGC Moving Feature with bounds

Example OGC Moving Feature with bounds

```
{
  ...
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "temporalGeometry": {
        "type": "MovingPoint",
        ...
      },

      "bbox": [
        0.3666873337477613,
        0.14164689242465356,
        0.5194375435035997,
        0.33452775033244386
      ],
      "time": ["1970-01-01T00:00:00Z","1970-01-01T00:00:00.001Z"]
    },
    {
      "type": "Feature",
      "temporalGeometry": {
        "type": "MovingPoint",
        ...
      },
      "bbox": [
        0.6273214391327322,
        0.3127051674314162,
        0.7076755252993154,
        0.37470246099128324
      ],
      "time": ["1970-01-01T00:00:00Z","1970-01-01T00:00:00.001Z"]
    }
  ],
  "bbox": [
      0.3666873337477613,
      0.14164689242465356,
      0.7076755252993154,
      0.37470246099128324
    ],
    "time": ["1970-01-01T00:00:00Z","1970-01-01T00:00:00.001Z"]
}
```

7.4. Target Bounds, outline and rigid geometry

In addition to the VMTI target location, MISB 0903 may specify a target's bounds. The temporalGeometry can be used to encode this information through a MovingPolygon which is a shape defined by OGC Moving Features. Both the MISB0903 target location and location bounds can be encoded simultaneously in a OGC Moving Features MovingGeometryCollection. See the json example below for more detail.

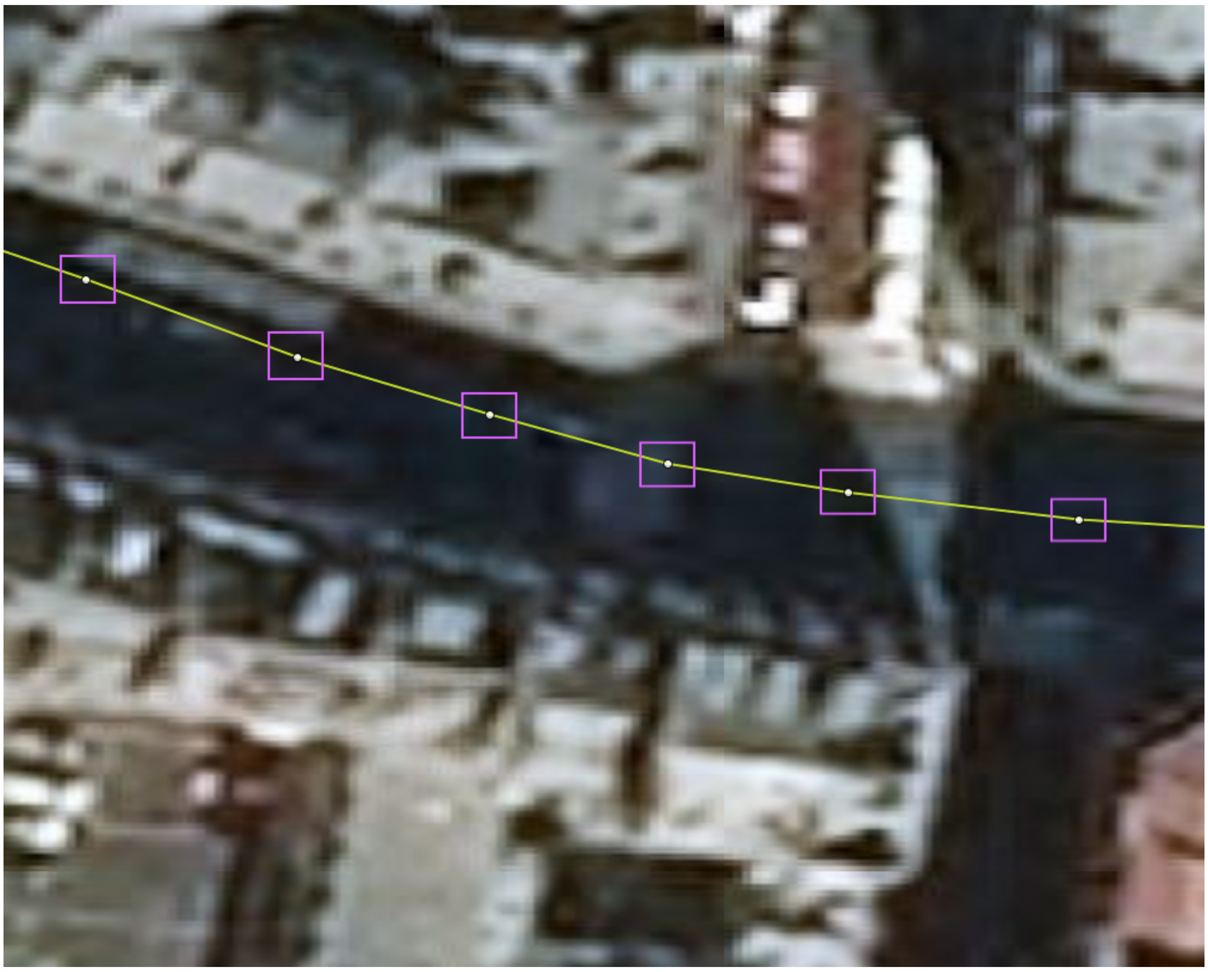


Figure 9. Target Bounds

The encoding of target bounds in MISB 0903 is only 2 dimensional while the target location can have a height component. The **vTargetPack** from MISB 0903 may contain an additional **target Boundary** object which is a collection of **Location** elements (lat/lon/height) therefore allowing, in theory, more complex 3D boundaries. This does not give MISB 0903 the capability to encode a boundary volume.

The XML encoding of OGC Moving Features allows a 1 to 1 mapping of the MISB 0903 bounds and a target Boundary. The latest OGC Moving Features JSON encoding Standard (1.0) only allows for 2D polygons. This means that the height components of the MISB 0903 target Boundary will be lost.

The MISB0903 VMask Local Data Set (LDS) is used to represent the outline of the target which may also be used to encode a 2D outline of the target in OGC Moving Features. Note that for a one to one mapping between MISB 0903 and Moving Features, the rigid geometry of Moving Features is not used but rather encoding the full polygon for each time-stamp.

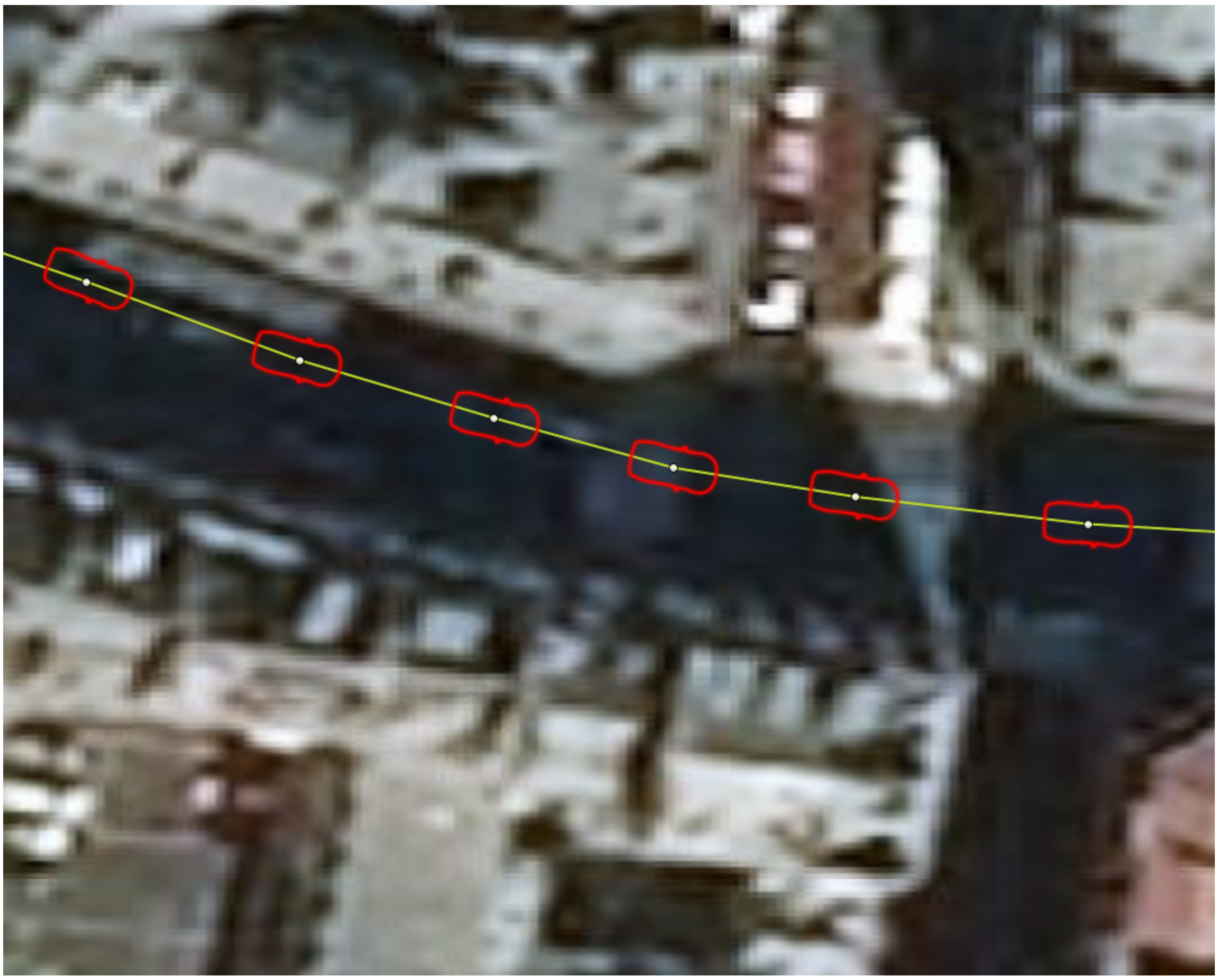


Figure 10. Target outline

A tracked object might be matched with a rigid geometry such as a drone model or a ship’s outline polygon. In such a case, the participants recommend using the **base** member of OGC Moving Features which describes a type and reference to a 3D model such as STL, OBJ, PLY, and glTF. When the base member is present, the Temporal Geometry type is always "MovingPoint" and the **orientations** member must be present. The orientations member is an array of json objects with the same size as the **datetimes** member.it describes the scale and orientation of the model in the **base**.

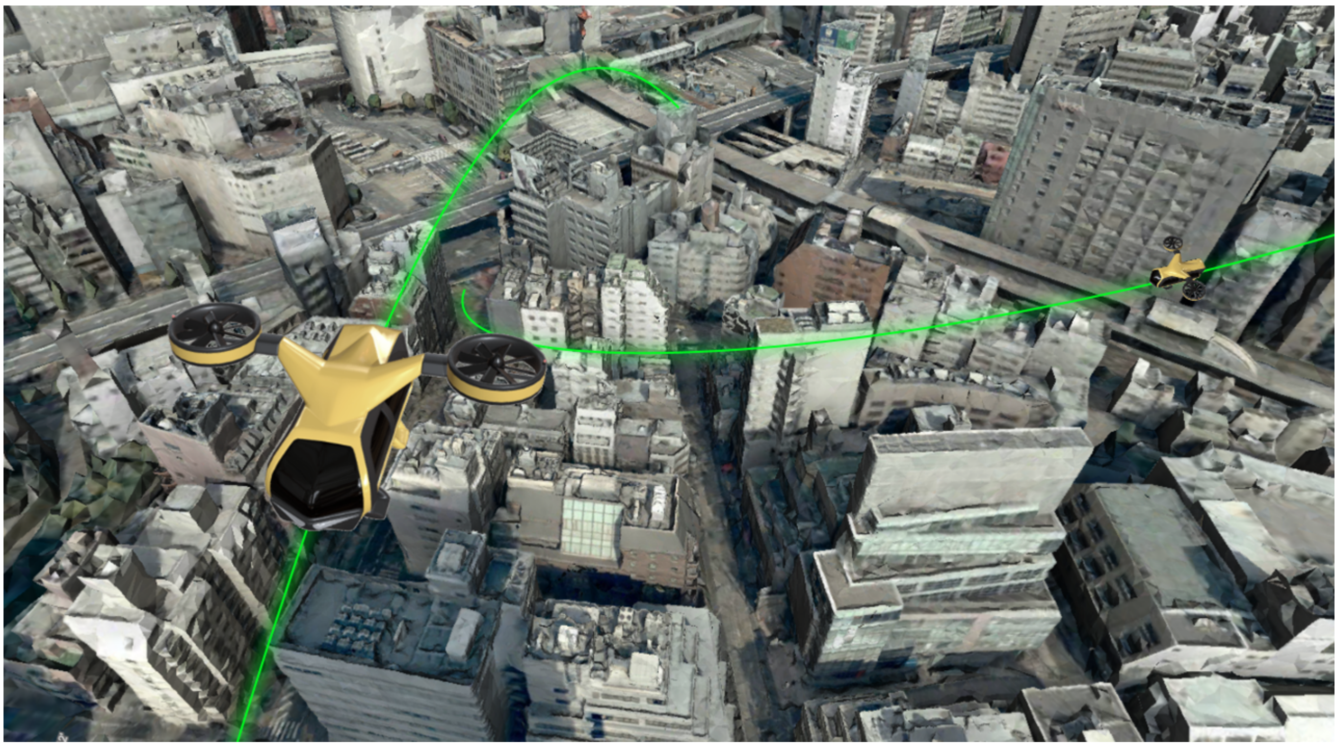


Figure 11. Drone model and path

Example Temporal Geometry with simple target location, target bounds, target outline and rigid shape.

```
{
    ...
    "temporalGeometry": {
      "type": "MovingGeometryCollection",
      "members":[
        {
          "type": "MovingPoint",
          "coordinates": [
            [0.707, 0.375],
            [0.627, 0.313]
          ],
          "datetimes": [ "1970-01-01T00:00:00Z", "1970-01-01T00:00:00.001Z" ],
         "interpolation": "Linear"
        },
        {
          "type": "MovingPolygon",
          "coordinates": [
            [[[0.698,0.365],
              [0.718,0.365],
              [0.718,0.385],
              [0.698,0.385]]],
            [[[0.617,0.303],
              [0.637,0.303],
              [0.637,0.323],
              [0.617,0.323]]]
          ],
          "datetimes": [ "1970-01-01T00:00:00Z", "1970-01-01T00:00:00.001Z" ],
          "interpolation": "Linear"
        },
        {
          "type": "MovingPolygon",
          "coordinates": [
            [[[0.698,0.365],
              [0.718,0.365],
              [0.725,0.375],
              [0.718,0.385],
              [0.698,0.385]]],
            [[[0.617,0.303],
              [0.637,0.303],
              [0.644,0.313],
              [0.637,0.323],
              [0.617,0.323]]]
          ],
          "datetimes": [ "1970-01-01T00:00:00Z", "1970-01-01T00:00:00.001Z" ],
          "interpolation": "Linear"
        },
        {
          "base": {"type": "OBJ", "href":  "./truck.obj"},
          "type": "MovingPoint",
          "coordinates": [
            [0.707, 0.375],
            [0.627, 0.313]
          ],
          "orientations": [
            {
              "scales":[1,1,1],
              "angles":[0,0,0.25]
            },
            {
              "scales":[1,1,1],
              "angles":[0,0,0.21]
            }
          ],
          "datetimes": [ "1970-01-01T00:00:00Z", "1970-01-01T00:00:00.001Z" ],
          "interpolation": "Linear"
        }
      ]
    },
    ...
}
```

7.5. Temporal properties

Several properties present in MISB 0903 can be encoded as **Temporal Properties** in OGC Moving Features. These are properties specific to a single VMTI target that change over time.

1. Target color

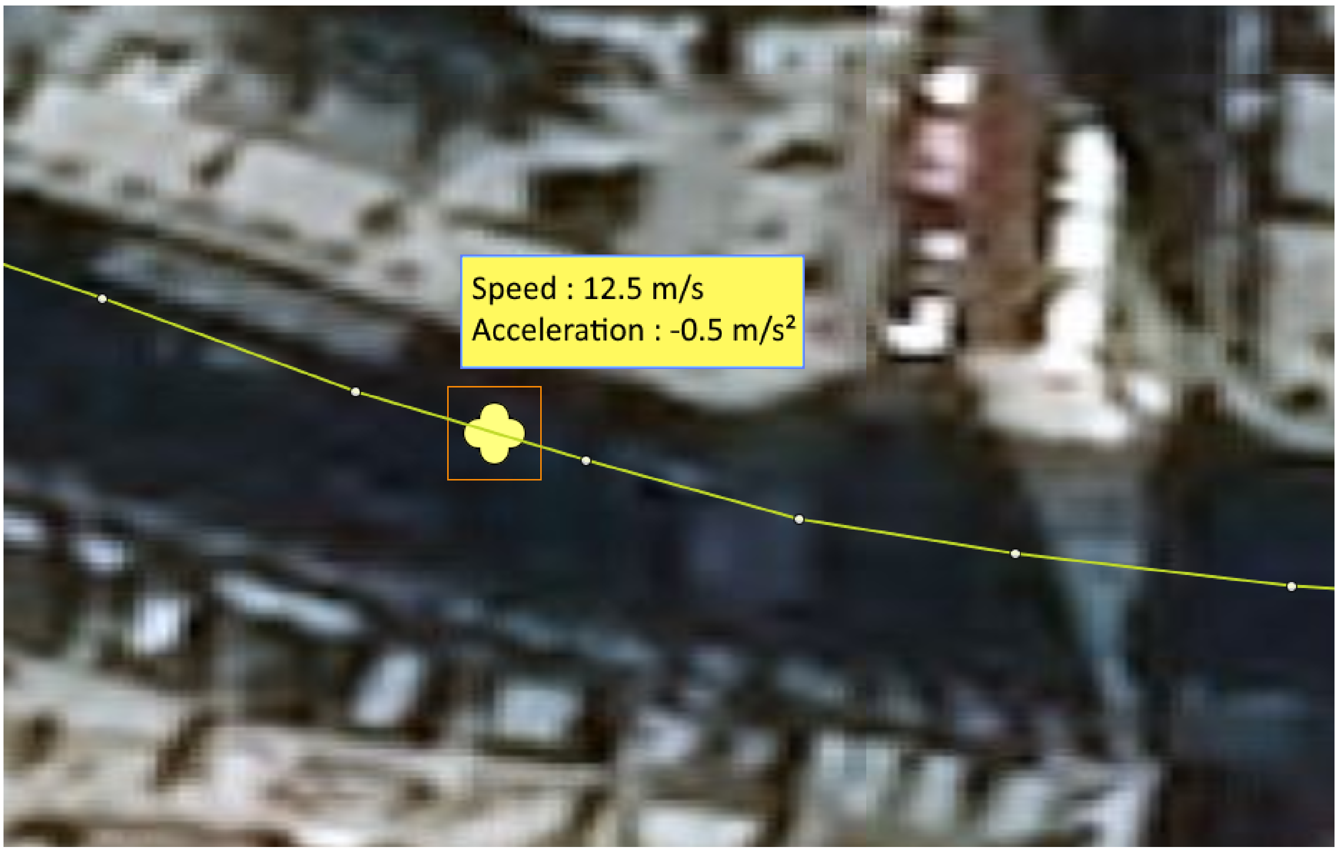
2. Target intensity
3. Target confidence Level
4. Target Priority

Example Temporal properties

```
{
    ...,
    "temporalProperties": [
      {
        "datetimes": ["1970-01-01T00:00:00Z", "1970-01-01T00:00:00.001Z"],
        "color": {
          "uom": "RGB",
          "values": ["16711808", "16711808"],
          "interpolation": "Discrete"
        },
        "confidence_Level": {
          "uom": "%",
          "values": [100, 75],
          "interpolation": "Stepwise"
        }
      }
    ],
    ...
}
```

7.5.1. Speed and Acceleration

The speed and acceleration of the vehicle may be provided inside the VTracker LDS packages or they may be calculated during tracking. The result may be encoded as temporal properties with stepwise, linear or spline interpolation



7.5.2. Class

The VObject LDS uses an ontology to describe the set of allowed classes or type values encoded as Web Ontology Language (OWL). The MISB 0903 element is made up of a URI link to the ontology and a class. Once a parser is available, encoding the temporal property may be trivial. However, the class may also describe enumerations, unions, intersections, complements and property restrictions. Rather than forcing this information into an ill-suited format, the class may be encoded as full text in the temporal property's values while the URI to the ontology can be encoded as the property's Unit of Measure (uom).

Example ontology encoding

```
"temporalProperties": [
      {
        "datetimes": ["2017-03-13T01:00:00Z", "2017-03-13T02:00:00Z"],
        "class": {
          "uom": "https://example.com/ontology",
          "values": ["<owl:Class rdf:ID=\"Car\"/>", "<owl:Class rdf:ID=\"Car\"/>"],
          "interpolation": "Stepwise",
        }
      }
    ],
```

While the use of a json **Property** rather than a **Temporal Property** for this field would be better suited in many cases, this approach could not account for a one to one mapping between MISB 0903 and OGC Moving Features.

7.5.3. VFeature Local Data Set

MISB 0903 adopted the OGC Geography Markup Language (GML) and the VFeature LDS is based on the OGC Observations and Measurements (O&M) Encoding Standard and related schemas.

The VFeature LDS is made up of a URI to a schema and a Feature. A Feature is represented as an OGC GML document structured according to the schema. The GML feature contains one or more values observed for a VMTI.

While the GML data may be parsed, nested data will be difficult to encode in the OGC Moving Features format because the values of a temporal property cannot be nested. According to the Moving Features standard, a temporal property must be a flat array of Strings or numbers.

While the **properties** field (rather than temporal properties) allows nested values, the temporal aspect of the data would have to be implemented as a parallel mechanism not described by the schema.

The data can instead be encoded in full text as values and the URI to the schema can be encoded as the property's uom.

example GML Features

```
"temporalProperties": [
      {
        "datetimes": ["2017-03-13T01:00:00Z", "2017-03-13T02:00:00Z"],
        "class": {
          "uom": "http://schemas.opengis.net/gml/3.2.1/gml.xsd",
          "values": [GML as text, GML as text],
          "interpolation": "Stepwise",
        }
      }
    ],
```

7.5.4. VTracker Local Data Set

The VTracker LDS, if present and populated, groups detections into tracks. This achieves the same task as the Tracking Algorithm although the technique used and results may be different. The VTracker LDS packages may provide Speed and Acceleration information as well as a tracker specific confidence value.

Other values from this LDS concern information that was already covered.

7.5.5. VChip Local Data Set

The VChip LDS defines an image or a link to an image with the corresponding Multipurpose Internet Mail Extension (MIME) type. If present, the information can be encoded as a temporal property. For the case where actual images are provided, they may be encoded as byte strings.

Example "chip" image encoding

```
"temporalProperties": [
      {
        "datetimes": ["2017-03-13T01:00:00Z", "2017-03-13T02:00:00Z"],
        "chip_images": {
          "uom": "jpeg",
          "values": ["https://example.com/ontolog\image1.jpg",
"https://example.com/ontolog\image2.jpg"],
          "interpolation": "Stepwise",
        }
      }
    ],
```

7.6. Global Temporal properties

Global Temporal properties describe an entire frame rather than individual tracks. These are properties such as reported targets per frame, frame number or frame size in pixels. The JSON encoding of Moving Features objects cannot accommodate these properties without repeating the information in every single Moving Feature object because the json Feature Collection lacks a temporal properties field and even a properties field at all.

7.7. Static Properties

Until now all the properties have been encoded as temporal properties to ensure a one to one mapping between MISB 0903 and OGC Moving Features. For example, it is unlikely that the class of a tracked object suddenly changes from "car" to "boat" but since the classification is likely automatic, this may well happen in MISB 0903.

However, an extra processing step can generate global properties for a particular track. In this case, the properties field of a moving feature can encode the information.

Example moving feature with global properties

```
{
    "type": "Feature",
    "temporalGeometry": {...},
    "temporalProperties": [...],
    "bbox": [...],
    "properties": {
        "class": "pirate ship"
    }
}
```

Chapter 8. WebVMT: Exporting MISB Metadata for the Web

8.1. Introduction To WebVMT

Web Video Map Tracks (WebVMT) is an open format designed to synchronize location and sensor data with video for web based applications. This common format makes it easier to share, search and present metadata with video footage online. The use of WebVMT also helps to break down the barriers between different devices, such as dashcams, drones, body-worn video and smartphones thereby enhancing accessibility and utilization. Further details on WebVMT can be found at the WebVMT website [https://webvmt.org/] which includes a non-technical overview, blog, and technical demos.

Mapping to WebVMT enables MISB metadata to be accessed on the web by developers using JavaScript in a web browser, by web crawlers for indexing, and by search engine users to find relevant data quickly. Data can be retrieved independently of the web client platform, machine hardware, video format, and sensor device.

8.2. Exporting MISB From MPEG-2

The initial Testbed-16 FMV to Moving Features development focused on parsing MISB metadata from MPEG-2 files. This highlighted the value of out-of-band metadata (in a separate file) in comparison to in-band metadata embedded within the media file. Before any attempt was made to read the actual metadata content significant effort was required to navigate the media format in order to parse the relevant sections and extract the raw data. Exporting metadata to an out-of-band format, such as WebVMT, eliminates this time-consuming task and allows developers to concentrate on parsing the metadata rather than on the non-trivial chore of accessing it.

8.2.1. Exported Metadata On The Web

Parsing the MPEG-2 Transport Stream (TS) sample files used in this project demonstrated the complexity of the process. The steps to access metadata content were:

1. Identify and parse the packet containing the Program Association Table (PAT).
2. From that information, identify and parse the packet containing the Program Map Table (PMT).
3. From that information, identify, demultiplex, and parse program elementary stream (PES) packets containing metadata.
4. From that information, extract and assemble metadata segments into packets ready for parsing.
5. For each metadata packet, identify and parse metadata Access Unit (AU) cells containing metadata content.

Established World Wide Web Consortium (W3C) formats such as Timed Text Markup Language (TTML) and Web Video Text Tracks (WebVTT) synchronize data with video using out-of-band metadata which is stored in a discrete linked file. This approach makes access faster and simpler by eliminating the complex video stream structure and offering a common format that is not dependent on the video codec or container format.

If video metadata were exported to a common out-of-band format, such as WebVMT, metadata content could be accessed more quickly and efficiently. This reduces development time and file bandwidth and also allows metadata parsing to be agnostic of the media encoding format. WebVMT has the added advantage of being designed for the web so metadata can be easily exposed through HTML DataCue [https://www.w3.org/TR/html51/semantics-embedded-content.html#datacue-datacue]. This enables access through web browser APIs, such as JavaScript, and by web crawlers so users can rapidly identify key sections of metadata and video footage using online search engines.

8.3. Mapping MISB To WebVMT

MISB metadata can be mapped to WebVMT in different ways depending on observation context and which optional MISB tags are present.

8.3.1. UAS Datalink LS Mapping

The following gives a general guide for mapping UAS Datalink LS packets.

Table 1. MISB UAS Datalink LS (0601) Mapping

Tag	Description	WebVMT Mapping
11	Source sensor	Path identifier
13	Sensor latitude	Path segment
14	Sensor longitude	Path segment
16	Sensor horizontal field of view	Map zoom [2]
17	Sensor vertical field of view	Map zoom [2]
21	Slant range	Map zoom [2]
22	Target width	Map zoom [1]
23	Frame center latitude	Map center
24	Frame center longitude	Map center
26	Upper left corner offset latitude	Map zoom [3]
27	Upper left corner offset longitude	Map zoom [3]
28	Upper right corner offset latitude	Map zoom [3]
29	Upper right corner offset longitude	Map zoom [3]
30	Lower right corner offset latitude	Map zoom [3]
31	Lower right corner offset longitude	Map zoom [3]

Tag	Description	WebVMT Mapping
32	Lower left corner offset latitude	Map zoom [3]
33	Lower left corner offset longitude	Map zoom [3]

In general, mapping the Frame Center (MISB 0601, tags 23 & 24) to the WebVMT map center is a sensible choice. There are three possible mappings to WebVMT map zoom using MISB target width [1], field of view and range [2], and corner offsets [3]. Object trajectories, such as the Sensor Location (MISB 0601, tags 13 & 14), can be mapped to a WebVMT path and in certain use cases mapping the Frame Center to a WebVMT path can be insightful.

8.3.2. Video Moving Target Indicator LS Mapping

Any set of Target Locations (MISB 0903, VTarget Pack tags 10, 11 & 17) which form an object trajectory can be mapped to a WebVMT path, and multiple paths can be discriminated by using the WebVMT path identifier. Care should be taken not to overload the user when using WebVMT data with multiple paths, either by limiting the number of paths shown or by limiting the length of the path history displayed.

8.4. Web Browser Demos

MISB metadata was exported from the sample MPEG-2 Transport Stream files to WebVMT using the C++ export utility code which was modified to create WebVMT paths from different MISB tags depending on the use case, as outlined above.

Exported data can be displayed at the WebVMT website [https://webvmt.org] by dragging and dropping a pair of video and VMT files into the Mobile Demo which can be displayed by most modern web browsers.

8.4.1. Drone Demos

These videos show footage captured from a commercial Unmanned Aerial Vehicle (UAV), colloquially referred to as a drone, with a camera that can face directly downward or look forward in the direction of travel.

In both cases, Frame Center (MISB 0601, tags 23 & 24) and Target Width (MISB 0601, tag 22) control the map center and zoom respectively, and Sensor Location (MISB 0601, tags 13 & 14) is mapped to a WebVMT path shown as a blue line in the figures below.

When the camera is facing downward, the front of the path remains aligned with the map center and accurately follows the path of the river, as expected, which demonstrates that the location details are correctly exported to WebVMT.

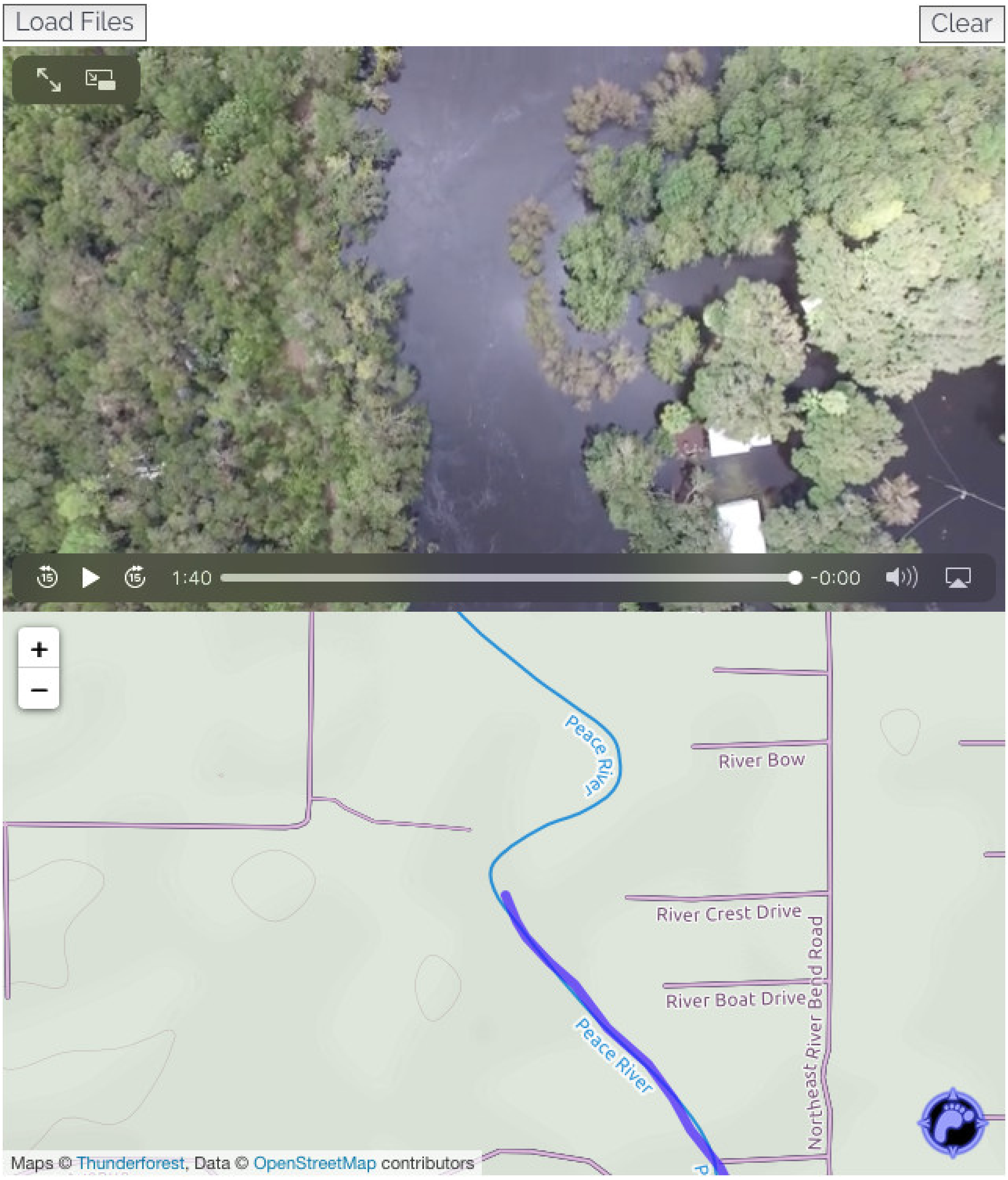


Figure 12. Drone With Downward Facing Camera

When the camera is facing forward, the front of the path lags behind the map center and is correctly positioned between the two parallel roads seen in the footage. Once again, this is consistent with the video content and demonstrates that location is accurately exported.

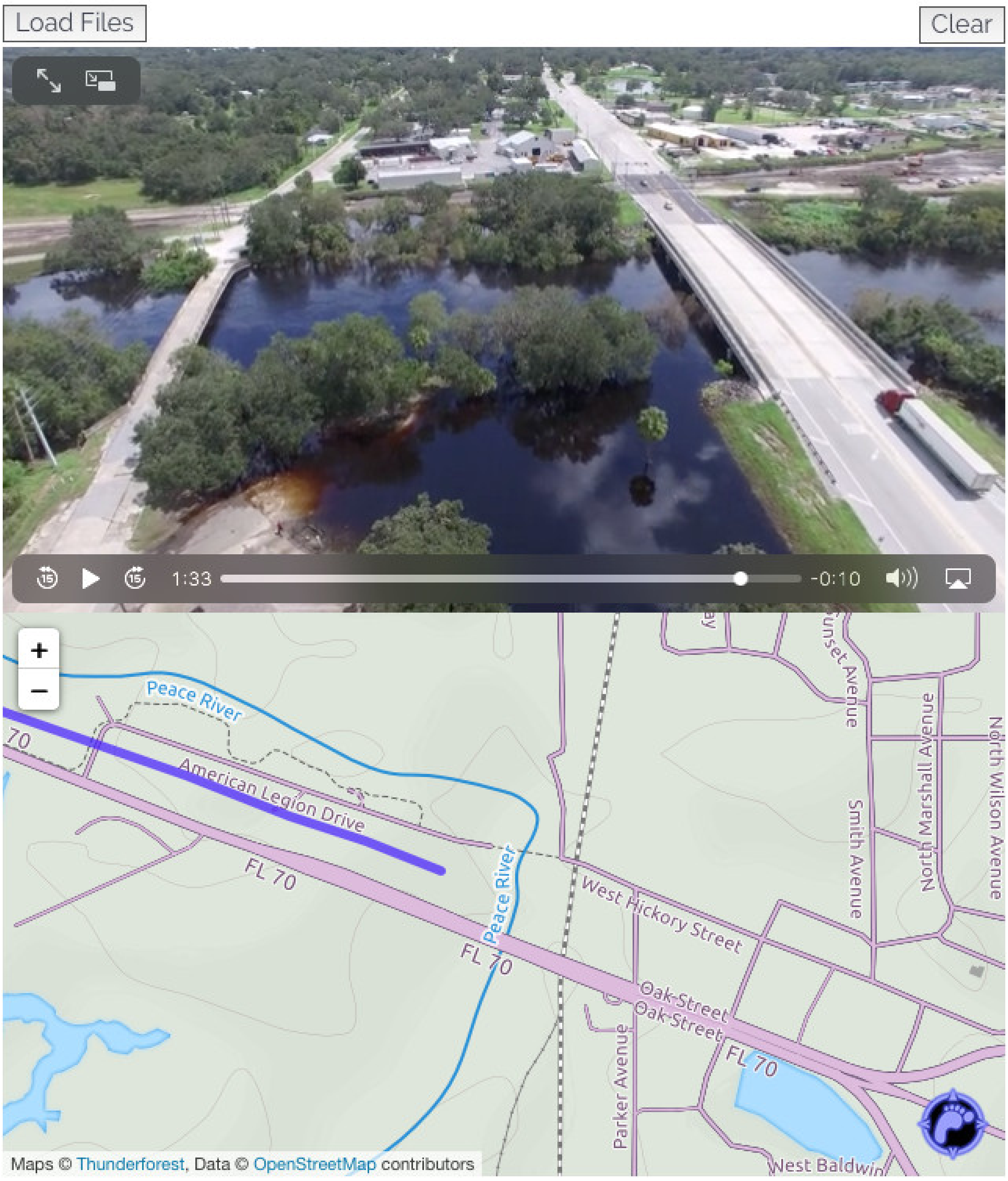


Figure 13. Drone With Forward Facing Camera

Files for these demos are available from the OGC Portal at Innovation Program/OGC Testbed-16/Full Motion Video/Demos/20200827_WebVMT [https://portal.ogc.org/modules/files/details.php?m=files&artifact_id=94463] which can be displayed at the WebVMT website [https://webvmt.org] as described in the Readme file. OGC Portal access is available to OGC Members only.

8.4.2. Truck Demo

This footage shows a haulage truck travelling along a highway filmed from a circling reconnaissance aircraft. Though the source video file contains no MISB 0903 data to track this target, the Frame Center (MISB 0601, tags 23 & 24) has been exported to a WebVMT path to best represent this from the information available.

The accuracy of the exported location data is demonstrated as the truck passes under a bridge, showing that the truck and path are well synchronized.

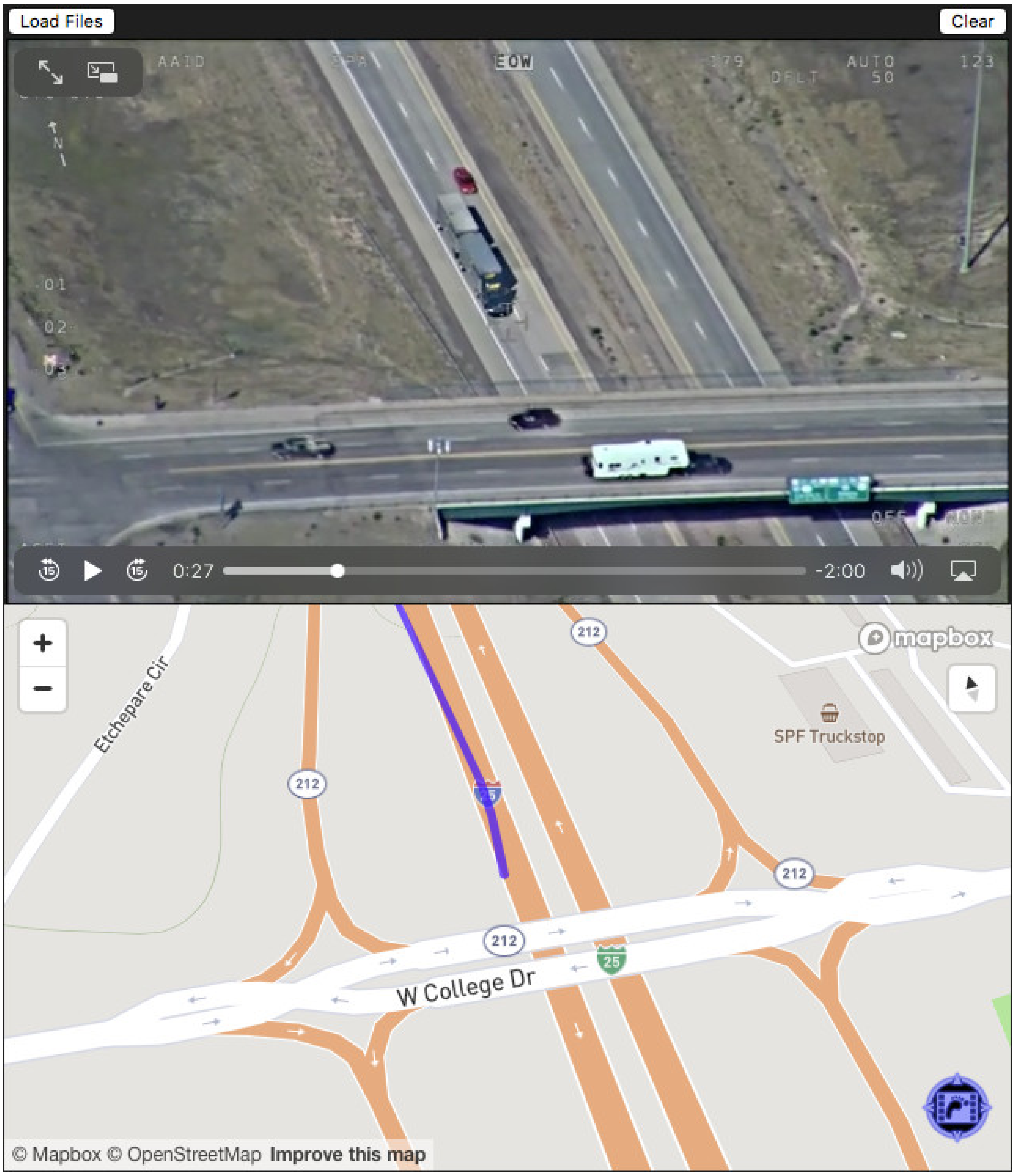


Figure 14. Truck Approaching Bridge

A crosshair overlay baked into the video footage indicates the position of the frame center. Once again, the accuracy is demonstrated as the location of the frame center drifts on to the opposite carriageway and across to the truck stop exit road before recentering on the target truck, and this is correctly reflected in the exported WebVMT path.

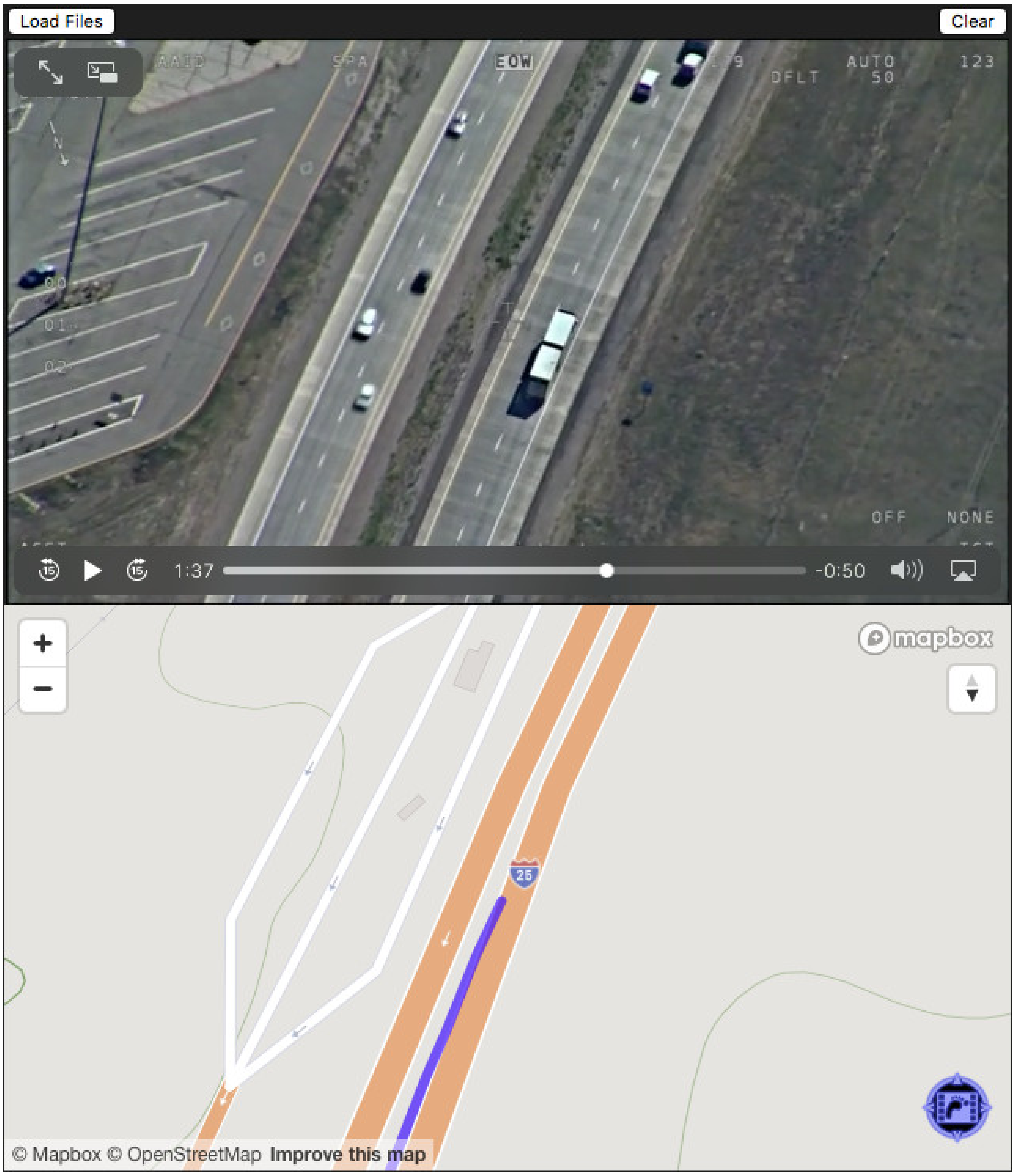


Figure 15. Truck Passing Truck Stop

Files for this demo are available from the OGC Portal at Innovation Program/OGC Testbed-16/Full Motion Video/Demos/20200903_WebVMT [https://portal.ogc.org/modules/files/details.php?m=files&artifact_id=94515] which can be displayed at the WebVMT website [https://webvmt.org] as described in the Readme file. Note that the MapboxGL [https://docs.mapbox.com/mapbox-gl-js] display option may not be available as this was a pre-release prototype feature at the time of publication of this ER.

8.5. Data Analysis and Visualization

As a preliminary step towards linking metadata targets (MISB 0903 VTarget Pack) into moving object trajectories, target locations were displayed with WebVMT shape commands [https://w3c.github.io/sdw/proposals/geotagging/webvmt/#webvmt-shape-commands] in order to better understand the nature of the sample data.

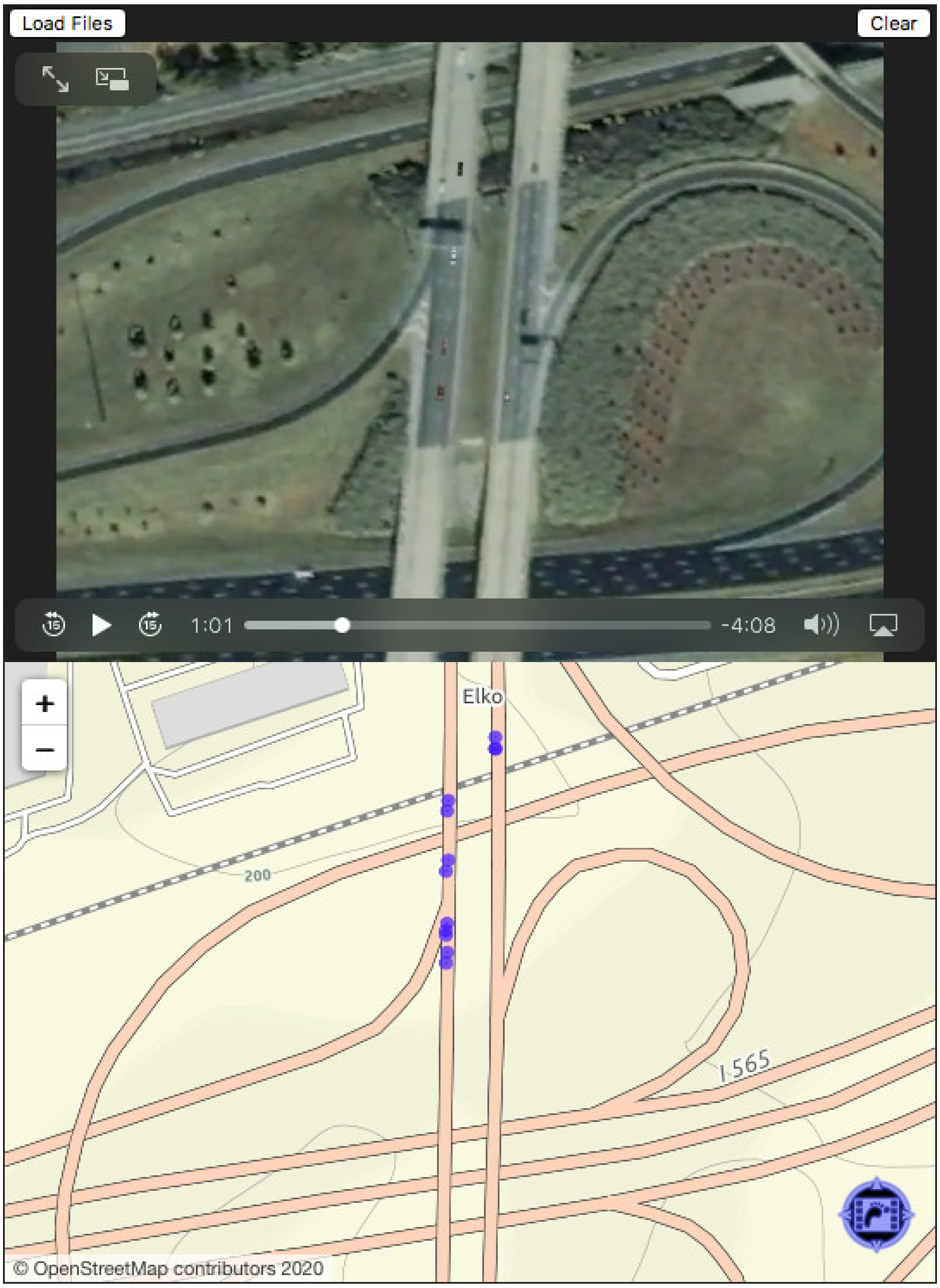


Figure 16. Metadata Tracking Visualization With WebVMT

Mapping targets to short-lived WebVMT shape cues allowed MISB content to be displayed quickly and easily and provided simple confirmation that:

1. Target locations can sensibly be linked into trajectories.
2. Target data is sporadic, with significant drop-out periods.
3. Only a subset of moving objects in the video are tracked in metadata.
4. Target location accuracy varies and does not always coincide with roads.

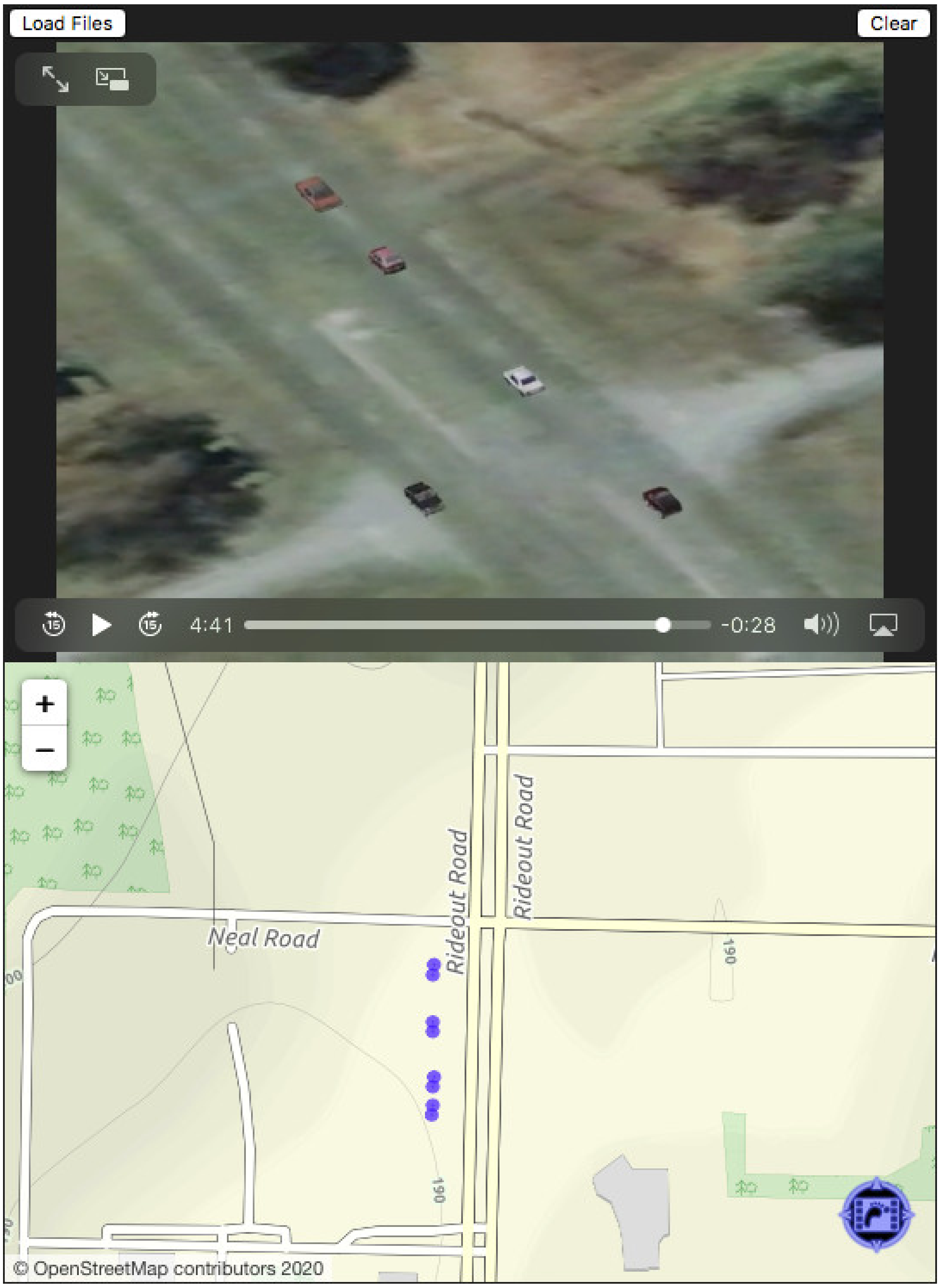


Figure 17. Metadata Accuracy Visualization With WebVMT

Such details can provide useful insight for data analysts to quickly assess metadata content and ensure that suitable techniques are employed to avoid wasting valuable time and resources.

8.6. Accessibility and Quality

An unexpected side-effect of building a media file metadata parser from scratch was that the software started to highlight errors in the sample video files being used to test it. Analysis pinpointed mistakes in the format usage and data content, including:

- The mandatory Version Number (MISB 0601, tag 65) was omitted from a couple of files so it was unclear which version of the MISB standard was used and how data content should be interpreted.

- A fixed Target Width of 0 (MISB 0601, tag 22) was contained in every packet of a file representing 'the linear ground distance between the center of both sides of the captured image' which did not tally with the video image and could be omitted as it is not mandatory.
- A spurious offset in the MPEG-2 Presentation Timestamp (PTS) values was present throughout another file so metadata were not correctly synchronized with video and shifted forwards in time by a fixed amount.

Quality control issues aside, the presence of these errors indicates that metadata are currently not easily accessible in video files as these systematic mistakes have not been picked up and corrected. Providing better metadata accessibility would help improve data quality.

8.7. Use Cases and Benefits

A number of video metadata use cases were identified during this project and are summarized below including the specific benefits of exporting data to WebVMT format.

8.7.1. Crowdsourced Evidence

Video footage evidence can be submitted by the public with accurate location, for instance to report dangerous drivers using dashcam footage or to report forensic evidence of boats smuggling goods/people spotted from pleasure craft or the shore using smartphone footage. See also the WebVMT Police Evidence [https://w3c.github.io/sdw/proposals/geotagging/webvmt/#policeevidence] use case;

Benefits: Video evidence with accurate location in machine-readable format can be sourced from the public.

8.7.2. Aggregated Streams

Provide a common format for sharing body-worn video footage for police and rescue services, which is compatible with disparate streams from vehicle dashcams and aerial video from drones and helicopters that can be easily aggregated for situational awareness or during pursuit. See also the WebVMT Area Monitoring [https://w3c.github.io/sdw/proposals/geotagging/webvmt/#areamonitoring] use case;

Benefits: Geotagged video can be aggregated and shared in a common format for live pursuit/situational awareness.

8.7.3. Underground Inspection

Underground features can be inspected using a remote camera, either integrated into a drilling rig or mounted on a remotely operated vehicle in a borehole or pipe. Location is calculated using inertial navigation so video frames can be tagged with geospatial information to enable features visible to the camera (and other sensor data) to be accurately mapped. This technique is used in the oil and gas industries, and can also offer valuable insight for inspection of underground assets such as pipes and cables.

Benefits: Valuable assets in inaccessible locations can be inspected cheaply and accurately with searchable access.

8.8. Testbed-16 Results

WebVMT work in Testbed-16 FMV demonstrated the lack of accessibility to geospatial video metadata in web browsers and highlighted the effort necessary to extract in-band metadata from an MPEG-2 stream in terms of complexity, bandwidth, and manpower.

8.8.1. Web Alignment

WebVMT's out-of-band design is well-suited for moving objects and sensor data to decouple metadata access from the current video playback time. This approach avoids the inherent overheads of in-band design and enabling seamless integration with web browsers and search engines to make video metadata easily accessible to the online community. Exporting MISB metadata to WebVMT has demonstrated the value of exposing geospatial metadata in a common format, including:

- Integration with web map APIs such as the Leaflet [https://leafletjs.com/] and MapboxGL [https://docs.mapbox.com/mapbox-gl-js] JavaScript libraries for rapid visualization.
- Enabling JavaScript access for developers to build web applications for mobile and desktop devices worldwide.
- Making online video metadata machine readable to enable search engine integration to rapidly identify relevant geotagged video content on the web.

8.9. Conclusion

The WebVMT contribution to the Testbed-16 FMV activity demonstrated valuable ways in which MISB metadata could be quickly and easily exploited online by mapping to WebVMT and how closer integration with web technologies could improve accessibility.

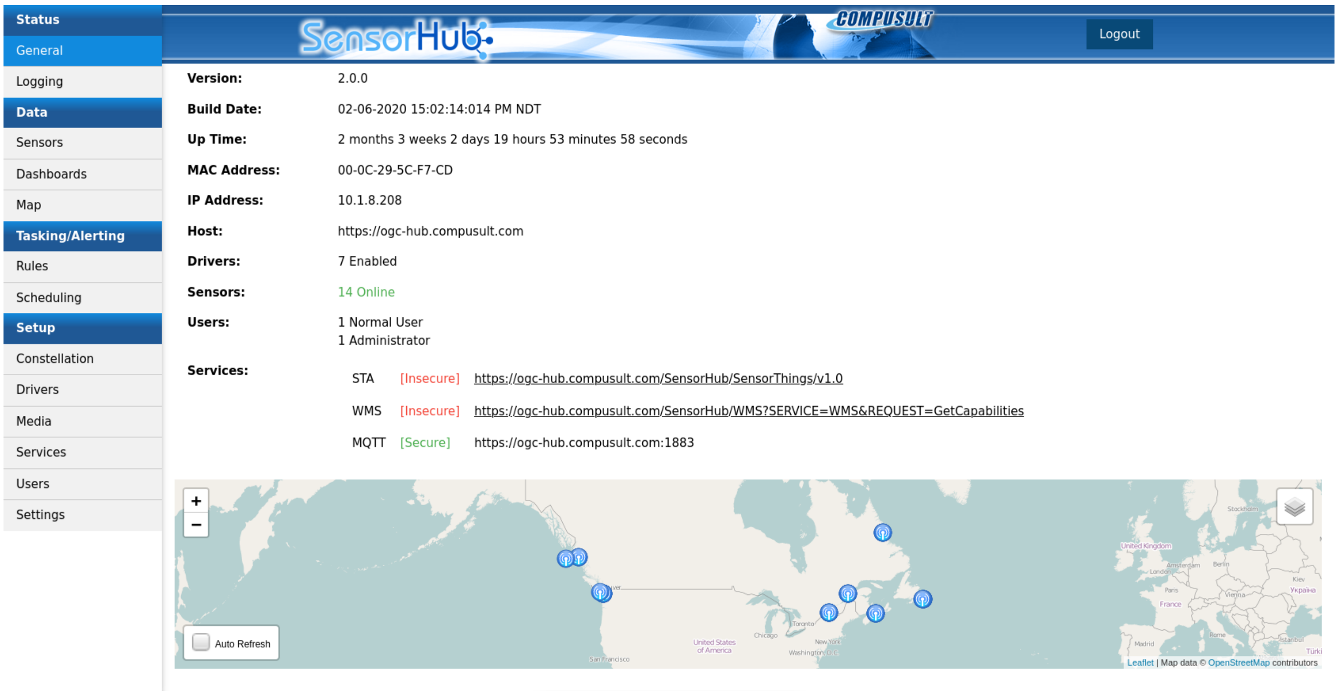
Chapter 9. Software Design and Demonstrator overview

9.1. Overview

This section contains a formal description of the different modules implemented in the Proof of Concept (POC) in terms of UML diagrams, libraries and technologies as well as an overview of the functionality and challenges faced.

9.2. Demonstrator 1

The goal of the Testbed 16 FMV demonstration is to use the SensorHub web application to seamlessly unite and broadcast observations from varying Full-Motion Video (FMV) and MovingFeatures (MF) sources in a manner intuitive to observers. As described in the Overview, this application primarily uses the OGC SensorThings API Standard to display sensor observations on dashboards. The SensorHub application's main use in this project was to convert diversely structured FMV and MF sources, accessible via the OGC SensorThings API, to be displayed on the end user's dashboard.



Version:	2.0.0
Build Date:	02-06-2020 15:02:14:014 PM NDT
Up Time:	2 months 3 weeks 2 days 19 hours 53 minutes 58 seconds
MAC Address:	00-0C-29-5C-F7-CD
IP Address:	10.1.8.208
Host:	https://ogc-hub.compusult.com
Drivers:	7 Enabled
Sensors:	14 Online
Users:	1 Normal User 1 Administrator
Services:	STA [Insecure] https://ogc-hub.compusult.com/SensorHub/SensorThings/v1.0 WMS [Insecure] https://ogc-hub.compusult.com/SensorHub/WMS?SERVICE=WMS&REQUEST=GetCapabilities MQTT [Secure] https://ogc-hub.compusult.com:1883

Figure 18. SensorHub Client.

9.2.1. SensorHub Overview

SensorHub is a Compusult product that harnesses the OGC SensorThings API to shuttle data from various Internet of Things (IoT) sources to a common destination dashboard. The SensorHub application consists mainly of *drivers*, which take either fixed files or periodically polled source URLs as inputs and generate SensorThings sensors as output.

Sensors can be viewed and manipulated via the Dashboard Editor. Users can see the current observation values of multiple widgets or navigate through the historically recorded values of a single one. Front-end widgets that depict similar units of measure will be displayed in a consistent manner, even if the originating data sources are in different formats.

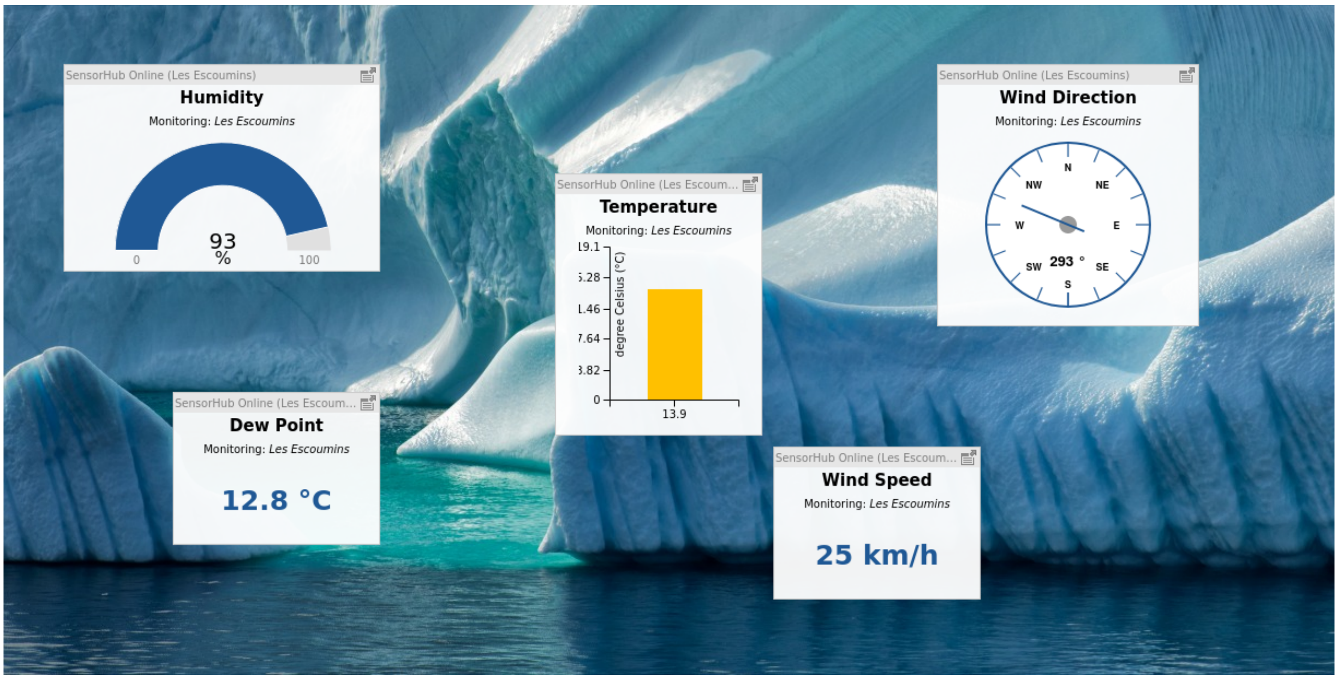

SensorHub Online (Les Escoumins)
Humidity
Monitoring: Les Escoumins
93
%
0
100
SensorHub Online (Les Escoum...
Temperature
Monitoring: Les Escoumins
degree Celsius (°C)
13.9
SensorHub Online (Les Escoumins)
Wind Direction
Monitoring: Les Escoumins
N
NE
E
SE
S
SW
W
NW
293 °
SensorHub Online (Les Escoum...
Dew Point
Monitoring: Les Escoumins
12.8 °C
SensorHub Online (Les Escoum...
Wind Speed
Monitoring: Les Escoumins
25 km/h

Figure 19. SensorHub Dashboard Editor.

SensorHub also supports logic to create rule-based alerts, which, via broadcast protocols such as Message Queuing Telemetry Transport (MQTT), can be transmitted using the Tasking component of the SensorThings API.

9.2.2. OGC SensorThings Sensing API

The Sensing side of the OGC SensorThings API deals with creating, housing, and broadcasting hierarchically structured observations and is the primary format into which the end products of all SensorHub inputs must be converted. SensorHub components are depicted in the below diagram:

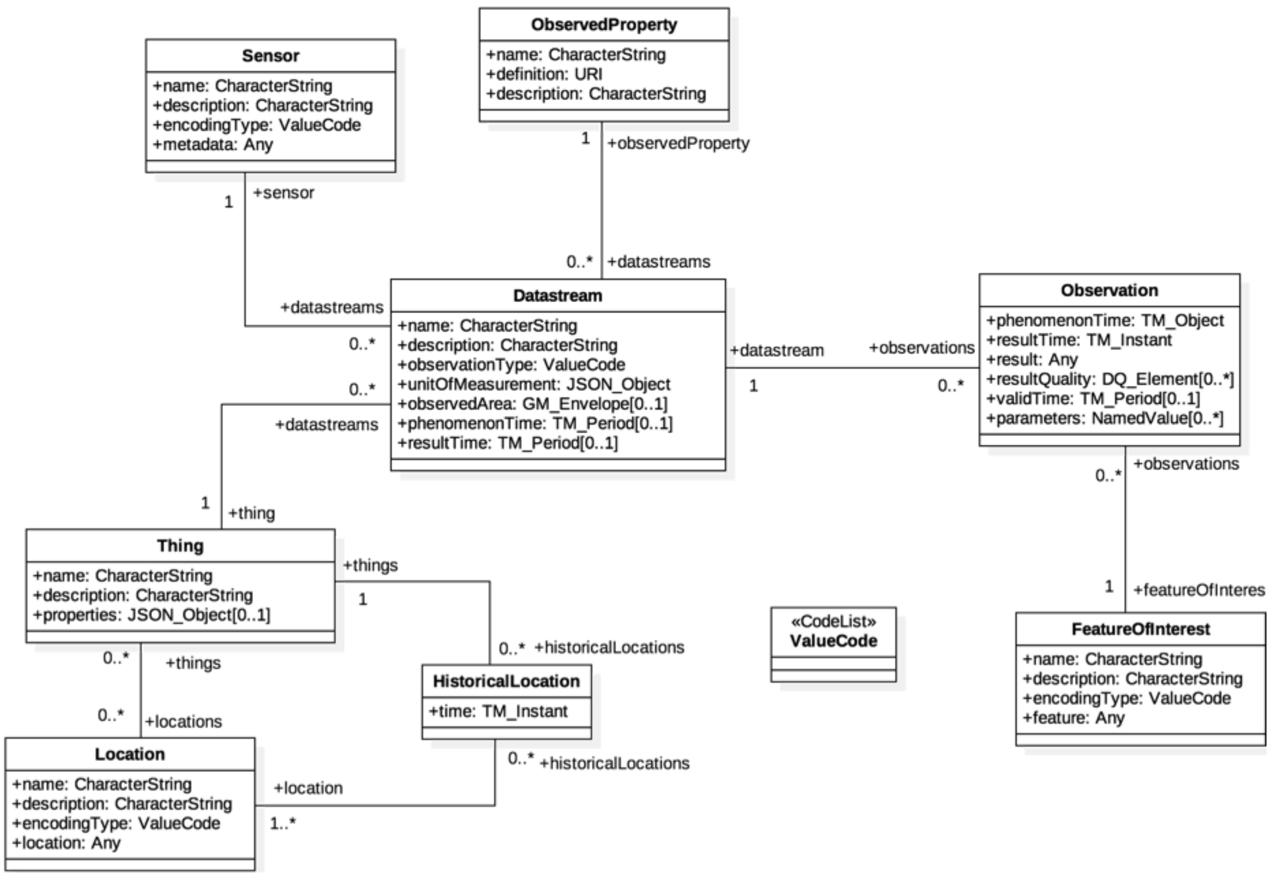


Figure 20. SensorThings class diagram.

9.2.3. MovingFeatures Package Structure

Structurally, the MovingFeatures package is typical of a SensorHub driver package. Each of these accepts files of a certain type (or set of types) as an input, converts the files to output OGC SensorThings entities, and performs the associated input/output (I/O). User interaction with the main class, **MovingFeaturesDriver**, is done via a **MovingFeaturesConfig** class and associated Jakarta Server Pages (.jsp) document, wherein the user can manage a list of file paths.

This driver supports inputs in either Full-Motion Video formats (e.g., .mp4, .h264, .mpeg4) or existing MovingFeatures .json formats, since video files will be converted to MovingFeatures JSON as part of the process of converting them to OGC SensorThings. The workflow for both file types is depicted in the below Unified Modeling Language (UML) diagram:

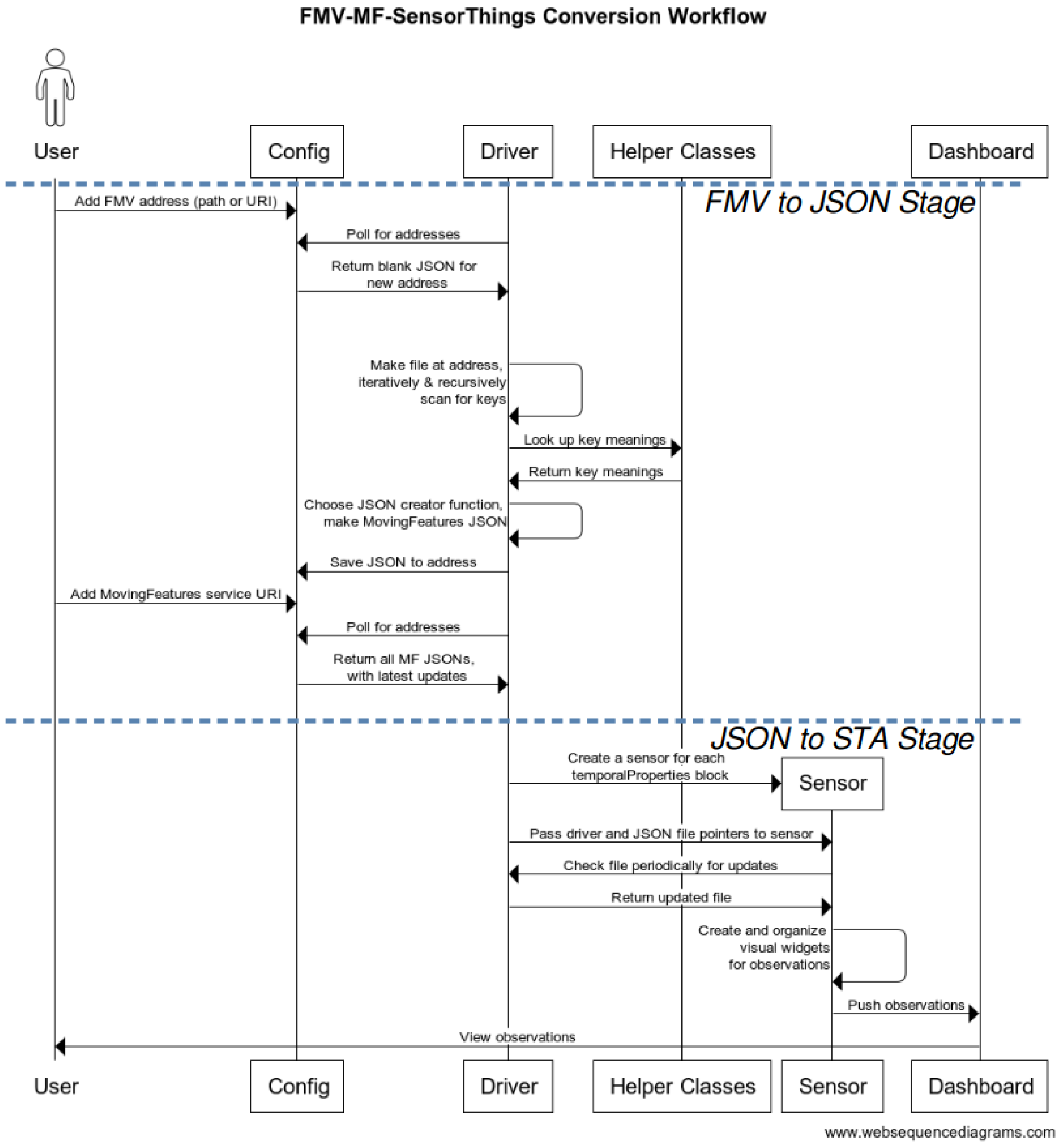
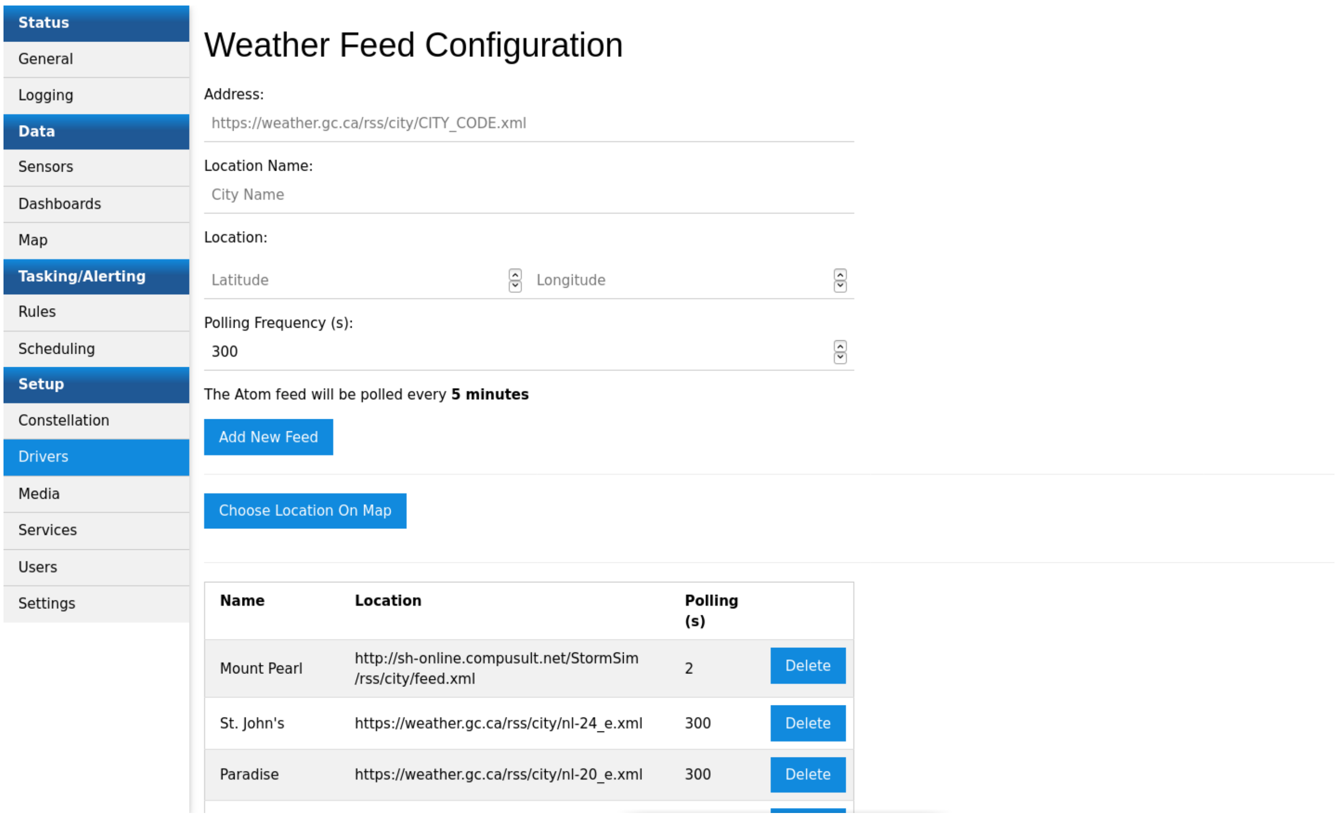

FMV-MF-SensorThings Conversion Workflow
User
Config
Driver
Helper Classes
Dashboard
FMV to JSON Stage
Add FMV address (path or URI)
Poll for addresses
Return blank JSON for new address
Make file at address, iteratively & recursively scan for keys
Look up key meanings
Return key meanings
Choose JSON creator function, make MovingFeatures JSON
Save JSON to address
Add MovingFeatures service URI
Poll for addresses
Return all MF JSONs, with latest updates
JSON to STA Stage
Create a sensor for each temporalProperties block
Sensor
Pass driver and JSON file pointers to sensor
Check file periodically for updates
Return updated file
Create and organize visual widgets for observations
Push observations
View observations
www.websequencediagrams.com

Figure 21. FMV-MovingFeatures-SensorThings sequence diagram.

9.2.3.1. MovingFeaturesConfig

In a SensorHub driver, the configuration code generally produces a landing page like the one shown below. Users have the option to enter and add a new Uniform Resource Identifier (URI), as well as to view the existing ones and delete any no longer desired ones.



Weather Feed Configuration

Address: https://weather.gc.ca/rss/city/CITY_CODE.xml

Location Name: City Name

Location: Latitude Longitude

Polling Frequency (s): 300

The Atom feed will be polled every **5 minutes**

Name	Location	Polling (s)	
Mount Pearl	http://sh-online.compusult.net/StormSim/rss/city/feed.xml	2	Delete
St. John's	https://weather.gc.ca/rss/city/nl-24_e.xml	300	Delete
Paradise	https://weather.gc.ca/rss/city/nl-20_e.xml	300	Delete

Figure 22. Weather feed configuration.

Internally, the config classes store a source JSON array containing the URIs and any MovingFeatures properties made from each file. Each time the list of source files changes, the driver class is re-initialized and a new fetch of any sensors originating from any new sensor files will be performed. The JSON array will be updated accordingly with the latest versions of all of the objects.

9.2.3.2. MovingFeaturesDriver main file

The **MovingFeaturesDriver** is the broker class that forms the core of the parsing process, assembling JSON objects from the data source locations held in the **MovingFeaturesConfig** and transforming them to Sensors that will be visually displayable.

FMV files in the configuration list are passed, one by one, to the **parse()** function. This takes in a JSON object, which is either blank or contains the observations that were made so far. **parse** also looks at the dataset type (e.g., Unmanned Air Systems [UAS], VMTI, or VTrack), permitting recursive parsing when one dataset is nested inside another. The **parse** function operates on a consistent basis for all MISB sets and at all levels of recursion, due to those sets' consistent KLV (Key-Length-Value) structuring.

In parsing, the **Driver** first checks the extension on each file. For each file that is already .json, the Driver moves to generating **MovingFeaturesSensor**s. This is shown in the bottom section of the UML diagram workflow. However, if the extension is not .json the file is considered an FMV and needs the JSON extracted first, as in the top section.

To extract, the **Driver** scans 16-byte sequences from the file to try to detect one of the 16-byte Universal Keys, defined in MISB 0903 and 0601, that signal the start of a new dataset. In a video file, this corresponds to the start of the metadata for a new frame.

Once this Key, or any of the Keys inside the dataset, is found, the immediately following bytes are scanned to determine the Length of the data contained therein. Lengths are given as Basic Encoding Rules (BER) numbers, either short-form or long-form, and thus the parser can inherently determine how many bytes it has to jump forward to get to the Value. The mechanics of looking up keys and parsing BER numbers are done mainly via the Helper Classes.

After the length is read, a number of bytes equal to the length are consumed as well, and these form the Value. The code runs a **switch()** statement on the original KLV Key to determine the meaning of the value (e.g., does it represent a geographical coordinate, a timestamp, or a name?).

Depending on the key meaning, a JSON creator function is selected and called. JSON creator functions take in the JSON object that was passed to the parser and populate its **properties**, **temporalProperties**, and **temporalGeometry**, creating new entities wherever necessary as stipulated by the OGC MovingFeatures specifications. If measurements or numerical readings are present in the bytes, additional helper functions may be called to convert them from bytes to doubles or integers via MISB encoding schemas such as Integer MAPping with starting point B (IMAPB) or Universal Data Set (UDS).

The ultimate JSON file revised by the parser will comply with the MovingFeatures specifications. This file is then placed back in the **Config**, so that all data previously read is available in case the FMV video feed has more data written to it later. The offset, or number of bytes the reader has gone through already, is also stored to prevent rereading of data. On the driver’s next activation, the stored JSON will be read by a different part of the parser, the same part that handles data sources that come in natively as MF-JSON. This process generates SensorThings entities.

9.2.3.3. MovingFeaturesSensor

The **MovingFeaturesSensor** (a subclass of **Sensor**, which implements the SensorThings API item of the same name) is the top-level entity that houses the SensorThings Datastreams and Observations that are passed to the dashboard.

When the **Driver** goes through the **Config**’s list of observations, it automatically creates a **Sensor** for each new JSON file it sees, including those made earlier by its own FMV-JSON conversion. From there, executes the workflow seen in the second stage of the UML diagram. This Sensor receives, as an argument, a path to the JSON object that represents its own data. Within this object, the **temporalProperties** are each converted to SensorThings **Datastream**s, and individual values within them to SensorThings **Observation**s. The **Sensor**’s constructor creates **Datastream**s and **Observations** based on the latest set of readings observed in the object; once the creation is done, the Hub’s core automatically makes widgets showing the **Observation**s, which can be arranged on a dashboard at will using SensorHub’s Dashboard Editor.

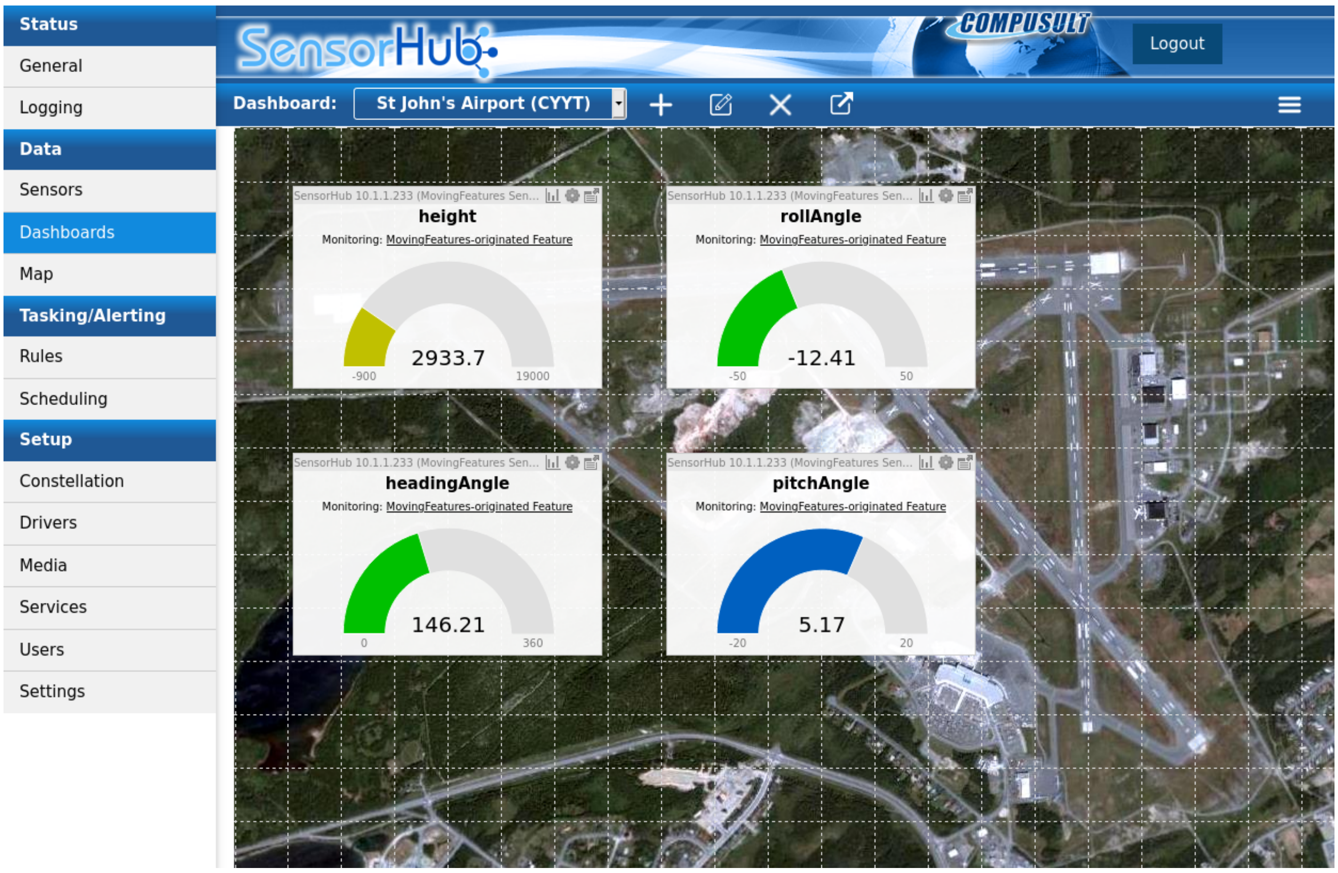


Figure 23. Observations in the Dashboard editor.

Periodically, every few seconds, the **Sensor** checks its own filepath to see whether its JSON contents have changed. If the **Driver** has detected new data in the frame, this will have been written to the JSON and the **Sensor** will send any new **Datastream**s and **Observation**s to the dashboard accordingly.

9.2.3.4. Helper Classes

A **BERParser** class is present in the package and linked to from the **Driver**'s parse function. Its role is to take in KLV Length and Basic Encoding Rules, Object IDentifiers (BER-OID) identifier portions of the incoming byte stream, translate them into a Java **long**, and output a two-integer pair containing (1) the decoded number and (2) the number of bytes that were consumed (the number of bytes *in* the number itself).

A **MovingFeaturesKeys** class is also present, serving as a lookup table that the **Driver** can use to access the key byte sequences. Maps that associate each key with the details specific to them are located here as well, and some of these have their own wrapper classes. For example, keys for properties that feed into the driver's UDS parse function are mapped to **IMAPDetails**, which states the minimum and maximum of the range being mapped onto, the number of input bytes (if fixed), and whether or not the input bytes should be treated as signed. This range is passed into the sensors, which use this information in constructing their widgets to set the minimum and maximum values that the gauge can display.

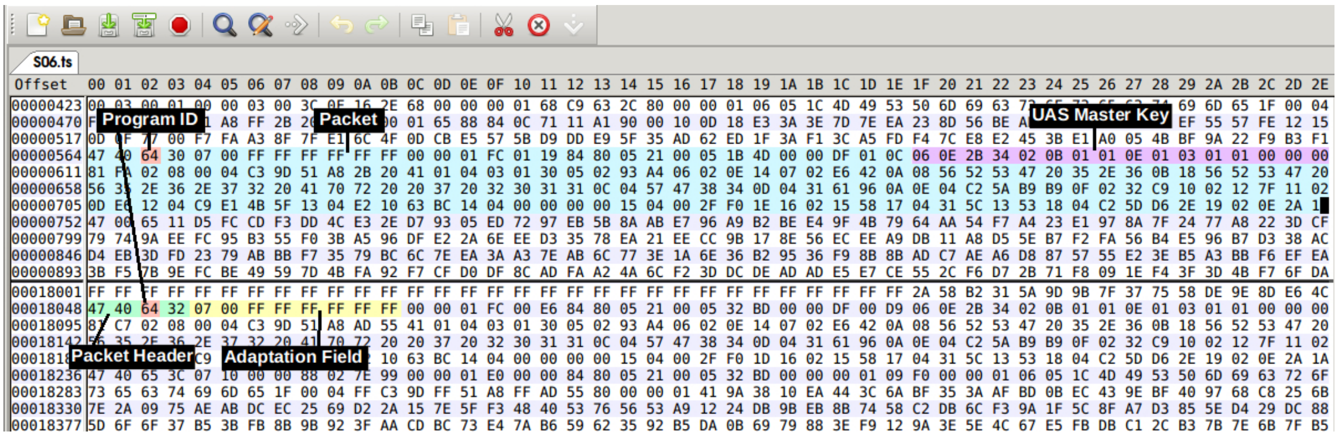
9.2.3.5. Handling of multi-packet MISB sets

The KLV data from a MISB metadata set is not always contiguous within a file due to the arrangement of Transport Stream (.ts) files in *packets* of data, which may be used to store content from multiple *programs* (i.e., video, audio, and metadata). However, the specifications for .ts itself impose several constraints on the sizes and formats of these packets, allowing the parser to unambiguously recover only the requisite KLV bytes.

The specifications for MPEG-2 Transport Stream files dictate that they will always be structured into packets of exactly 188 bytes. Each packet must begin with a four-byte header, optionally followed by an adaptation field and data payload, only the latter of which must be read to get the MISB metadata. As such, the locations of the headers and adaptation fields must be identified so that they can be skipped. However, the fixed size of the packets makes this a manageable task since packets are always aligned to begin on offsets that are multiples of 188. The software design of the parser makes use of this fact and a function called **muxOffset** inside the Driver will offset the parser, byte by byte, until it reaches a multiple of 188, at which point it will pause reading the data to search for the point where the MISB bytes resume.

Information from the four bytes of a packet's header, along with the following unsigned byte, which equals the length of the adaptation field, can be looked at to pinpoint the next byte of MISB data. Stored in each packet's header is a byte indicating the ID of the program that the packet's data belongs to. Typically, a .ts file will use one program ID for all its video and audio, and a second one for metadata. While **muxOffset** has the parser in pause mode, it will ingest the program ID and then offset itself 188 bytes at a time until it lands on the next one that belongs to the metadata. Once the resuming packet has been located, the parser will read the adaptation field length and jump forward by a number of bytes equal to it, thereby landing on the first byte of the payload and of the MISB data's continuation.

The diagram below depicts raw hex bytes in the example **S06.ts** file, indicating some features of MISB packetization. When the UAS Master Key is detected, its packet, containing 188 bytes beginning on an offset divisible by 188, is scanned to check its Program ID (here, 0x64). At the end of the packet, the Packet Headers will be scanned one by one until another containing a Program ID of 0x64 is found. Then this next packet's Header and Adaptation Field (length given by its first byte, 0x07) are read, and data resumes at the next byte after that.



```
Offset   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E
00000423 00 03 00 01 00 00 03 00 3C 0E 16 2E 68 00 00 00 01 68 C9 63 2C 80 00 00 01 06 05 1C 4D 49 53 50 6D 69 63 72 6F 73 65 63 74 69 6D 65 1F 00 04
00000470 F[illegible]1 A8 FF 2B 2[illegible]00 01 65 88 84 0C 71 11 A1 90 00 10 0D 18 E3 3A 3E 7D 7E EA 23 8D 56 BE A[illegible] EF 55 57 FE 12 15
00000517 0D 0F 77 00 F7 FA A3 8F 7F E1 6C 4F 0D CB E5 57 5B D9 DD E9 5F 35 AD 62 ED 1F 3A F1 3C A5 FD F4 7C E8 E2 45 3B E1 A0 05 4B BF 9A 22 F9 B3 F1
00000564 47 40 64 30 07 00 FF FF FF FF FF FF 00 00 01 FC 01 19 84 80 05 21 00 05 1B 4D 00 00 DF 01 0C 06 0E 2B 34 02 0B 01 01 0E 01 03 01 01 00 00 00
00000611 81 FA 02 08 00 04 C3 9D 51 A8 2B 20 41 01 04 03 01 30 05 02 93 A4 06 02 0E 14 07 02 E6 42 0A 08 56 52 53 47 20 35 2E 36 0B 18 56 52 53 47 20
00000658 56 35 2E 36 2E 37 32 20 41 70 72 20 20 37 20 32 30 31 31 0C 04 57 47 38 34 0D 04 31 61 96 0A 0E 04 C2 5A B9 B9 0F 02 32 C9 10 02 12 7F 11 02
00000705 0D E6 12 04 C9 E1 4B 5F 13 04 E2 10 63 BC 14 04 00 00 00 00 15 04 00 2F F0 1E 16 02 15 58 17 04 31 5C 13 53 18 04 C2 5D D6 2E 19 02 0E 2A 1
00000752 47 00 65 11 D5 FC CD F3 DD 4C E3 2E D7 93 05 ED 72 97 EB 5B 8A AB E7 96 A9 B2 BE E4 9F 4B 79 64 AA 54 F7 A4 23 E1 97 8A 7F 24 77 A8 22 3D CF
00000799 79 74 9A EE FC 95 B3 55 F0 3B A5 96 DF E2 2A 6E EE D3 35 78 EA 21 EE CC 9B 17 8E 56 EC EE A9 DB 11 A8 D5 5E B7 F2 FA 56 B4 E5 96 B7 D3 38 AC
00000846 D4 EB 3D FD 23 79 AB BB F7 35 79 BC 6C 7E EA 3A A3 7E AB 6C 77 3E 1A 6E 36 B2 95 36 F9 8B 8B AD C7 AE A6 D8 87 57 55 E2 3E B5 A3 BB F6 EF EA
00000893 3B F5 7B 9E FC BE 49 59 7D 4B FA 92 F7 CF D0 DF 8C AD FA A2 4A 6C F2 3D DC DE AD AD E5 E7 CE 55 2C F6 D7 2B 71 F8 09 1E F4 3F 3D 4B F7 6F DA
00018001 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 2A 58 B2 31 5A 9D 9B 7F 37 75 58 DE 9E 8D E6 4C
00018048 47 40 64 32 07 00 FF FF FF FF FF FF 00 00 01 FC 00 E6 84 80 05 21 00 05 32 BD 00 00 DF 00 D9 06 0E 2B 34 02 0B 01 01 0E 01 03 01 01 00 00 00
00018095 81 C7 02 08 00 04 C3 9D 51 A8 AD 55 41 01 04 03 01 30 05 02 93 A4 06 02 0E 14 07 02 E6 42 0A 08 56 52 53 47 20 35 2E 36 0B 18 56 52 53 47 20
00018142 56 35 2E 36 2E 37 32 20 41 70 72 20 20 37 20 32 30 31 31 0C 04 57 47 38 34 0D 04 31 61 96 0A 0E 04 C2 5A B9 B9 0F 02 32 C9 10 02 12 7F 11 02
0001818[illegible] C9 [illegible] 10 63 BC 14 04 00 00 00 00 15 04 00 2F F0 1D 16 02 15 58 17 04 31 5C 13 53 18 04 C2 5D D6 2E 19 02 0E 2A 1A
00018236 47 40 65 3C 07 10 00 00 88 02 7E 99 00 00 01 E0 00 00 84 80 05 21 00 05 32 BD 00 00 00 01 09 F0 00 00 01 06 05 1C 4D 49 53 50 6D 69 63 72 6F
00018283 73 65 63 74 69 6D 65 1F 00 04 FF C3 9D FF 51 A8 FF AD 55 80 00 00 01 41 9A 38 10 EA 44 3C 6A BF 35 3A AF BD 0B EC 43 9E BF 40 97 68 C8 25 6B
00018330 7E 2A 09 75 AE AB DC EC 25 69 D2 2A 15 7E 5F F3 48 40 53 76 56 53 A9 12 24 DB 9B EB 8B 74 58 C2 DB 6C F3 9A 1F 5C 8F A7 D3 85 5E D4 29 DC 88
00018377 5D 6F 6F 37 B5 3B FB 8B 9B 92 3F AA CD BC 73 E4 7A B6 59 62 35 92 B5 DA 0B 69 79 88 3E F9 12 9A 3E 5E 4C 67 E5 FB DB C1 2C B3 7B 7E 6B 7F B5
```

Figure 24. Main features of MISB packetization in **S06.ts**, *highlighted in a hex editor.*

9.3. Demonstrator 2

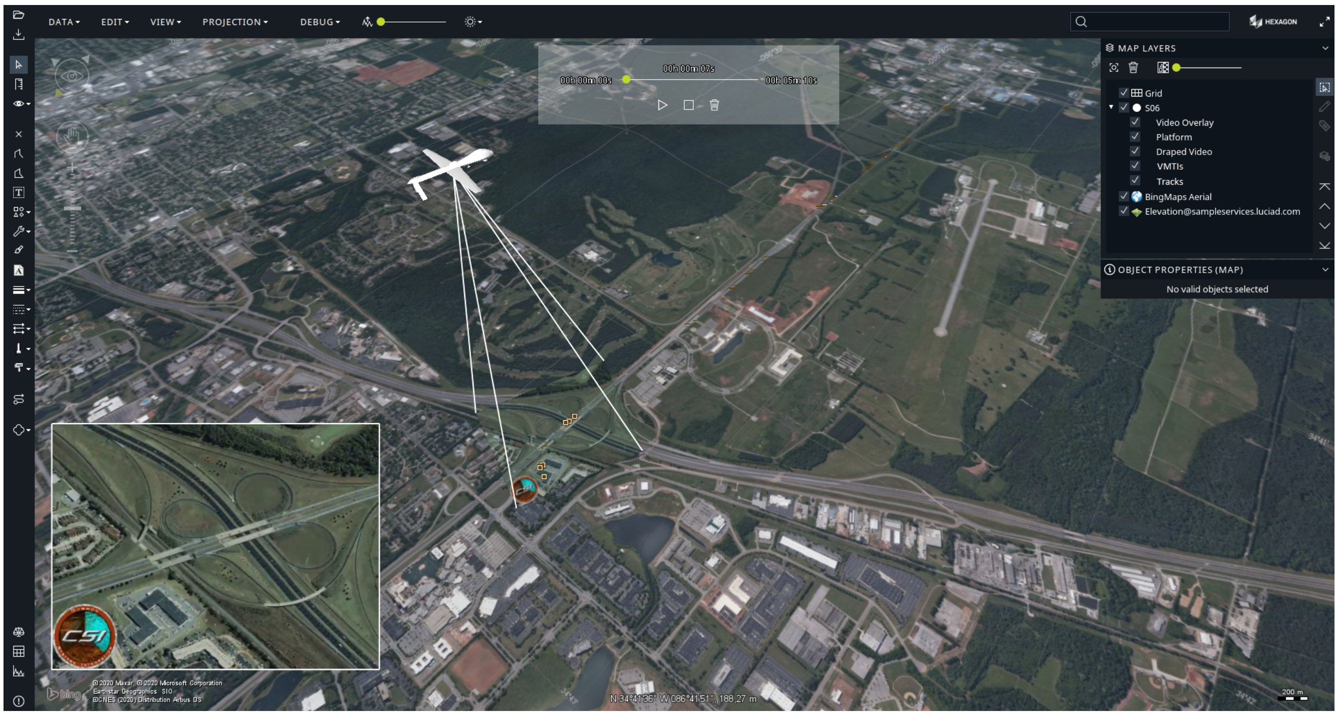


Figure 25. Overview of the client application for Demonstrator 2.

This demonstrator is based on the Luciad platform and previous work done to decode MISB 0601 metadata in order to drape video coming from a file or live stream.

9.3.1. Architecture

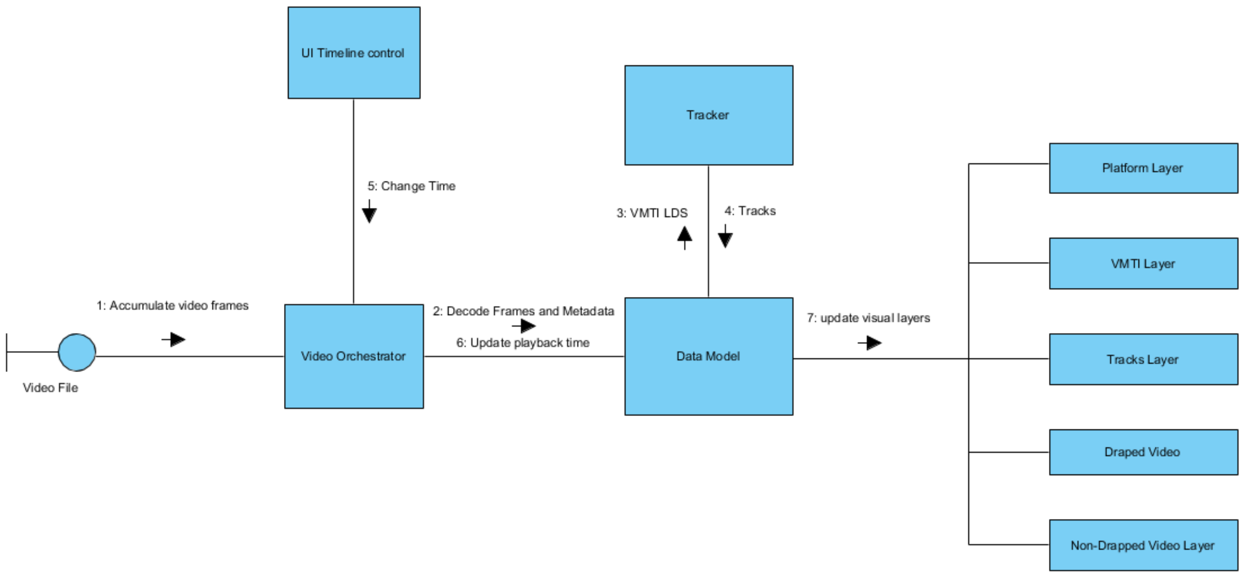


Figure 26. Communication diagram for a video file.

The simplified diagram above gives an idea of how the application handles a Video File. The MISB metadata is accumulated and decoded to a common data model. The data model contains the telemetry information and VMTIs. Then, the data model is augmented with tracks obtained from the tracker.

A UI widget allows the user to change the playback time. The Video Orchestrator sends a message to the data model that the current time has changed updating the different visual layers.

The layers show the individual VMTI detections, the tracks, the platform and field of view, the draped video and a separate window for an un-draped video.

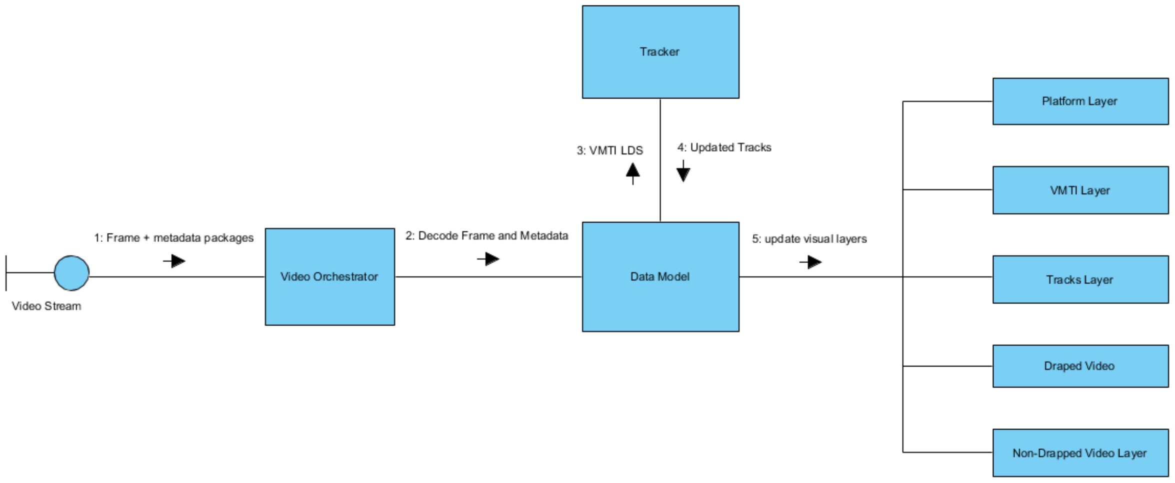


Figure 27. Communication diagram for a video stream.

The application also allows connecting to a video stream. The difference is the absence of a playback widget and the tracking that occurs in real time.

9.3.1.1. Common Data Model

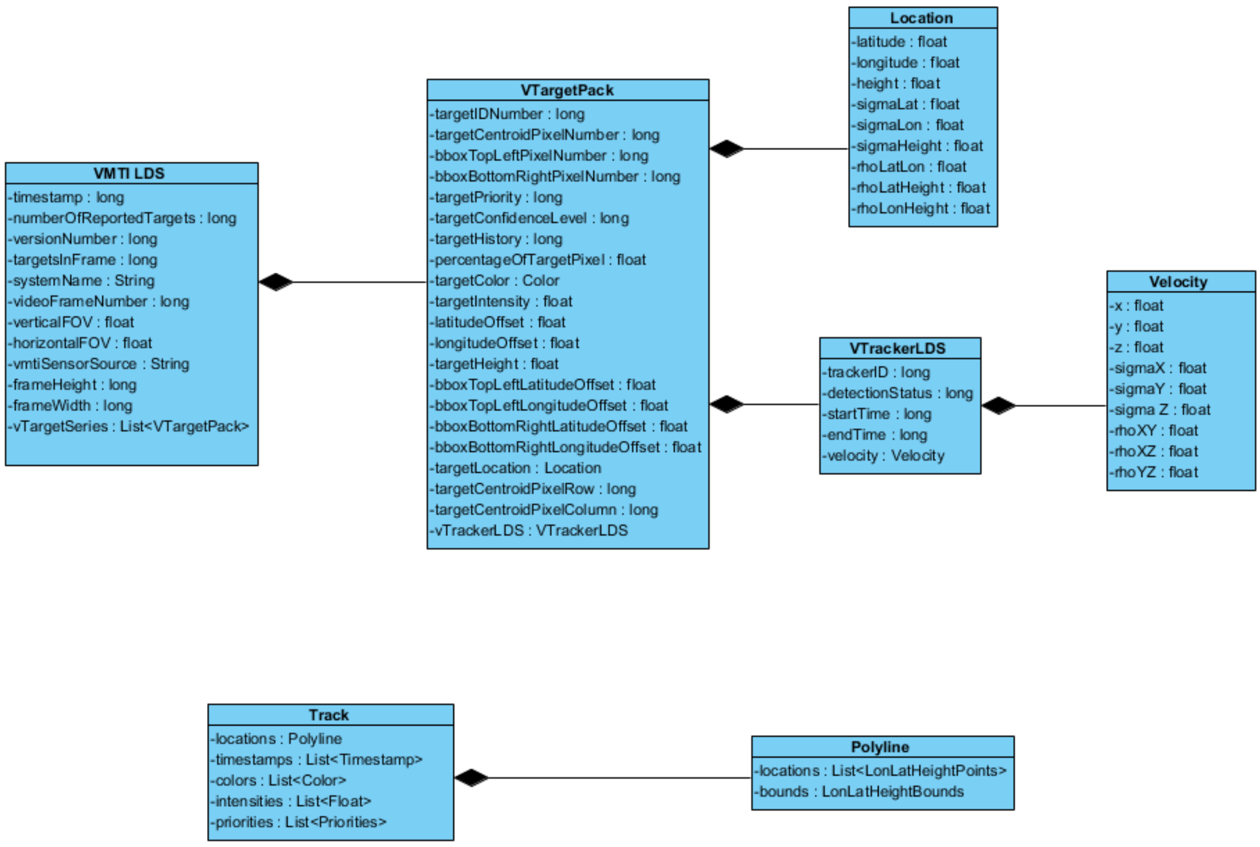


Figure 28. Data Model.

The graph above shows the most common parts of MISB 0903 and the track data model. These data-models are implemented as temporary in-memory representation of the data for on screen representation.

The intermediate data model also serves as the basis to encode tracks to interoperability formats (OGC Moving Features and OGC O&M).

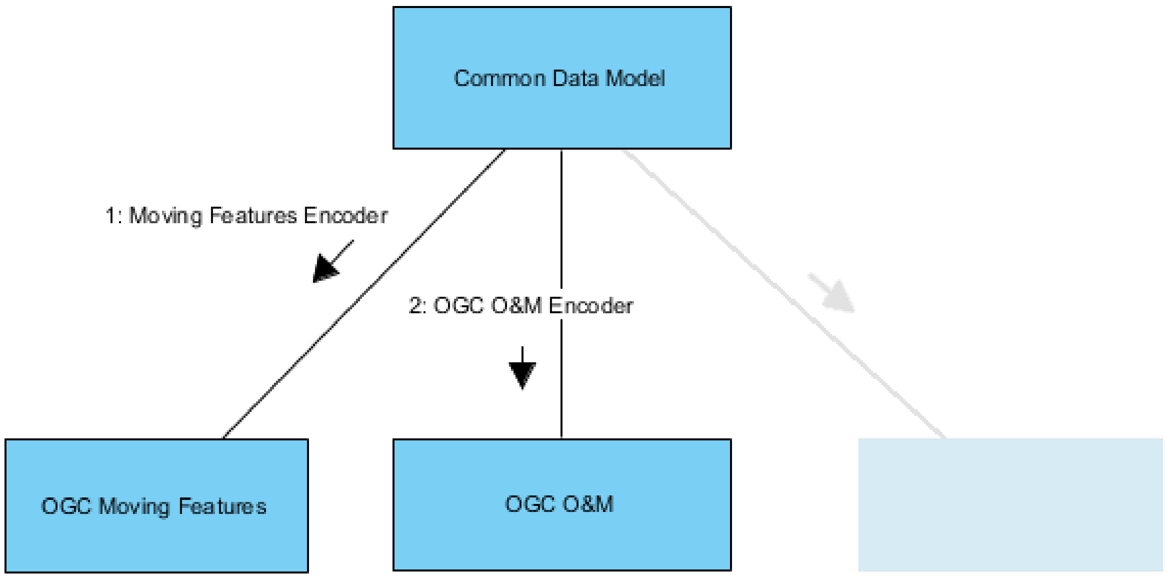


Figure 29. A common Data Model helps in the design of multiple format encoders.

This common data model is much easier to manage than the KLV encoded binary messages of MISB 0903 and the encoding process is decoupled from the decoding of MISB 0903.

9.3.2. Data quality

Drones have limited hardware but are asked, in real-time, to encode video, detect moving objects and calculate their location relative to the telemetry of the drone and the elevation model they have in memory. As a result, the data is not always high quality.

Several factors can affect the accuracy of detections. The algorithm that detects moving objects from the video is subject to errors, the elevation model of the earth directly impacts the conversion from pixel to longitude/latitude and finally, the telemetry of the drone and camera may be inaccurate.

More generally, this results in inaccurate locations, false positives and missing detections.

To some extent, the quality of the data can be improved with post-processing.

9.3.2.1. Effect of elevation on location accuracy

Elevation will have an effect on the location and deformation of a video that is being draped on terrain.

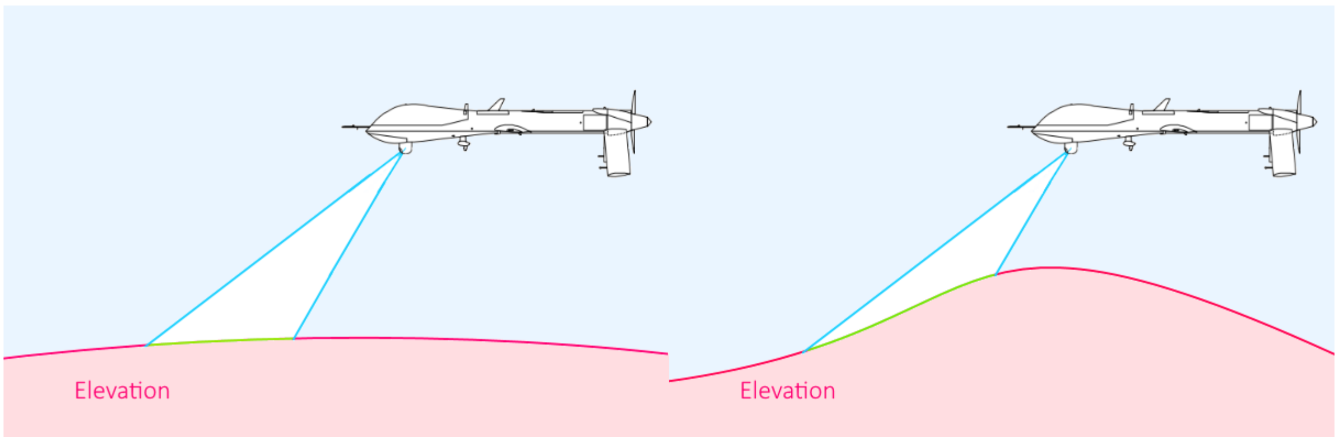


Figure 30. Effect of elevation on draping.

Different elevation models (left and right) can result in large differences the location and deformation of a draped video.

The detection of moving objects will be done relative to the video in terms of affected pixels but in order to communicate a longitude, latitude and height, a projection on terrain elevation must be done.

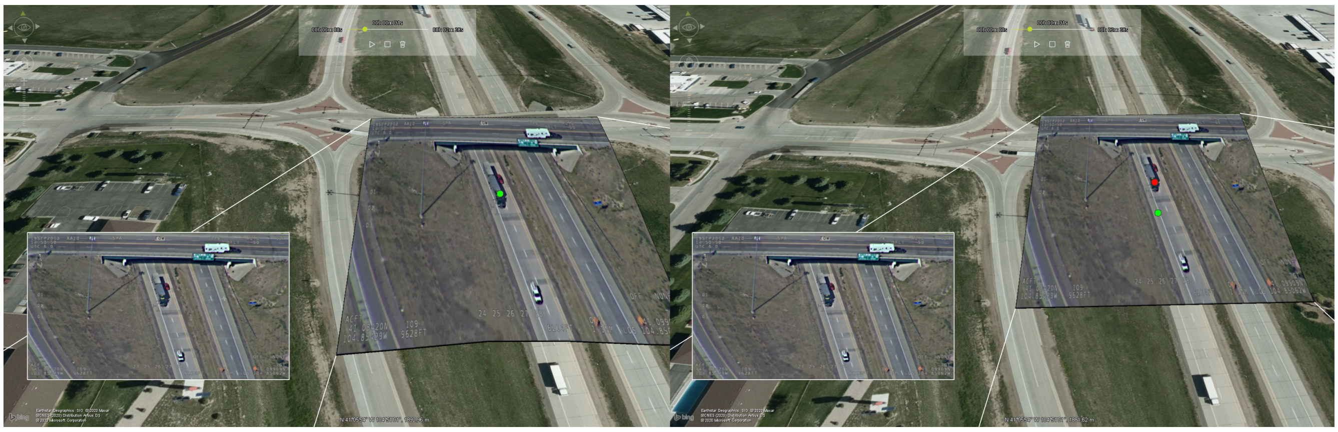


Figure 31. Effect of elevation on location accuracy.

The images above compare the location of a truck from a projected video with high accuracy elevation (left) and low accuracy elevation (right). The green and red dots show the error caused by the low quality elevation.

As a result, the video and detections may not match depending on what elevation model is used and there is no way of knowing how accurate an elevation model was used in the calculations.

Because MISB 0903 has fields to indicate the pixels that correspond to a detection, the location may be re-calculated in post-processing with a high accuracy elevation model.

9.3.2.2. false positives and false negatives

The on-board software responsible for detections will be imperfect and false positive and negative detections will occur.

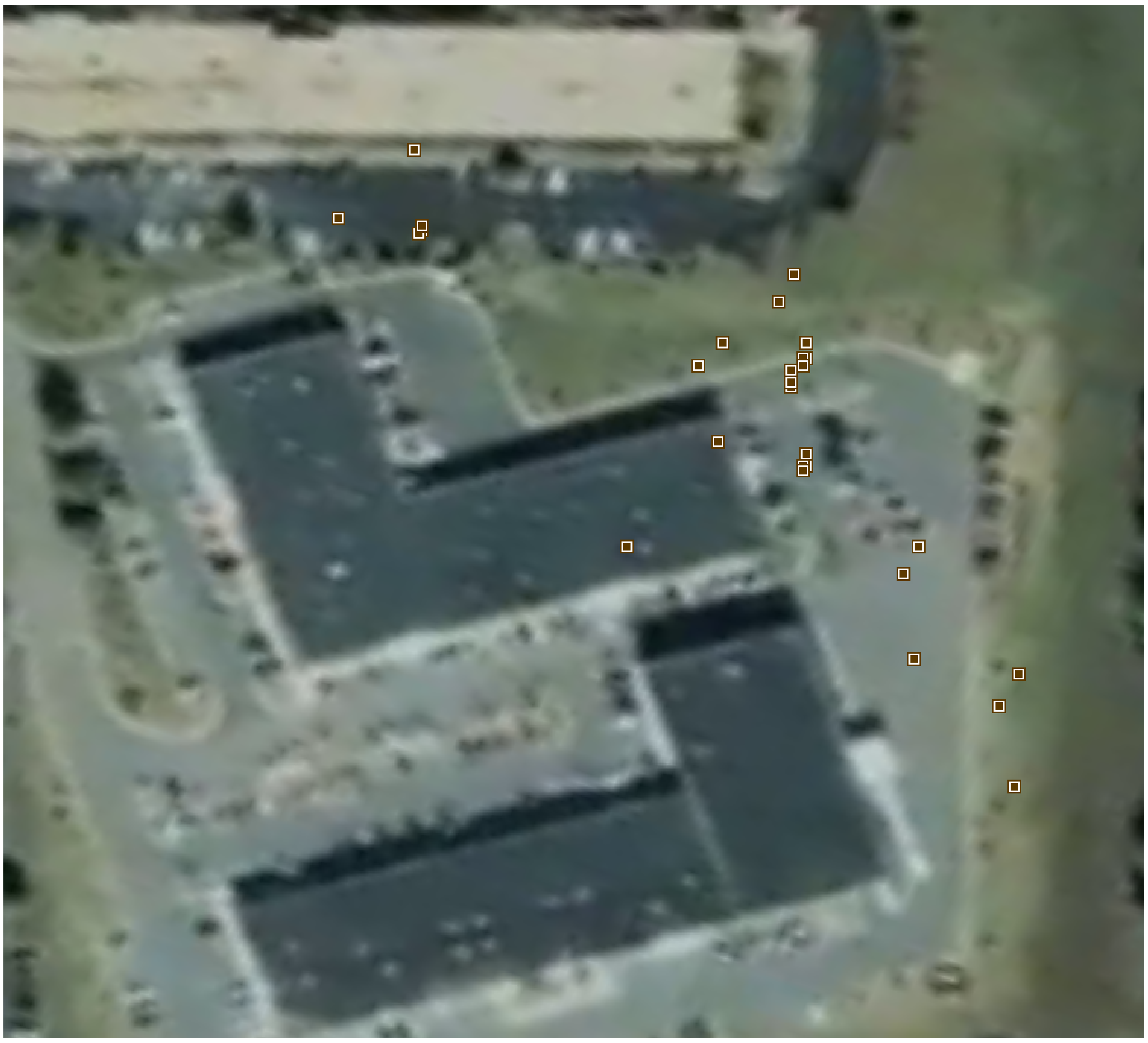


Figure 32. False positive detections.

The detections above are false positives detected because of a jittery and out of focus camera. False positives may be identified because they are outliers, too distant from any other detection. During tracking, entire tracks are dismissed based on length.

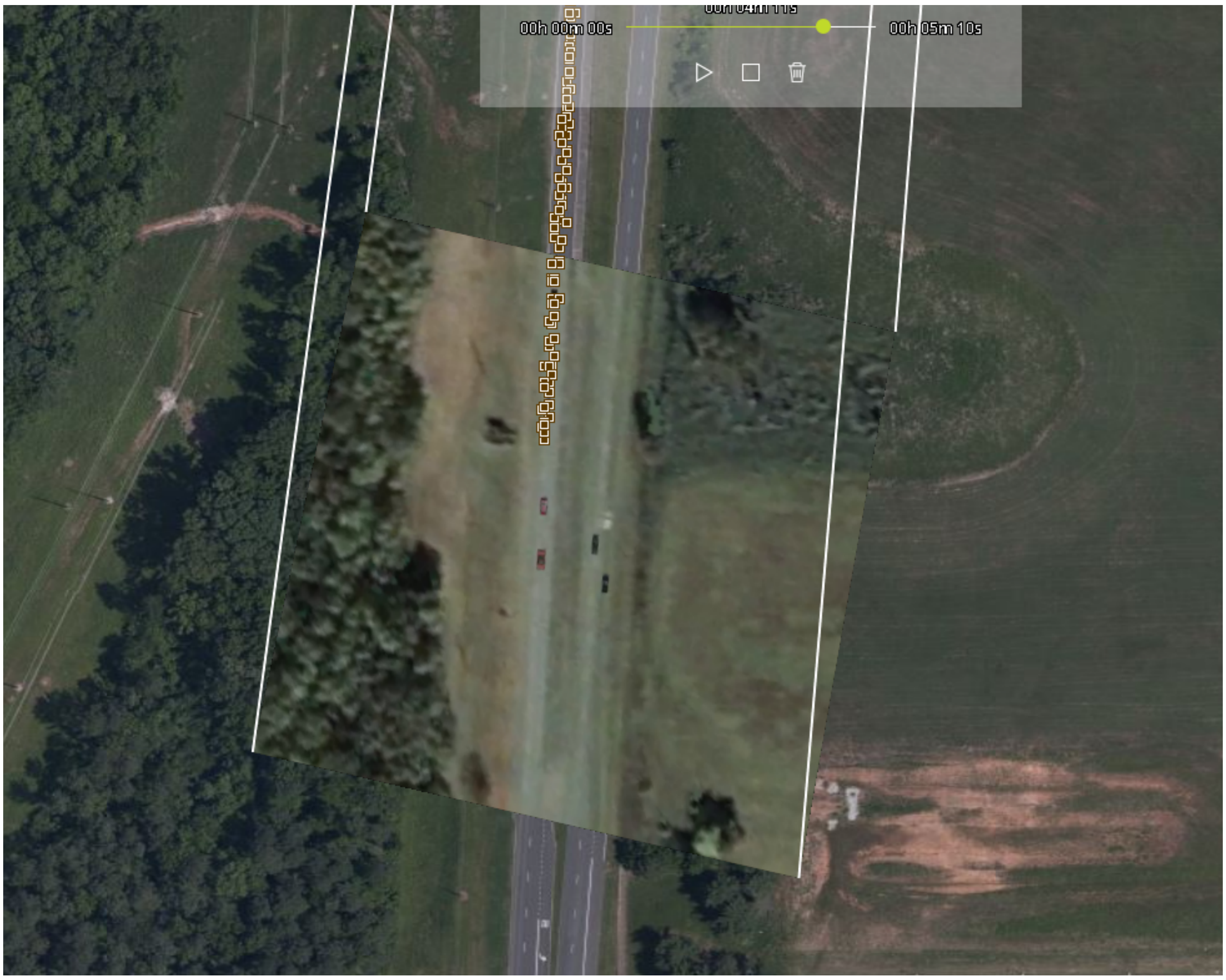


Figure 33. False negative detections.

The image above shows missing detections. In our tracker implementation, false negatives or missing values are handled by extending the track after it has been lost with constant bearing and speed in the hope to match it with a future detection. When color or intensity values are present, they can also help to match detections to previous tracks.

In both cases, it is up to the tracking algorithm to handle such cases.

9.3.2.3. detection jitter

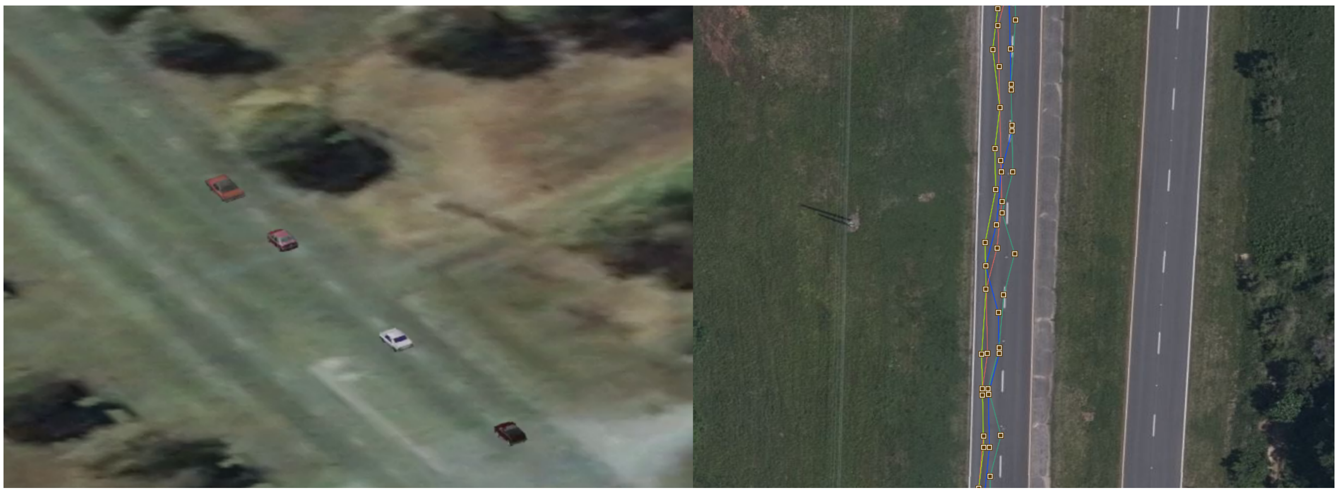


Figure 34. Detection Jitter.

The vehicles tracked in the image above travel in a straight line but the detections suffer from some jitter.

Detection jitter may be due to errors in the telemetry (the orientation, position and field of view of the camera) and jitter may be caused by the software. Either way, it makes the dataset more confusing to interpret. The data can be improved via smoothing methods.

One straight forward approach to reducing jitter is to apply some sort of non-linear regression through the points. The track is then interpolated over the regression to smooth it.

There is an argument against interpolating the tracks. The information gathered, though imperfect, is directly obtained from sensors while any interpolation method is a speculation on the actual location of objects.

A more drastic approach to solving the problem, specific to the use-case of objects travelling on predefined roads, is to match the detections with a vector model of roads. A routing algorithm may give more hints of the actual path that an object is traveling on by avoiding long detours for successive positions.

Chapter 10. Test Scenarios

10.1. Demonstrator 1 Test Scenarios

10.1.1. Overview

Testing of the MovingFeatures package was limited, largely due to a shortage of available sample data that is compliant with MISB and other North Atlantic Treaty Organization (NATO) standards.

10.1.2. Data

The difficulty of obtaining NATO standards–compliant FMV data files has, for both demonstrators, been the greatest shortcoming of the FMV-to-Moving Features initiative and its associated ability to be tested. Nonetheless, a varied collection of data files from two main sources were obtained.

One of the demonstrator's data sources is an archive of datasets provided by a public Google Drive folder. The data source comprised eight files containing Unmanned Air System (UAS) full-motion data recorded in various locations. Specific files and their locations were:

- **Cheyenne.ts** – Cheyenne, Wyoming
- **Cheyenne_Handoff.H264** – Cheyenne, Wyoming
- **CheyenneVAhospital.mpeg4** – Cheyenne, Wyoming
- **Esri_multiplexer_0.mp4** – Arcadia, Florida
- **Esri_multiplexer_1.mp4** – Arcadia, Florida
- **falls.ts** – Fall City, Washington
- **klv_metadata_test_sync.ts** – Silver City, North Carolina
- **Truck.H264** – Cheyenne, Wyoming

Although the videos' streaming formats differ, the metadata bytes' structure is identical, being comprised of UAS data with a top-level UAS universal key. As such, the MovingFeatures package can accept and parse these data sources in the same way.

Eight sample files were contributed by the United States Department of Defense's Joint System Integration Laboratory (JSIL). These files depict moving objects on various military bases. The eight specific files and the locations covered are:

- **S01.ts** – Dugway Proving Grounds, Utah
- **S02.ts** – Redstone Arsenal, Alabama
- **S03.ts** – Afghanistan
- **S04.ts** – Fort Huachuca, Arizona
- **S05.ts** – Kandahar, Afghanistan
- **S06.ts** – Huntsville, Alabama
- **S05_VTracks.ts** – Kandahar, Afghanistan

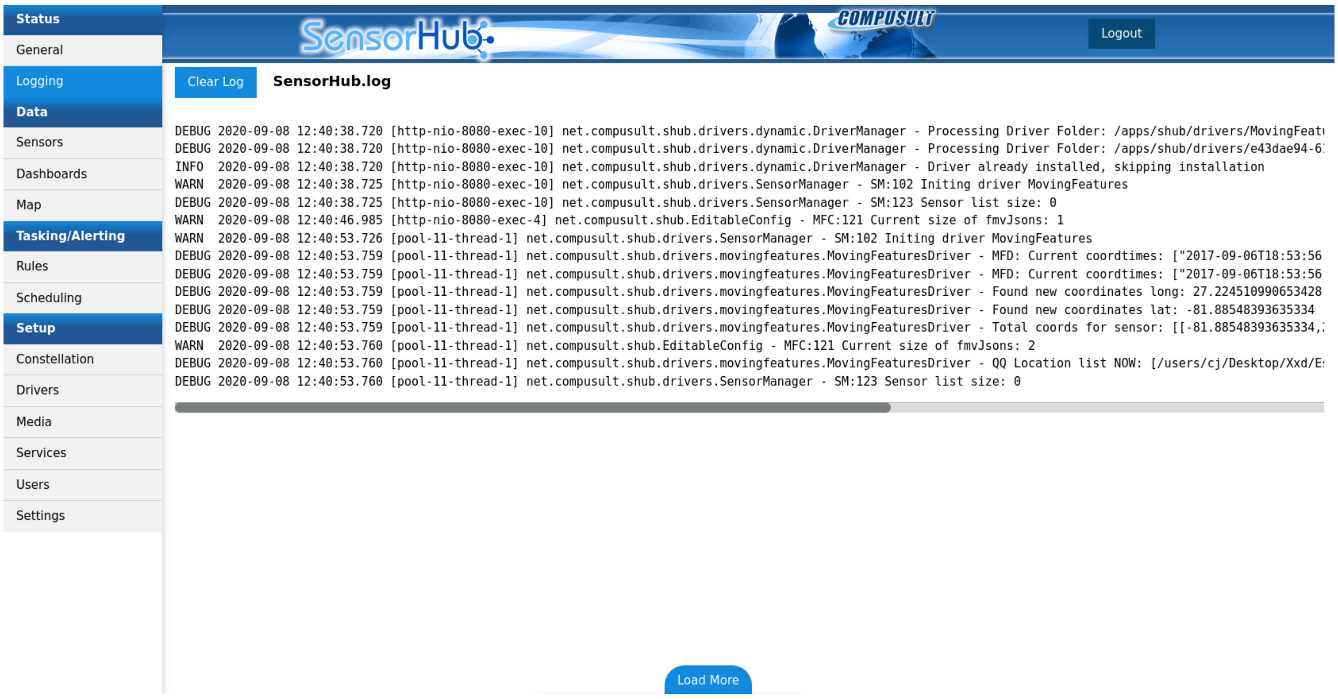
- **S06_VTrack.ts** – Huntsville, Alabama

As in the other sources, the top-level data sets in these files are all UAS and most use exclusively those keys defined in MISB 0601. Of these source files, only **S05.ts** and **S06.ts** contain VMTI data, and thus only they can be used as inputs to test certain use-cases of the WebVMT application. Moreover, even their VMTI sets are compliant with only the deprecated MISB 0903.3 standard and not the current 0903.5 revision.

S05_VTracks.ts and **S06_VTrack.ts** contain VTrack local sets as descendants of the top-level UAS ones. As such, they permit writing and testing of the MovingFeatures workflow that parses data out of this dataset type. They are also otherwise equivalent to **S05.ts** and **S06.ts**, meaning VTrack observations can be compared against the observations from these. The MISB 0903 in these files contains missing data, inaccurate detections and inaccurate telemetry to re-create a realistic situation.

10.1.3. Test Scenarios

Testing of the MovingFeatures package has thus far consisted mainly of ad-hoc unit-tests that arose as new modules were written. Feedback from the tests is simply available via Compusult's SensorHub through its built-in Apache Log4j logging system as well as the results conveyed to the dashboards.



```
DEBUG 2020-09-08 12:40:38.720 [http-nio-8080-exec-10] net.compusult.shub.drivers.dynamic.DriverManager - Processing Driver Folder: /apps/shub/drivers/MovingFeat
DEBUG 2020-09-08 12:40:38.720 [http-nio-8080-exec-10] net.compusult.shub.drivers.dynamic.DriverManager - Processing Driver Folder: /apps/shub/drivers/e43dae94-6
INFO  2020-09-08 12:40:38.720 [http-nio-8080-exec-10] net.compusult.shub.drivers.dynamic.DriverManager - Driver already installed, skipping installation
WARN  2020-09-08 12:40:38.725 [http-nio-8080-exec-10] net.compusult.shub.drivers.SensorManager - SM:102 Initing driver MovingFeatures
DEBUG 2020-09-08 12:40:38.725 [http-nio-8080-exec-10] net.compusult.shub.drivers.SensorManager - SM:123 Sensor list size: 0
WARN  2020-09-08 12:40:46.985 [http-nio-8080-exec-4] net.compusult.shub.EditableConfig - MFC:121 Current size of fmvJsons: 1
WARN  2020-09-08 12:40:53.726 [pool-11-thread-1] net.compusult.shub.drivers.SensorManager - SM:102 Initing driver MovingFeatures
DEBUG 2020-09-08 12:40:53.759 [pool-11-thread-1] net.compusult.shub.drivers.movingfeatures.MovingFeaturesDriver - MFD: Current coordtimes: ["2017-09-06T18:53:56
DEBUG 2020-09-08 12:40:53.759 [pool-11-thread-1] net.compusult.shub.drivers.movingfeatures.MovingFeaturesDriver - MFD: Current coordtimes: ["2017-09-06T18:53:56
DEBUG 2020-09-08 12:40:53.759 [pool-11-thread-1] net.compusult.shub.drivers.movingfeatures.MovingFeaturesDriver - Found new coordinates long: 27.224510990653428
DEBUG 2020-09-08 12:40:53.759 [pool-11-thread-1] net.compusult.shub.drivers.movingfeatures.MovingFeaturesDriver - Found new coordinates lat: -81.88548393635334
DEBUG 2020-09-08 12:40:53.759 [pool-11-thread-1] net.compusult.shub.drivers.movingfeatures.MovingFeaturesDriver - Total coords for sensor: [[-81.88548393635334,
WARN  2020-09-08 12:40:53.760 [pool-11-thread-1] net.compusult.shub.EditableConfig - MFC:121 Current size of fmvJsons: 2
DEBUG 2020-09-08 12:40:53.760 [pool-11-thread-1] net.compusult.shub.drivers.movingfeatures.MovingFeaturesDriver - QQ Location list NOW: [/users/cj/Desktop/Xxd/E
DEBUG 2020-09-08 12:40:53.760 [pool-11-thread-1] net.compusult.shub.drivers.SensorManager - SM:123 Sensor list size: 0
```

Unit tests conducted on the MovingFeatures package, in chronological order, included the following.

1. The **BERParser** helper class needed to be checked to ensure that it correctly interpreted Basic Encoding Rules (BER) and Basic Encoding Rules for Object IDentifiers (BER-OID)-encoded numbers, and that the **Driver**'s read identified and skipped over any initial bytes (e.g., the "82" in 0x8201A1, which really encodes the length 0x01A1) that give the length *of* the length value itself.
2. The **MovingFeaturesKeys** helper class was set up and checked to make sure that each one-byte

key that could produce relevant observations was recognized by the **switch** statements in **parse()**, which used them as cases.

3. Each JSON creator function was tested to make sure it placed the right **temporalProperty** or **property**, as appropriate, in the new MF-JSON object; in particular:
   a. Creators for keys such as the “Platform Designation” and “Version Number” were checked to ensure that they appropriately translated their ASCII hex bytes into strings;
   b. The creator for the timestamp, which is the same for MISB 0601 and 0903, was checked to ensure that it translated its bytes into a number of milliseconds since the Unix epoch and that this got housed in the timestamps arrays; and
   c. **decodeUds()**, the creator for any numerical observations encoded in Universal Data Set (UDS), was checked against the UDS specs in the MISBs to make sure the numbers it was generating were correct. UDS-encoded observations include the Latitude and Longitude.
4. **Config** was checked to ensure that the reader’s byte offset was saved correctly between one poll and the next, preventing the same frame from being reread.
5. An internal lookup table, housed inside properties for convenience, was needed in order for sensors to find themselves, since a JSON file can comprise multiple sensors but each sensor has a reference to the entire file. Made sure lookup table worked via logging and dashboards.
6. Dashboard widgets were checked to make sure the endpoints of their ranges matched those that were defined in **MovingFeaturesKeys** and in the MISBs. For example, Height **GaugeWidget**s generated from UAS must support displaying heights between -900 and +19000 feet.
7. Widgets were checked to make sure the coordinates they were reporting for themselves were up-to-date with their <u>latest</u> observed locations.

Chapter 11. Discussion

11.1. OGC Moving Features

OGC Moving Features is adequate as an interoperability format to communicate tracks that were extracted from MISB 0903. All of the metadata contained in MISB 0903 and inferred metadata such as track classification or rigid shape can be transported by the format.

The only limitation is for the MISB 0903 VObject Local Data Set (LDS), VFeature LDS and VChip LDS. They describe a class as Web Ontology Language, Features as OGC O&M and frame images respectively. OGC Moving Features is sufficiently flexible to transport this information but by doing so looses its interoperability quality because the encoding would require a specific schema that is not yet defined.

11.2. OGC Observations and Measurements

The general impression is that OGC O&M can be used as an interoperability format as it has the necessary components for the use-cases that have been considered. The representation of geometry through GeoJSON and the temporal metadata structure are adequate relative to what has been tested for OGC Moving Features. However, an encoder could not be implemented and the assessment is still largely speculative.

11.3. Future work recommendations

11.3.1. OGC Observations and Measurements

OGC O&M was extensively reviewed but not implemented through a working encoder and decoder of moving tracks. A more thorough investigation of OGC O&M is needed to conclude on that standard's usability as an interoperability format for tracks exported from MISB 0903 and post-processed for various use-cases.

11.3.2. SensorHub Future Work Components

11.3.2.1. Visual Location Reporting via SensorHub

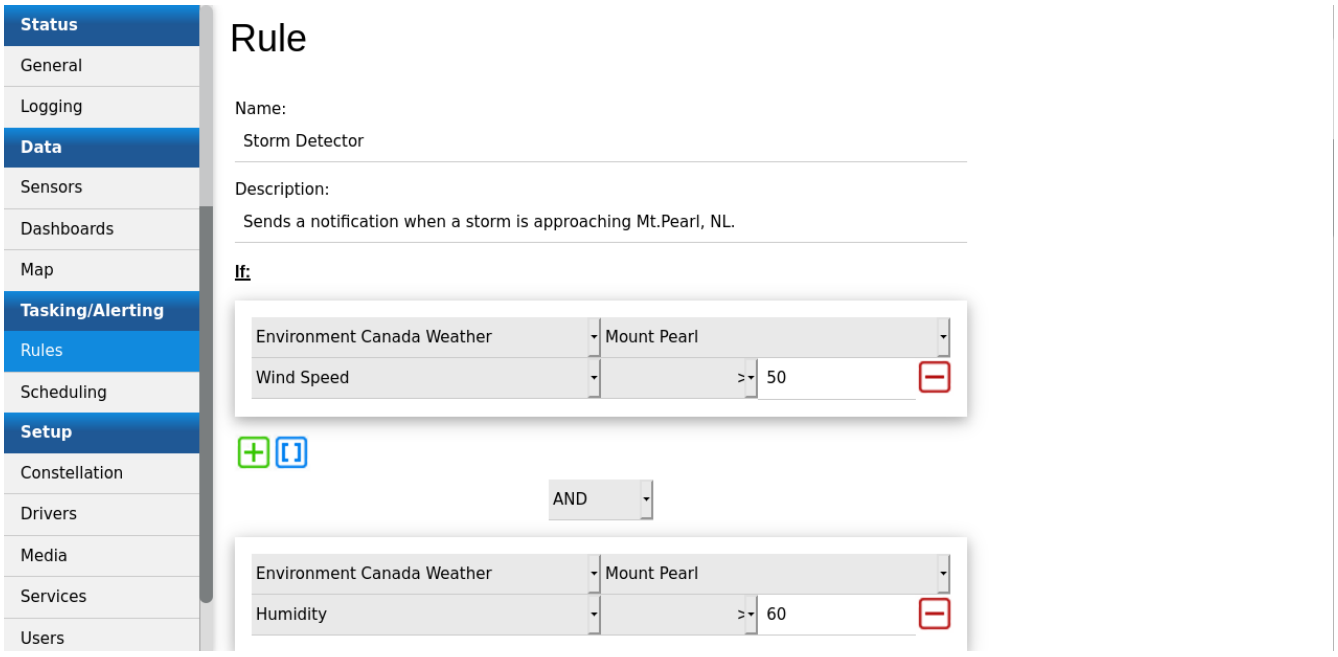
An envisioned use case for SensorHub is to plot the locations of moving items on a map. Were this utility completed and coupled to the **MovingFeaturesDriver**, this capability would allow a visual and more intuitive representation of SensorThings **Historical Location**s (coordinate pairs) over time, similar to the WebVMT contributor's map.

11.3.2.2. UDP Support

In real-world applications, FMV livestreams are commonly broadcast to control systems in the form of User Datagram Protocol (UDP) packets to ensure real-time transfer. The UDP protocol is not natively supported by SensorHub due to its lack of a UDP socket in its data model. If a receiver for this protocol were added, the **MovingFeaturesDriver** would be extensible to a wider range of applications.

11.3.2.3. Tasking API

As discussed in the Section 4 overview, the OGC SensorThings API permits not only fetching of observations by clients, but also automatic, immediate alerting and broadcasting to clients if abnormal observations are detected. This side of the API – its "tasking" component – is supported in the SensorHub web application mainly through the MQTT broker that is contained in the SensorHub core. While SensorHub is running, by default, a broker is gathering data on topics for each **Datastream**, **Observation**, and other entity that is contained in the app. Rules can be configured in the central portion of the app to broadcast a specific message when a datastream gives an observation in a certain range (for example, if Height exceeds 10,000 ft, "Extra oxygen needed" could be sent to mountain climbers' safety sensors). Observers who have MQTT client programs can then subscribe to these topics at will and be automatically notified if the conditions of any of the rules are met. While rules *can* be added to MovingFeatures streams, the SensorHub tasking interface is still in its early stages of development.



Rule

Name:
Storm Detector

Description:
Sends a notification when a storm is approaching Mt.Pearl, NL.

If:

Environment Canada Weather | Mount Pearl
Wind Speed | > | 50

AND

Environment Canada Weather | Mount Pearl
Humidity | > | 60

Figure 35. SensorHub tasking interface.

11.3.2.4. UI and Widget Specifications

SensorHub dashboards support a wide range of widget appearances, not only the **GaugeWidget**s shown in current MovingFeatures observations. For example, wind speeds on meteorological sensors have been displayed using **SpeedometerWidget**s like the one below:

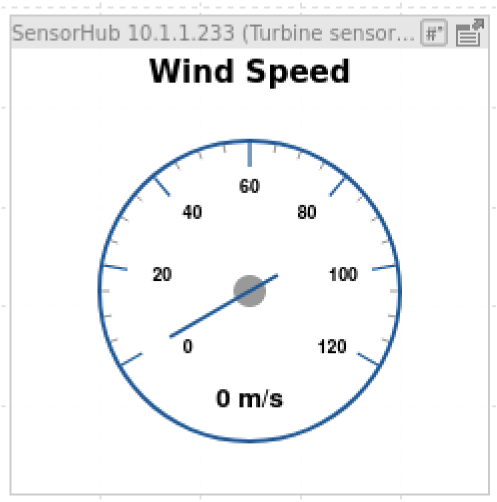


Figure 36. Speedometer widget.

The main current barrier to diversifying the widgets in this Driver is that, as per the MovingFeatures Sensor standards, every datastream must be able to determine all of its details solely by reading its own JSON. A **temporalProperties** entry, as defined in the MovingFeatures Standard, is only permitted to store its value and its unit of measurement (UOM), not additional details such as what alternative widget to use. A lookup table associating either key names or units with widget types would likely be the simplest possible way to implement alternative widgets. This would have to be separate from the **MovingFeaturesKeys** class, which does not interact with the **Sensor**.

11.3.3. WebVMT Future Work Components

WebVMT work in Testbed-16 FMV produced successful results which could be developed further and identified new topics worth investigating in the future.

11.3.3.1. Web Browser DataCue Integration

Work is progressing in parallel at the W3C in the Web Incubator Community Group (WICG) DataCue activity [https://github.com/WICG/datacue/] to integrate the DataCue element into HTML and expose video metadata, including geospatial information, natively in web browsers. Building an OGC demo with DataCue support would:

1. Guide requirements for geotagged video on the web and ensure that proposed W3C solutions are aligned with OGC standards such as Moving Features and SensorThings to maximise interoperability.
2. Encourage browser implementers to provide support for geotagged video online which enables timed metadata integration with search engines, digital maps and web applications.
3. Help accelerate geospatial DataCue feature development in the web community.

A breakout session [https://www.w3.org/wiki/TPAC/2020/SessionIdeas#Video_Metadata_For_Moving_Objects_.26_Sensors_On_The_Web_.28WebVMT.29] to discuss a web API suitable for moving objects and sensor data with video on the web was proposed for the W3C Technical Plenary and Advisory Committee (TPAC) in October 2020 to raise awareness in the W3C community and help gather support.

11.3.3.2. Metadata Export Utility Enhancement

The WebVMT export utility has successfully mapped MISB metadata to various WebVMT features, but is currently limited to MPEG-2 Transport Stream input files containing MISB 0601 and 0903 metadata. This could be expanded to include other formats:

1. For video:
   - MPEG-4 is a primary focus for media on the web.
   - Other web video formats such as MPEG-H [https://en.wikipedia.org/wiki/MPEG-H], AV1 [https://en.wikipedia.org/wiki/AV1], WebM [https://www.webmproject.org/] and OGG [https://xiph.org/ogg/].
2. For geospatial metadata:
   - More MISB tag mappings for specific use cases;
   - Open metadata formats such as GoPro Metadata Format (GPMF) [https://github.com/gopro/gpmf-parser] for body-worn video devices;
   - SubRip Text (SRT) format files used by drone manufacturers.

11.3.3.3. Metadata Export Benefit Analysis

Metrics from the WebVMT export utility could be analyzed to identify the benefits of using in-band and out-of-band video metadata both qualitatively and quantitatively to determine where best advantage can be gained. Key topics include:

1. Privacy to control data access with file permissions.
2. Security to monitor online access patterns and correctly identify unauthorized data use.
3. Efficiency to assess overheads involved in accessing metadata.
4. Bandwidth to compare data throughput during common use cases such as web crawling and search engine queries.

11.3.3.4. Search Engine Demo

Build a modified search engine based on open source code to demonstrate search by location and/or sensor data capability that returns either still video frames or video clips centered around the matching media time. Applications include the Vehicle Collision [https://w3c.github.io/sdw/proposals/geotagging/webvmt/#vehiclecollision] and Area Monitoring [https://w3c.github.io/sdw/proposals/geotagging/webvmt/#areamonitoring] use cases detailed in the WebVMT Editor's Draft would benefit from automated collision detection, preserved evidence, reduced bandwidth and structured security. The demo provides:

1. Proof of concept for end-to-end geospatial integration from source video file to web browser display.
2. Reference code to test and verify issues of privacy, security and bandwidth associated with online search, and gather metrics.

11.3.4. Deep learning for tracking and track classification

A Deep Learning approach for point clouds can be applied to the VMTI tracking problem. In theory, a network like PointNet++ or similar can segment the detections into tracks and handle the problem of missing detections and false positives. Such a tracking algorithm can also be trained with varied datasets so that the tracking algorithm does not need to be tuned to the type of object it is tracking. Moreover, a point-cloud deep-learning network can be used as a classifier to identify abnormal behaviors such as drunk-driving or sea piracy.

Such an algorithm can also play a role in organizing a library of videos by identifying videos with similar content.

Training these algorithms requires considerable data and building a database of tracks with a classification schema would be invaluable.

11.3.5. Real-time traffic monitoring

Using a fleet of autonomous sensors to monitor the volume and velocity of traffic as it flows through city streets. The collected data is the moving feature representation of all moving vehicles. Collection will go through motion imagery sensors capable of generating VMTI. Note that only the VMTI needs to be collected. The motion imagery can be discarded once the motion data has been extracted.

The result is a collection of moving features which represent the actual traffic during a period of time. This would allow detailed modeling of traffic behavior within an urban setting. The model would include not just the aggregate properties, but also the behavior of individual vehicles within a flow. Similar capabilities have been discussed in the context of urban planning. See https://portal.ogc.org/files/?artifact_id=82917 for one example.

11.3.6. Counter UAV use-case

This use case focuses on detecting a small UAV in video streams, estimating the position of that UAV and then generating MovingFeatures data with the position and trajectories. Multiple Video streams would be needed to triangulate the position of the object.

11.3.7. Drone Swarms

A variety of algorithms are being developed to control the movement of drone swarms. Drones within the flock adapt their location based on the observations that are being made by the collective of sensors on the UAVs and this may call for a new standard.

Appendix A: Revision History

Table 2. Revision History

Date	**Editor**	**Release**	**Primary clauses modified**	**Descriptions**
October 9, 2020	E. Beaufays	.1	all	Initial Version
October 27, 2020	E. Beaufays	.2	all	First Review

Appendix B: Bibliography