Digital Analytics and Robotics for Sustainable Forestry

CL4-2021-DIGITAL-EMERGING-01
Grant agreement no: 101070405

DELIVERABLE 4.1

Report defining data format and software API for multi-robot map sharing

Due date: month 6 (February 2023)
Deliverable type: R
Lead beneficiary: UOXF

Dissemination Level: PUBLIC

Main author: Nived Chebrolu, Maurice Fallon

Contents

1 Data Format for Shared Maps

1.1 Introduction

The goal of this document is to describe a common data format for sharing and exchanging maps built by multiple robots. The need for a common data format arises from the fact that each robot typically runs a customized SLAM (Simultaneous Localization and Mapping) system tailored for its sensor suite and maintains an internal map representation. This makes it difficult to exchange and merge mapping data collected from different platforms. Therefore, a common data format and conventions need to be established to achieve interoperability.

The common data structure, referred to as the "map payload" or "payload" in the document, forms the central unit that will be exchanged between robots. This map payload is expected to be the primary data unit consumed by downstream tasks for further processing (See Fig. 1).

The map payload defined in this document is designed to be compatible with the pose-graph map representation described in D3.1. This compatibility ensures that data collected by different robots and over multiple missions can be merged in a consistent manner.

The technical details of the map merging procedure will be provided in D3.1 and D4.2. This will enable a seamless exchange and merging of mapping data collected by different robots.

In addition to defining the common data format, the document also outlines software API guidelines for accessing and retrieving the data for downstream processing. These guidelines will ensure that the map payload is accessed and used in a consistent manner.

In summary, the document establishes a common data format for sharing and exchanging maps built by multiple robots. The common data format, referred to as the map payload, is designed to be compatible with pose-graph map representation, ensuring that data collected by different robots and over multiple missions can be merged in a consistent manner. The document also outlines software API guidelines for accessing and retrieving the data for downstream processing.

1.2 Payload as Common Data Exchange Unit

1.2.1 What is a payload?

The payload is established as the common data exchange unit for sharing data between multiple robots. A payload is created as an accumulation of sensor data and other derived metrics in a local area. The data is accumulated or stitched together using the onboard odometry system on each platform. Once a payload unit is created, it decouples the downstream tasks from the SLAM/Odometry system used to create them.

Typically, the odometry system on each platform uses the live sensor data to estimate the robot's motion and generate a payload representing the local area around the robot. Once a payload unit is created, it becomes a standalone data unit that contains all the necessary information about the local environment that it covers. This information can include the raw sensor data used to create the payload, locally accumulated data as well any other relevant derived metrics and other metadata.

For example, a typical payload from a ground robot such as ANYmal would consist of raw sensor data such as camera images, IMU, GPS measurements as well as

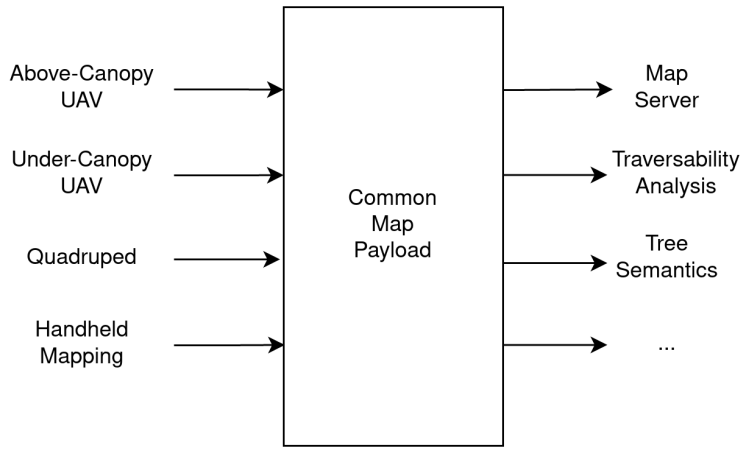

Figure 1: Each platform/mapping system generates an output using the common payload format. This enables downstream processes to treat the input data uniformly decoupling it from the source.

accumulated sensor data such as a local pointcloud map spanning an area around the robot. In addition, the payload also supports further derived information such as a terrain map, occupancy information etc. that may be computed online during robot operation. Essentially, the payloads allow us to group local information efficiently and decouple the downstream tasks from the SLAM/Odometry system used to create it.

In addition to enabling downstream processing, the use of a common payload format allows for seamless sharing and merging of mapping data collected by different robots. By standardizing the payload format, different robots can exchange their mapping data without needing to understand each other's unique data structures or SLAM systems. This helps to reduce the overall complexity of the data exchange process, allowing for more efficient and effective sharing of mapping data.

1.2.2 What are the payloads designed for?

A payload is designed such that it encapsulates a data unit covering a certain fixed spatial area. The spatial area covered by an individual payload unit can be set based on platform specific requirements such as the computational capacity or the size of the data transfer channel.

As the payload is typically generated at a much lower frequency than the raw sensor data, it makes the payload data stream amenable to online processing tasks .This makes it suitable for detailed analysis tasks such as semantic analysis, computing individual tree properties, or generating area-level statistics.

In addition to online processing, the payload unit can also be used for offline processing tasks. Since each payload unit contains all the necessary information about a fixed spatial area, downstream processing can be performed independently of the source or platform from which the raw data has been captured. This makes it easier to develop algorithms and applications that are independent of the underlying sensor suite or SLAM system. The payload unit also enables efficient storage and retrieval of mapping data, allowing for easier management and sharing of mapping data across different platforms and systems.

Overall, the use of a payload as the common format for exchanging data provides

numerous benefits, including standardized data units, flexibility in data management, and efficient processing and analysis of mapping data. This makes it a valuable tool for enabling interoperability and collaboration between multiple robots operating in the same environment.

1.2.3 How is a payload unit created?

The usual operation of creating payload is done onboard the robot or the mapping device in an online fashion. As an example to understand the payload creation process, we describe the process for creating a payload unit for a handheld laser scanning system.

Creating a payload unit for a handheld laser scanning system involves a series of steps. First, the space to be scanned is divided into smaller areas to create a fixed spatial coverage for each payload unit. In this case, each payload unit covers an area of 20mx20m, as shown in Fig. 2, with each payload unit illustrated in a different color. The payload unit is a dense accumulation of scans that is the locally consistent output from laser odometry.

To maintain the alignment of the payload data in the overall map, each payload unit is attached to a unique pose, chosen at the center of the area covered. The pose attached to the payload data allows it to be re-aligned or adjusted by the map server using the pose-graph optimization framework described in D4.2. Although the pose information is typically not directly required by downstream tasks such as extracting tree semantics or

The payloads can be considered as an atomic unit of a scanning mission. For instance, 1000m scanned distance can result in 100 payload units. The payload units can be written to disk and reprocessed by software modules developed by partners.

Furthermore, any processed product can be computed on a payload unit and can be merged with the original payload unit as an additional layer of information. For example, additional information could be individual tree instances and other attributes that have been computed using the payload data. This approach ensures that the payload data can be reused and repurposed for different downstream tasks. An example of the downstream task of estimating individual tree instances and the local terrain information is described in Sec. 1.3.

1.2.4 How is the payload integrated with the pose-graph defined in deliverable D3.1?

The pose graph-based adjustment and optimization procedure defined in D3.1 allowings for re-alignment of the pose in case the constraints are updated at a later stage. The updated constraints can arise from further offline processing resulting in new loop closures, merging with another overlapping SLAM pose-graph obtained from a different mission from the same robot, or merging of the pose-graph created by another robot. In each of these cases, we would be able to re-align the payload data to be consistent with the new observations/constraints that may be added.

As illustrated in Fig. 3, the payload data is attached directly to a node in the pose-graph structure. This is ensured by creating the payload data having the pose node as its reference origin. Typically, the payloads are created at a much lower frequency or a larger spatial distance in between consecutive payloads, and therefore only a subset of the pose-graph nodes will have a payload attached to them. Note that along with the raw sensor data, all the additional information layers attached to the payload will also be re-aligned and globally consistent.

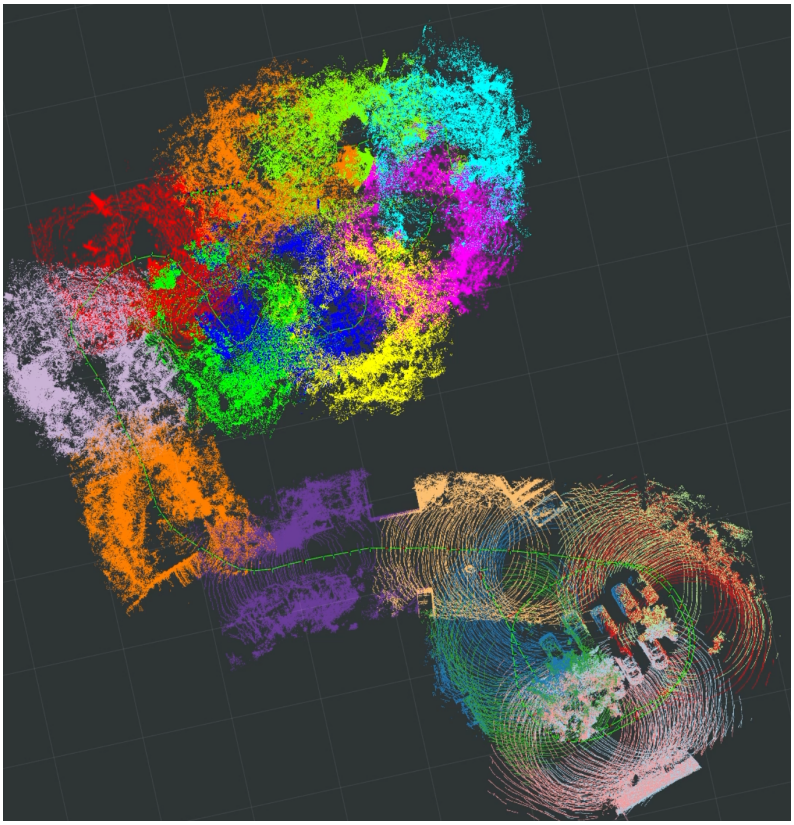

Figure 2: Individual payloads visualized in different colors. Note that the payload units are created using the odometry output as the robot/sensor moves in the surroundings. As a result, the payloads may be spatially overlapping.computing other attributes from the payload data, it is important to maintain the spatial information of the payload unit for accurate mapping.

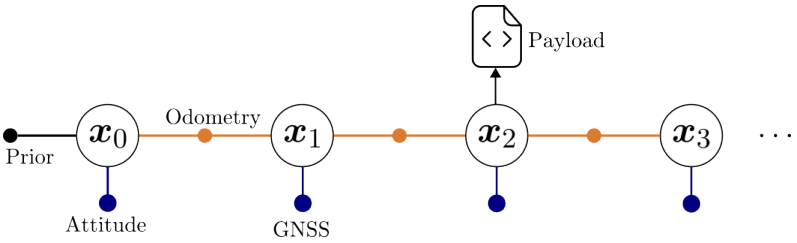

Figure 3: Payload integration with the pose graph structure defined in D3.1. This allows the payload data to be deformed/re-aligned along with the node it is attached to. Each payload unit is associated with a corresponding node in the pose graph at the time of its creation. The origin of the data in the payload is defined by the pose of the corresponding node, allowing for a spatial transformation of the data to occur. This transformation is crucial for ensuring the compatibility of the payload with the pose-graph optimization framework outlined in D3.1. By associating each payload unit with a node in the pose graph, it becomes possible to adjust and optimize the payload data in a consistent manner.

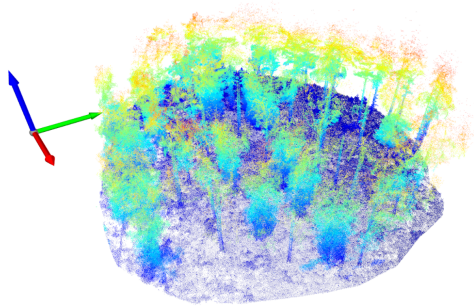

Figure 4: Input point cloud payload from a handheld mapping pipeline. This payload is generated online and written to disk for downstream processing.

1.3 Example Application of Payload Processing: Tree Instance Segmentation

In this example application, we create payloads containing aggregated point clouds onboard the Frontier handheld device. The point clouds from the payloads are then stored offline after the SLAM mission (as .pcd or similar files) and can be loaded for downstream tasks, for example semantic analysis or tree instance identification. Fig. 4 shows an example payload and its use is demonstrated next for an instance segmentation pipeline. First, in a sequence of preprocessing steps, the ground is segmented – highlighted in red in Fig. 5 – and subsequently removed. To account for possible elevation changes, the local ground height information obtained previously is used and the point cloud is height normalized by bringing all the trees onto a "level" field. The final result after preprocessing is shown in Fig. 6 . Finally, Fig. 7 then shows the result of a geometric density based clustering algorithm used to perform tree instance segmentation. Each identified tree is assigned a unique ID (per patch) which can be used for further detailed tree-level analysis. This result can then be stored as an additional layer of information in the payload.

1.4 Map Tessellation

1.4.1 Why do we need map tessellation?

The payloads created during the robot's movement in the surroundings, as shown in Fig. 2, may overlap depending on the trajectory. However, in certain offline tasks that involve analyzing a spatial area in the forest, it is useful to have a unique payload unit corresponding to the physical space. This is particularly important for temporal analysis and monitoring of the same area in the forest. By having a unique payload unit corresponding to a physical space, the analysis can be performed on the same area over multiple missions without the need for additional alignment or transformation steps.

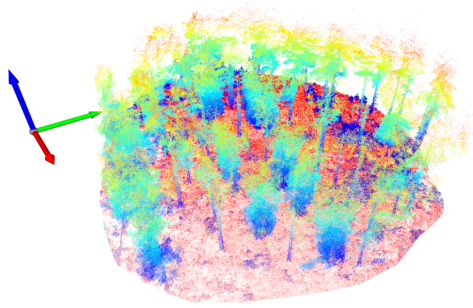

Figure 5: Ground segmentation on the payload. Points in shades of red represent the ground plane.

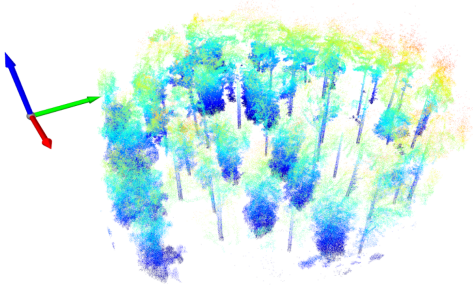

Figure 6: Ground removal and height normalization as further pre-processing steps.

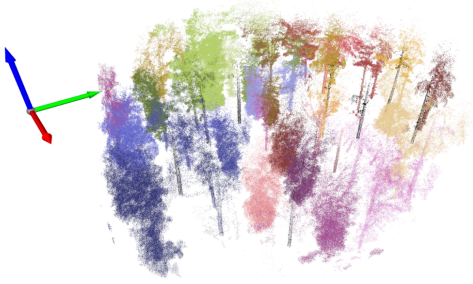

Figure 7: Result of Tree Instance Segmentation algorithm on the payload patch. Each tree instance is represented in a different color.

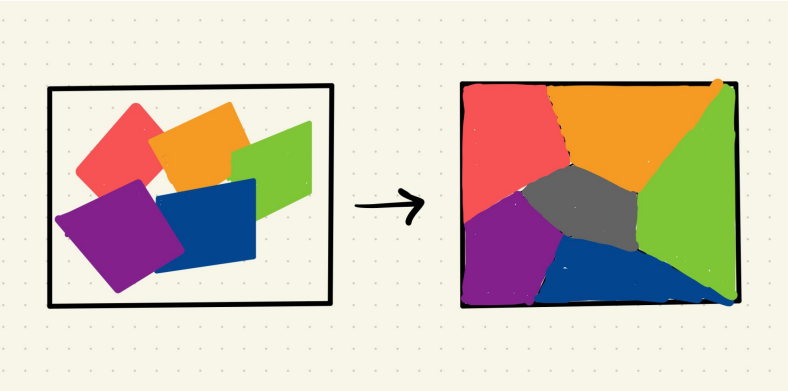

Figure 8: Spatially overlapping payloads can be tessellated such that they are non-overlapping. This may be desired for certain downstream tasks such as area monitoring or growth analysis.

1.4.2 How can we compute a unique tessellation?

To enable monitoring of spatial areas in the forest, the map data can be partitioned into non-overlapping partitions, with each partition being associated with a payload data unit. This partitioning is done as a post-processing step and the original payload units are still stored in case they are preferred for certain applications. An illustration of the non-overlapping partitioning scheme is shown in Fig. 8.

There are several schemes for partitioning the map data into non-overlapping partitions. One approach is to use voronoi segmentation based on the corresponding payload pose to crop away or merge overlapping points. This method ensures that each partition contains only the data within the spatial area covered by the payload unit associated with it. Another approach is to partition the map into a grid structure of a desired size, resulting in "stands" that are typical in forestry mapping. This approach provides a simple and standardized way of partitioning the data, with each partition being a fixed size.

The choice of partitioning scheme depends on the specific requirements of the application. Voronoi segmentation is suitable when there is a need to create partitions that follow the contours of the terrain, and when the size and shape of the partitions can vary. Grid-based partitioning, on the other hand, is suitable when a fixed and standardized size of partitions is desired. Both methods provide a way to create non-overlapping partitions of the map data, which are associated with payload data units for efficient offline analysis and monitoring of spatial areas in the forest.

1.5 Full Data Pipeline for Forest Operations

1. **Robot Operation:** Robots move through a forest. Each robot typically runs their own odometry/SLAM systems and builds up maps. The maps are stored in a pose graph representation. This allows us to support robots without GNSS and with poor GNSS reception within the forest. In that situation, maps would be limited to a local coordinate/map frame. Robots can and will use data from prior operations to do pre-planning.

2. **Raw payload:** The data output by the robots is in a general pose graph format and payload data that includes raw sensor measurements and any other

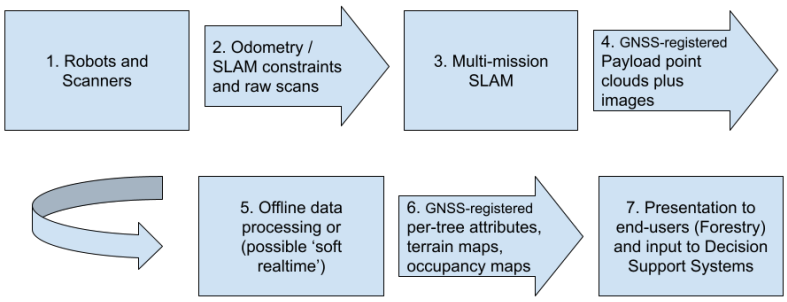

Figure 9: Complete Data Pipeline. Boxes correspond to algorithms or applications. Arrows correspond to data. Here, we describe a complete operation scenario of forest mapping with multiple robots and emphasize the role played by payloads in the full data pipeline.

information estimated onboard. These payload blocks would be created after traveling a certain distance, eg. 20m of travel as shown Fig. 2.

3. **Multi-mission SLAM:** If there are multiple missions which physically overlap and/or if GNSS/RTK is only available in post-processing, missions from different times or robots can be merged together to progressively map several hectares.

4. **Offline data processing:** This involves all data processing pipelines which extract relevant data from the raw sensor data, primarily the point clouds and camera images. The tasks include individual tree segmentation and parameter extraction, terrain mapping, species classification, trunk quality assessment, understory growth estimation, semantic segmentation, traversability maps for robot navigation etc (WP4). Note that certain types of algorithms could be run online as the payloads would be produced at a lower rate, for example one payload every 20m traveled.

5. **Enriched payloads:** The payload data at this point now contains per-tree attributes and the types of labeling which would be output by the above data processing. Conversion to regularly sized grids and GNSS aligned data may be performed at this stage (WP4).

6. **Presentation to Foresters:** At this point the data would be well organized and higher level attributes could be presented to foresters or provide inputs to DSS such as growth modeling (WP6)

1.6 Payload Data Organization for Offline Storage

1.6.1 What data formats will be supported for different sensors?

As a minimum set, we aim to support the following file formats for offline storage of raw sensor data. In addition to the raw sensor data, additional derived metrics computed online can be stored as well. The file formats for the derived metrics will vary and file access methods for this data will be provided by the partner generating the data.

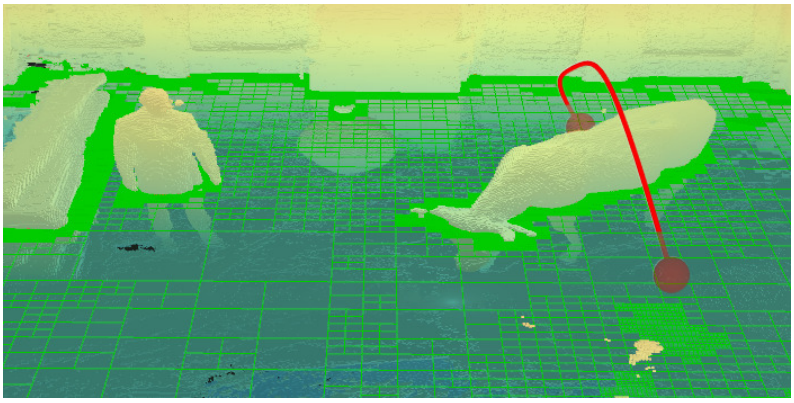

Figure 10: Example of a volumetric map generated with the Supereight2 library, a framework created by the Smart Robotics Lab from TUM.

Sensor	Supported File Formats	Attributes
Laser Scanner	.pcd, .ply, .las	Point positions, normals
RGB Camera	.png	RGB images
Depth Camera	.png, .pcd, .ply	Depth images/pointclouds
GNSS Receiver	.txt	latitude, longitude, altitude, Signal quality metrics
IMU	.txt	Acceleration, biases

1.6.2 What type of map formats will be supported?

In addition to the raw sensor data, a payload unit can also support maps that may be created either online or offline. The simplest representation is a pointcloud map of the local area defined with its origin coincident with a node in the pose-graph map. Examples of such pointcloud maps are illustrated in Fig. 2. The payload also supports elevation maps and volumetric map representations that are typically computed onboard the robot platforms for navigation and path-planning. These volumetric map representations may encode occupancy information, surface information using Truncated Signed Distance Functions (TSDF) etc. An example of a volumetric map is shown in Fig. 10.

1.6.3 How is the payload data organized on disk?

The payload data for the mapping mission is stored on the disk in a hierarchical manner. The data is organized based on the sensor data type and associated with the corresponding node in the pose graph with the timestamp as the identifier. An example of how the payload data is organized is illustrated in Fig. 11.

2 Software API for multi-robot map sharing

This section describes software API guidelines for accessing payload data in both online and offline mode.

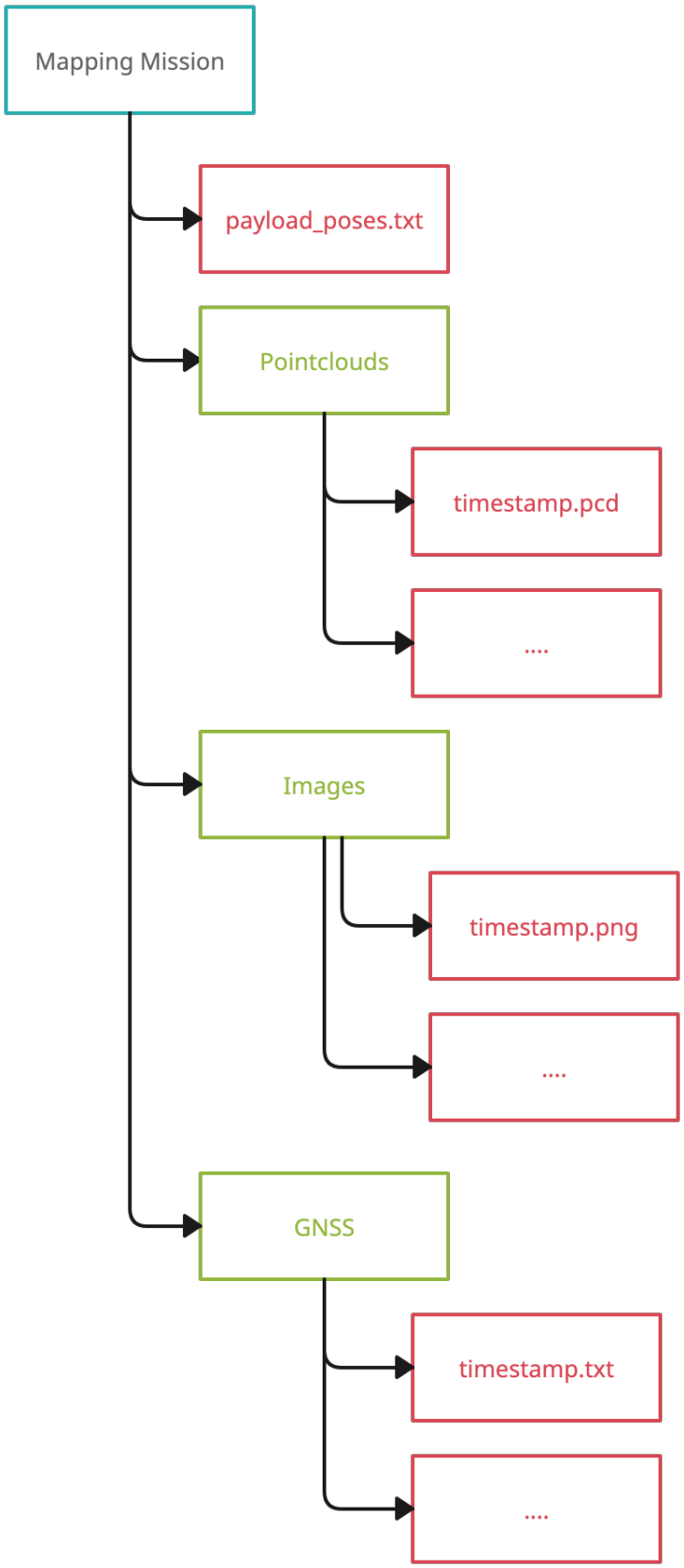

Figure 11: Folder structure showing the payload data organization for offline storage.

2.1 Payload access for online processing

During online operation, we leverage the ROS infrastructure for data transfer and communication between different programs, also referred to as nodes, running on the robot. For low data rate applications, ROS based communication via WiFi can be used in-between multiple robots or between a robot and an operator computer.

We use standard ROS messages to publish sensor data. This data is broadcasted over the ROS network and is available to all programs that subscribe to this data. Some of the commonly used sensor data types across the robot platforms are:

- sensor_msgs/Pointcloud2 : Supports point cloud data generated by LiDARs and Depth cameras
- sensor_msgs/Image: Supports multiple image types including onboard compression for efficient data transfer
- sensor_msgs/Imu: Supports high frequency IMU measurements
- sensor_msgs/NavSatFix: Supports GNNS position and additional metadata.
- nav_msgs/Odometry: For robot pose information

In addition to raw sensor data, the payload units may also include derived information computed online such as local elevation maps etc. For exchanging such data, custom data types will be defined using ROS custom message system. These message definitions will be shared between all project partners.

2.2 Payload retrieval for post-processing

After a SLAM mission, the payload data is stored to the disk according to the hierarchical organization described in Sec. 1.6.3. For processing the payload data offline, we provide some guidelines for the software API.

- Retrieve payload data by type

retrieve_payload_by_type(mission_folder_path, data_type) -> payload data
Input: path to mission folder on the disk, data type to retrieve, eg. pointcloud
Output: payload data as relevant data structure
Requires: SLAM pose-graph for the mission as described in deliverable D3.1

- Retrieve payload data by SLAM pose

retrieve_payload_by_pose(mission_folder_path, slam_pose) -> payload data
Input: path to mission folder on the disk, SLAM pose for which data is requested
Output: payload data as relevant data structure
Requires: SLAM pose-graph for the mission as described in deliverable D3.1

- Retrieve payload data by GNSS position

retrieve_payload_by_gnss(mission_folder_path, slam_pose) -> payload data
Input: path to mission folder on the disk, GNSS coordinates for which data is requested
Output: payload data as relevant data structure

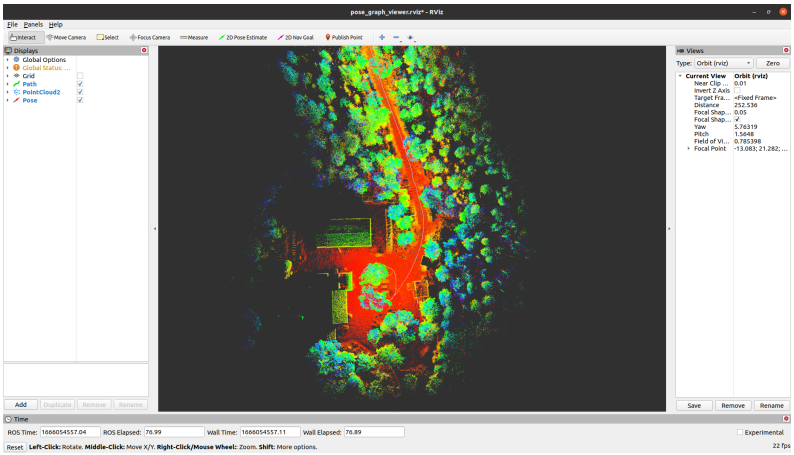

Figure 12: Visualization of Payload data of ROS supported formats with RVIZ GUI. We provide RVIZ based tool (Fig. 12) and configuration files to visualize basic payload data such as point clouds, images, robot trajectories etc. The RVIZ visualizer subscribes to the sensor data topics and visualizes the stream of data. The visualizer relies on the sensor data to be ROS compatible. This tool may be used both for visualizing data online or in a playback fashion. In the future, the visualization tool will be extended to multiple missions. If geo-referenced payload data is available, then it can be visualized as on overlay on satellite imagery.

Requires: SLAM pose-graph with GNSS reference for the mission as described in deliverable D3.1.

Note that in this section, we only provide guidelines for the API. The concrete implementation of the API will be done in C++ and Python over the course of the project.

2.3 Tools for Payload Visualization

2.3.1 RVIZ based Visualization for ROS supported types

2.3.2 Director GUI for Offline Payload Visualization

Complementary to the RVIZ visualizer, we provide a visualization tool for offline payload data based on Director. The visualizer loads offline payload data from the folder structure described in Sec. 1.6.3. The user can interact with individual payload units and inspect various attributes of the payload such as tree instances, terrain map etc. We will expand the functionality of the visualizer over the project period as new attribute layers will be developed.

2.4 Conventions and Protocols

2.4.1 Naming of Dataset

The dataset name should contain the date of the recording, place indicator, and the robot name/id. Some examples of good naming practice are:

- 2023-01-13-wytham-anymal-coyote
- 2023-04-21-stein-am-rhein-frontier-v15

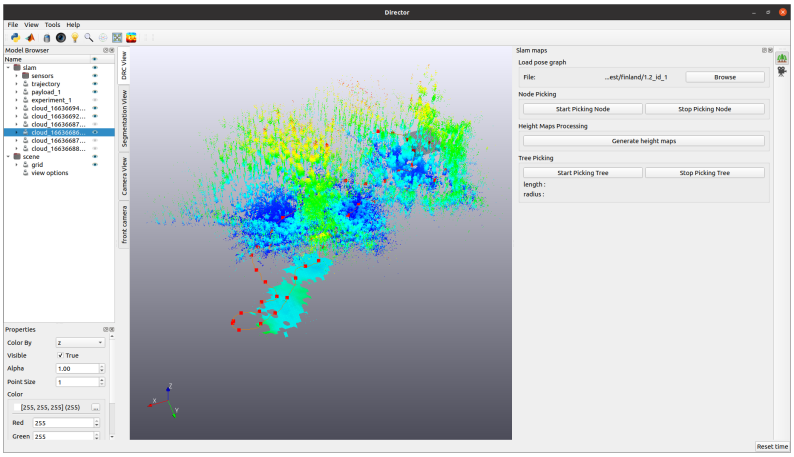

Figure 13: Offline payload visualization tool based on Director.

While saving the sensor data as rosbags, date and time for each rosbag should be retained. The recommended format is: *RobotID-Date-Time-AdditionalTag.bag*

2.4.2 Dataset Metadata

In addition to retrieval methods for individual missions described in Sec 2.2, we would like to efficiently retrieve relevant missions and payload data from the common dataset collection. As multiple datasets would be collected from various robot platforms over the course of the project, we propose to attach a metadata file to each dataset. This metadata information would be useful for retrieving relevant data from a particular test site, time frame when data was acquired, or a particular robot platform. An example metadata file is described below. Note that the metadata file may be automatically generated if the naming conventions described in Sec. 2.4.1 are followed.

metadata.yaml

- Date and time of dateset recording
- Robot ID
- GNSS coordinate (if available)
- List of sensors
- Custom tags/Additional notes from user