2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Streamlined Acquisition of Large Sensor Data for Autonomous Mobile Robots to Enable Efficient Creation and Analysis of Datasets

Mark Niemeyer[1], Julian Arkenau[1], Sebastian Pütz[1,3], Joachim Hertzberg[1,2]

***Abstract*— The increasing usage of modern AI techniques represents a transforming shift in the robotics domain. Training and accessing new models requires substantial amounts of application-specific data, but the limited resources onboard mobile robots (like processing power, network bandwidth, etc.) pose a challenge for the development of efficient data recording and provisioning pipelines. Furthermore, accessing specific information based on a combination of spatial, temporal and semantic information is generally not supported by currently available tools. In this paper, we present a methodology which allows the efficient recording of robotic sensor data streams. We show that our approach reduces the overall time needed until the data can be served via the spatio-temporal-semantic query interface of the semantic environment representation SEEREP. We further present that the maximum sensor data rate which can be stored to disk in real-time is increased for large robotic data types like images and point clouds in comparison to frequently employed solutions within the ROS ecosystem.**

I. Introduction

The ongoing transformation in robotics is driven by the increasing usage of data driven artificial intelligence (AI) and machine learning (ML) models. This requires more and more data to be captured by high resolution sensors, like cameras or LiDAR. Examples of downstream data processing steps, which cannot performed on the fly, include the generation of sub-datasets with distinct properties for training or evaluating new models. Moreover, the extraction of data from a specific region of interest over multiple points in time enables change detection. These examples motivate the necessity to retrieve specific subsets of the data based on spatial, temporal and semantic information. For the subsequent analyses of data captured by a robot in the field, it is necessary to store the data on a local storage device. This is because streaming sensor information to a network-based storage solution during a mission often presents challenges due to the limited network bandwidth available in the environments in which robots operate. Typically, the data is uploaded when a fast and reliable connection to a server becomes available, for example when arriving at a base station for charging.

When storing the sensor data to the disk, two challenges arise: (1) The disk only has a limited write speed. Thus, the data dump has to be efficient in order to maximize the number of sensor data streams which can be handled concurrently. (2) The storage format has to be compatible with the subsequent data provisioning. Otherwise, additional conversion steps need to be performed.

Fig. 1: The autonomous monitoring robot *Lero* collecting sensor data of a bed in a garden with bio-diverse cultivation

An often employed method to record sensor data within the Robot Operating System (ROS) involves using the rosbag tool [1]. The resulting rosbag file allows for efficient access to the data through the time modality, but spatial or semantic extraction is not possible.

Within this context, the paper presents an approach for storing ROS sensor data in HDF5, which is a self-describing data format suitable for high throughput (GiB/s) and partial I/O. The HDF5 file's internal layout is structured as needed by the semantic environment representation SEEREP [2]. This mitigates the need for restructuring steps and allows the direct use of the HDF5 file in the SEEREP server. Because sub-datasets can be created using spatio-temporal-semantic queries over SEEREP's gRPC API, the SEEREP compatible HDF5 file is the desired data provisioning format.

The relevance of the proposed methodology is illustrated with an agricultural robotic use case, as a crucial bottleneck for developing new algorithms and models to improve agricultural practices is the scarce availability of datasets [3]. The autonomous plant monitoring robot *Lero*

The DFKI Niedersachsen is sponsored by the Ministry of Science and Culture of Lower Saxony and the VolkswagenStiftung. This work is supported by the Federal Ministry for Economic Affairs and Climate Action within the AgriGaia project (grant number: 01MK21004A)

[1]Plan-Based Robot Control Group, DFKI Niedersachsen, German Research Center for Artificial Intelligence, Osnabrück, Germany
mark.niemeyer@dfki.de
[2]Knowledge-Based Systems Group, Osnabrück University, Osnabrück, Germany
[3]Nature Robots GmbH, www.nature-robots.com, Osnabrück, Germany

developed by *Nature Robots* is used in this paper as a real world motivation and application. *Lero* is equipped with multiple RGBD cameras and several LIDAR scanners, which can produce demanding sensor streams. The robot is used to autonomously record data in a garden with bio-diverse cultivation of vegetables with 30 to 60 different plant types (see fig. 1). The use case faces the described challenges of data recording and provisioning as part of the plant analysis pipeline. Subsets of the recorded data defined by spatio-temporal-semantic queries are needed to perform plant-specific analysis.

After an overview of the related work in section II and the presentation of the proposed methodology in section III, we demonstrate in section IV that our approach for storing ROS sensor data is more efficient than the MCAP-based rosbag tool of ROS 2 and allows for higher data rates on typical sensor message sizes of > 600 KiB. Additionally, we show that the overall time from the start of the data acquisition until the sensor data can be provisioned by SEEREP is reduced. This is achieved by avoiding conversion steps and storing the data directly in SEEREP compatible HDF5 file. Finally, this paper concludes with section V.

II. Related Work

Common tools for data recording in robotics apply a log-replay workflow leveraging a time-based index. In ROS, such data recording is implemented via the rosbag tool [1]. The tool collects the messages from a predefined set of communication streams (topics) and saves them in a *.bag* file. The file format has evolved over four major versions [4], where the latest allows for chunked data storage. Chunks are blocks of messages grouped by timestamp or message type. At the end of each rosbag, a collection of chunk information records is created to allow for a direct access pattern to each chunk. Further, the chunks can be compressed to save storage space.

The most discussed aspect of rosbags in their inherent dependency on the ROS message definition format and the ROS ecosystem. Generating the programming language specific code from the message definitions requires knowledge of various ROS specific tools and concepts, which makes it difficult to adopt. With the release of ROS 2 a pluggable rosbag API has been implemented to support its middleware independent architecture [5]. Thus, different message and transport serialization mechanisms can be used. Further, the lower level storage API supports plugins for saving data into different file formats [6]. Until the release of ROS 2 Iron, the default file format for storing rosbags in ROS 2 has been SQLite. While SQLite takes advantage of a widely used open-source library, with generalized indexing and querying, it lacks a key aspect of the original rosbag format. SQLite is not self-describing, meaning that the schemas of the stored messages are not included in the file and up-to-date message schemas must be present on the system to use the recorded data. Additionally, it lacks the capability to perform stream-oriented data appending to a file.

To address these issues, the MCAP [7] file format has been developed by Foxglove [8]. It adopts many concepts of the original rosbag with the main difference being that MCAP is serialization format agnostic. Currently, supported formats include Protocol Buffers, Flatbuffers, JSON Schema, ROS 1 and ROS 2 messages. Moreover, different serialization formats are supported in a single file. MCAP can be viewed as the next iteration of the rosbag format, as it provides vital benefits compared to SQLite. As a consequence, MCAP has replaced SQLite as the default file format for storing rosbags.

Another format optimized for recording and replaying sensor data is the Vision Replay System (VRS) from Meta Research [9]. Its main application is in the VR/AR domain, although it deals with many similar problems as in robotics. It supports chunked and multithreaded writing to a file, while also allowing for very large file sizes (terabytes of data). However, making modifications to a file requires a complete rewrite. With VRS it is recommended to use data types that incorporate temporal information or exhibit a recurring pattern. Consequently, the storage of commonly used point clouds is not advisable, making it challenging to justify its use in robotic applications.

In the field of AI and ML research, there is no universally accepted file format for sharing datasets. Instead, datasets are composed of a variety of formats, e.g. images are stored as *.png* or *.jpg* files, while corresponding labels are stored in *.json*, *.xml* or *.txt* files. To match corresponding information, custom directory structures are used. As pointed out in [10] the directories require unique processing steps for reading and merging the information in an application. In order to enhance this process, frameworks such as Datumaro [11] have been implemented to consolidate and convert various dataset formats, such as *Kitti* and *COCO*, into a unified structure. Data storage is mostly handled via object or network storage.

A popular framework for storing scientific data is the Hierarchical Data Format 5 (HDF5). It introduces a data model with an accompanying file format and a standard library for storing and managing large quantities of numerical data. As pointed out by Collette [12], HDF5 is most suitable for applications with hierarchically organized data. Additionally, arbitrary metadata composed of key-value pairs can be attached. The specification and standard library are open-source and maintained by the *HDF Group*. An overview of the format is given by Folk et al. [13]. With its support for MPI based parallel I/O and parallel file systems, it has become a popular choice within the high-performance computing (HPC) community, powering large physics experiments [14]. One drawback of HDF5 is its inefficient mechanism for deleting data. While data can be unlinked, the allocated space is not reclaimed. To recover the disk space, a new file must be created, which causes resource-intensive data copying. Further, HDF5 is defined in a complex 150-page specification and implemented in an equally complex library, causing a slowly developing standard.

Based on the advantages and disadvantages of the different data recording methods, we choose to apply HDF5 to the robotic sensor data acquisition task since it currently represents the most fitting format for recording robotic data, without focusing on a time-based replay mechanism. Moreover, HDF5 has demonstrated its efficient I/O operations for large numerical datasets, making it suitable for accommodating images and point clouds [15].

After the data acquisition, the stored data must be made available for analysis. Concerning rosbags, tools such as the rosbag database [16] offer a web interface that allows users to browse and perform post-processing on the collected data. However, for the efficient extraction of specific data points, browsing capabilities do not suffice, instead data structures with indices are needed. When dealing with robotic sensor data, the indices are spatio-temporal-semantic, akin to those detailed for GIS [17].

Different indices are available for either one or two of these modalities. For the spatial modality, examples are grid maps [18], octrees [19] and G-Arrays [19]. Time series databases, like *Prometheus* [20] and *InfluxDB* [21], address the time modality. The semantic modality requires the usage of ontologies to establish a shared vocabulary that is understandable for both humans and robots.

Instances of enhancing the spatial modality of point clouds with temporal data are illustrated in works by Krajnik et al. [22] in 2017 and Macenski et al. [23] in 2020. An approach for a better process understanding, using a spatio-semantic model, is presented by Deeken et al. [24]. GIS employ all three modalities [17], yet the databases utilized in the backend are not optimized for robotic sensor data [25]. To address these shortcomings, SEEREP has been developed to handle spatial, temporal and semantic information. Although, it does not allow replaying the sensor messages like *rosbag*, it enables the efficient access to the data via spatial, temporal and semantic indices. It has a tight integration to ROS as it adopts the sensor message definitions and uses the TF library for coordinate frame transformations. The architecture follows a client-server model, featuring a gRPC API as its primary data interface, as detailed in previous research [2].

Building on these arguments, the SEEREP server is chosen as the data provisioning tool within this work. Thus, the sensor data has to be made available in a SEEREP compatible HDF5 file after the data acquisition. The contribution of this paper is therefore the optimization of the sensor stream storage to a SEEREP compatible HDF5 file.

III. Methodology

As already motivated in the introduction, the work presented in this paper has two objectives. The first objective is to store the data as fast as possible to the disk in order to maximize the data rate from the sensors, which can be handled in real-time. Handling the data streams in real time is a hard constraint. If the constraint is not satisfied, data waiting to be written will accumulate in RAM until messages have to be dropped. The second objective targets the provisioning of the captured data.

As outlined in the related work section, the semantic environment representation SEEREP enables the efficient creation of sub-datasets based on the position and spatial dimensions, timestamp, and semantic annotations of the sensor data. As soon as the data is transferred to a SEEREP server, the server can load the data and create the indices needed for the sub-dataset extraction.

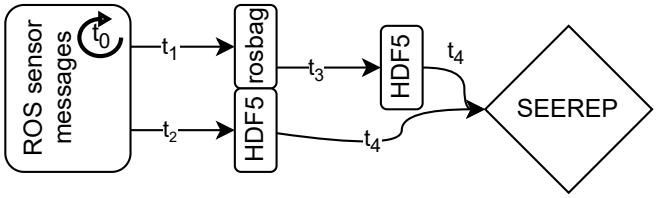


Fig. 2: Visualization of the time needed for the storage to rosbag with a subsequent conversion to HDF5 in comparison to the direct storage to HDF5.

Currently, the data is stored through commonly used tools (in most cases: rosbag) during the acquisition mission. Because the data is stored in real-time and the sensor data cannot be stored faster than it is created by the sensors, the storage requires a time period t_1 which corresponds to the mission time t_0. After the storage, the sensor data has to be converted into a SEEREP compatible HDF5 file. This requires some additional time t_3. Afterward, the HDF5 file has to be transferred to a SEEREP server, where it has to be loaded and processed, which needs some additional time t_4.

To optimize the overall required time, we directly store the data in a SEEREP compatible HDF5 file. Again, because the data cannot be stored faster than it is created, the overall time t_2 for the direct storage of the sensor data is also nearly the same as t_0 ($t_0 \approx t_1 \approx t_2$). Thus, the duration t_3 for the conversion is avoided. The time period t_4 is the same for both approaches and is therefore not further discussed. An overview of the relationships between the storage durations is visualized in figure 2.

To address the optimization of the data rate (GiB/s) which can be handled in real-time, all unnecessary operations (like the creation of indices) are avoided while dumping the sensor data in HDF5. Though, to enable all queries of SEEREP, indices for the spatial, temporal and semantic information have to be created. Thus, the creation of the indices is performed when the server loads the HDF5 file after the data acquisition mission.

During data storage, a universally unique identifier (UUID) is created for each ROS message and within the corresponding data type's HDF5 group an HDF5 subgroup is created for each message. The UUID serves as a unique identifier for the stored data and is used by the indices as a reference to the data within the HDF5 file, thus enabling random access. In the newly created group, the main sensor data (the image or the point cloud itself) is stored as a contiguous 1D byte array within an HDF5 dataset. The metadata of the sensor messages (like the header) are stored alongside the HDF5 dataset as HDF5 attributes.

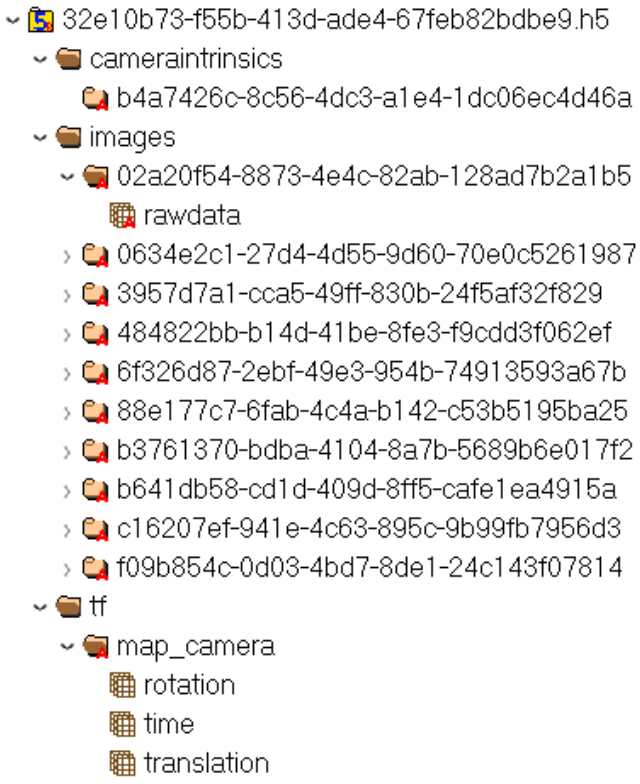


Fig. 3: Structure of the HDF5 file for images with the corresponding camera intrinsic and coordinate frame transformation (tf).

The structure of an exemplary HDF5 file containing data from an image acquisition mission is presented in figure 3. At the topmost level lies the root of the HDF5 file, with the file name also serving as a UUID. In this example, three HDF5 groups are needed to encompass all the required information for the image datatype. In the first group */cameraintrinsics*, the camera intrinsic of the camera(s) capturing the images are stored. The intrinsics are commonly published on a ROS topic with the message type *sensor_msgs/CameraInfo*. The UUID of the stored camera intrinsic is also used for referencing it in the related image messages.

The second group */images* is used to store the captured image messages. In the exemplary HDF5 file of figure 3 ten of these image groups are shown. Alongside the dataset */images/[UUID]/rawdata* of the image itself, the timestamp (from the header of the *sensor_msgs/Image*) for the temporal index and the coordinate frame of the sensor message (also from the header of the *sensor_msgs/Image*) are stored as HDF5 attributes.

To enable the creation of a spatial index in a common coordinate frame, the transformation (tf) from the sensor frames to the common frame has to be known. For this, the transformations (timestamp + translation + rotation) published in ROS are stored in the third group */tf*. In figure 3 only one group is used to represent all transformation updates from the map frame to the camera frame. For point clouds, a single group */pointclouds* is required to store all relevant information from the ROS message. Based on this structure, the experiments regarding the direct storage of ROS sensor data into HDF5 are conducted in the next section.

IV. Experiments

We conduct two experiments, the first establishes a baseline for the time savings by removing conversion steps in the data pipeline. This is done by measuring and comparing the times required for the individual steps (see figure 2) in the pipeline. The second experiment compares the maximum throughput between MCAP and our HDF5-based storage solution. MCAP was chosen since it reflects the current state of the art for multimodal data logging inside the ROS ecosystem. Both experiments were performed on a Lenovo P15 equipped with an Intel Core i7-11800H processor with 8 physical cores and 32 GB of RAM using Ubuntu 20.04.5 LTS (kernel version 5.15.0-79-generic). The data is written to an Intel 670p NVME SDD, which is rated at a maximum 3500 MB/s read and 2500 MB/s write. Our objective with using an SSD compared to a RAM-DISK is to assess the practical, real world performance including all intermediate caching steps.

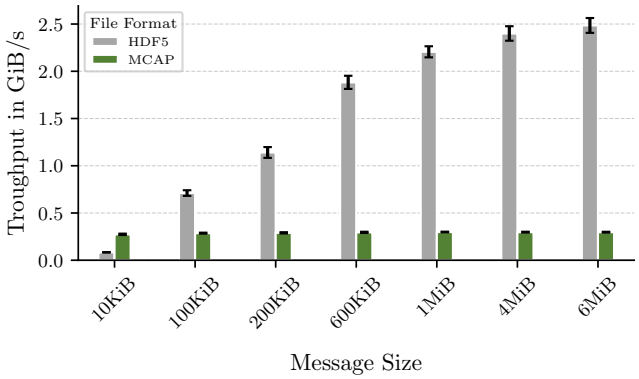
A. Data Storage Experiment

We utilize a rosbag recording (ROS 1) of beds in a garden sourced from *Lero* (see figure 1). The hardware setup of *Lero* includes three RealSense D435i RGB-D cameras incorporating NIR filters, allowing the capture of RGB images, near-infrared images, and depth maps of the plants. For this experiment, we only rely on the RGB images, since they are the most computational heavy datatype. The RGB images have a resolution of 480×640 pixels, resulting in images with a size of around 900 KiB which are captured at a rate of 15 Hz The rosbag created by Lero is about 4 min long, contains 10986 RGB images from the three cameras, and has an overall size of about 9.5 GiB.

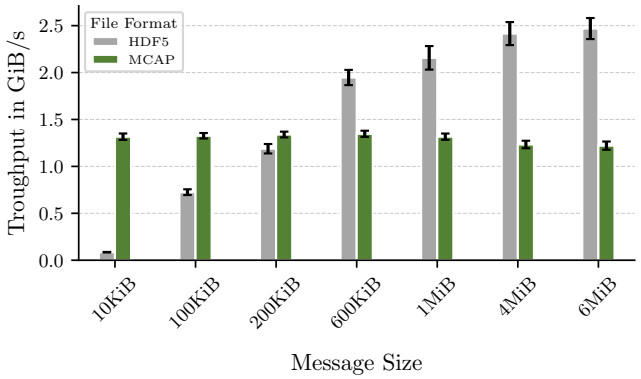
In this experiment, we measure the time needed for the conversion from a rosbag to HDF5 (see t_3 in figure 2). The mission duration t_0 and the time needed to store the data to a rosbag t_1 or to HDF5 t_2 have to be nearly the same, as pointed out earlier. For the given camera setup, this means that 45 RGB images (3 cameras @ 15 Hz) have to be stored every second, or one image every 22.2 ms on average.

For the measurement of t_3 the same methodology and the same software is used as presented in the previous section, but instead of directly dumping the sensor message, we iterate over the messages stored in the rosbag. The time needed for the conversion of all 10986 RGB images was measured 50 times, with a resulting mean of 26.43 s with a standard deviation of 0.54 s. As the overall rosbag is 4 min long, the additional 26.43 s on average for the conversions are an increase over the data acquisition time of about 11 %.

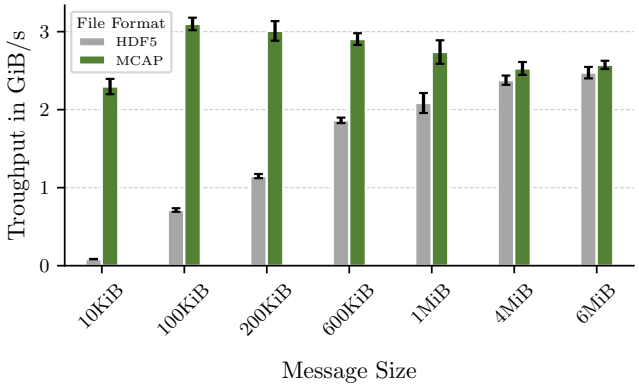
These results demonstrate that in cases where subsequent processing is required (such as importing to SEEREP), storing sensor images directly to HDF5 is a more efficient and therefore preferable approach.



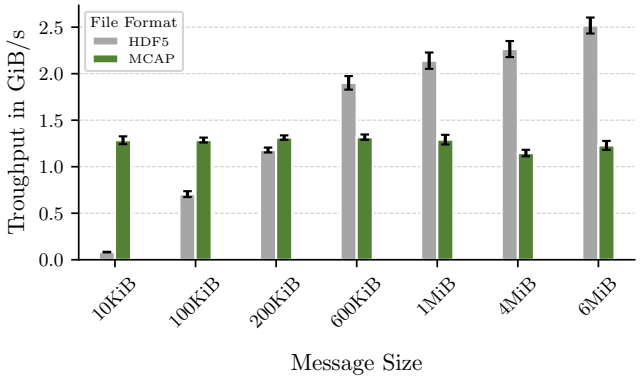
(a) Settings for MCAP include CRC calculations, chunking (768 KiB) and Zstd compression, while HDF5 uses no chunking, no compression and no CRC calculations.



(b) Settings for MCAP include CRC calculations, chunking (768 KiB) and no compression, while HDF5 uses no chunking, no compression and no CRC calculations.



(c) Settings for MCAP include no CRC calculations, no chunking and no compression, while HDF5 uses no chunking, no compression and no CRC calculations.



(d) Settings for MCAP include CRC calculations, chunking (1 MiB) and no compression, while HDF5 ueses no chunking no compression and no CRC calculations.

Fig. 4: Benchmark results for the data throughput in GiB/s between MCAP and HDF5. Messages sizes from 10 KiB up to 6 MiB are used while writing 250 MiB of total data. The error bars represent one standard deviation, calculated from $N = 50$ runs.

B. Data Throughput Experiment

To assess our proposed method for storing robotic sensor data in HDF5, and to compare it with current state-of-the-art solutions, we perform a benchmark on the write throughput. The maximum throughput, considering a representative setting, is a key characteristic as it limits the number of concurrent sensor streams that can be handled. If a recorder can't keep up with incoming streams, either incoming messages are dropped or message queues grow very quickly, resulting in rapidly expanding memory requirements. To easily reproduce the results, we provide the required code in the SEEREP repository*.

For the benchmark, ROS messages with sizes from 10 KiB up to 6 MiB are written as fast as possible into the respective file formats until a maximum of 250 MiB of total data is reached. When the total size is not a multiple of the message size, the last message is limited to the remaining size. For the message payload, sequential slices are sampled from a rosbag recording of *Lero*, representing typical robotics data. For example, for 10 KiB messages, the first message would contain the range $[0\ \text{KiB}, 10\ \text{KiB})$ while the second message would contain $[10\ \text{KiB}, 20\ \text{KiB})$. If more messages than the total bag size are requested, the sampling will wrap around to the beginning. The message sizes are chosen to represent small messages like coordinate transformations, as well as large uncompressed images with RGB and depth channels. The 480×640 pixel depth images captured by *Lero* have a message size of around 600 KiB while corresponding RGB images have around 900 KiB. Full HD (1920×1080 pixel) images require around 4 MiB for depth images and around 6 MiB for RGB images. For both formats, the required time for each write call is measured and then averaged over 50 runs. Setup tasks like serialization, opening and closing of the files are not included. In MCAP the messages are stored using the C++ interface provided by the open source implementation [7]. As MCAP relies on an external serialization mechanism, such as Protobuf, Flatbuffers, ROS1, or ROS2, the benchmark utilizes the ROS 1 serialization

*https://github.com/agri-gaia/seerep

API. For HDF5 *HighFive*, a modern header only C++ library for *libhdf5* is used. The message payload is stored in a contiguous 1D dataset, and other fields are saved as attributes to the dataset. Using a contiguous dataset layout in HDF5 is reasonable, as the datasets are entirely read and written during dataset reception or transmission. However, this precludes the utilization of filters such as compression or CRC calculations, as they exclusively operate on chunks. The results of different benchmark configurations are shown in figure 4, the throughput is measured in Gibibytes per second (GiB/s). Error bars represent one standard deviation, calculated from 50 runs of each message size. In figure 4a the results from the default configuration of MCAP and HDF5 are shown. For HDF5 this means no chunking, no compression and no cyclic redundancy checks (CRC). In MCAP CRC calculations, chunking (768 KiB) and *Zstd* compression [26] are enabled by default, resulting in a smaller and safer file. In figure 4b MCAP is used without compression, resulting in a throughput increase from around 250 MiB/s to around 1.3 GiB/s. Further, a strong benefit of using MCAP for small messages ≤ 100 KiB can be noticed. At around 200 KiB, MCAP and HDF5 exhibit a similar throughput. Beyond this point the throughput in HDF5 increases with larger messages capping at around 2.5 GiB/s due to the hardware limitations of the used SSD. MCAP surpasses HDF5 at every message size when using non-chunking storage (see figure 4c). However, in this configuration no info records are written, requiring the file to be re-indexed before it can be read. The last configuration in figure 4d uses a chunking size of 1 MiB. No benefit can be recognized compared to the results in figure 4b, where the default chunking size of 768 KiB is used.

The results obtained demonstrate that ROS messages can be stored in HDF5 with comparable or superior data rates compared to MCAP for larger message sizes. While the non-chunking configuration of MCAP offers a higher data rates, it requires re-indexing before reading data.

V. Conclusion

This paper presents a methodology for streamlined data acquisition on autonomous mobile robots which allows for sensor data storage into HDF5. Based on the created HDF5 file, SEEREP can provide specific subsets of the data defined by spatio-temporal-semantic queries.

Besides the avoidance of additional conversions steps, the proposed methodology also enables fast storage of high-resolution sensor streams like point clouds and images. Our results also show real-world applicability, since representative sensor data from the plant monitoring robot *Lero* is used. The experimental comparison with MCAP, the capturing library used by rosbags in ROS2, shows a direct performance increase for sensor messages of 600 KiB and above.

Overall, the presented methodology enables a more efficient data acquisition approach onboard robots, suitable for subsequent provisioning of specific subsets of the sensor data via, e.g., SEEREP. These subsets are suitable for training and assessing application specific AI models, or they can serve as input for analysis algorithms.

Currently, the presented methodology is slower than MCAP for smaller message sizes. However, using a buffering strategy to aggregate multiple small messages and writing them in a single operation is likely to enhance the storage speed.

The software is still in active development, and we continue to improve its usability and integration into SEEREP. Furthermore, the incorporation into large data recording campaigns (multiple terabytes of data) in the agricultural domain and accompanying ML pipelines is planned.

References

[1] "rosbag - ROS Wiki." [Online]. Available: http://wiki.ros.org/rosbag

[2] M. Niemeyer, S. Pütz, and J. Hertzberg, "A spatio-temporal-semantic environment representation for autonomous mobile robots equipped with various sensor systems," in *2022 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, 2022, pp. 1–6. [Online]. Available: https://doi.org/10.1109/MFI55806.2022.9913873

[3] Y. Lu and S. Young, "A survey of public datasets for computer vision tasks in precision agriculture," *Computers and Electronics in Agriculture*, vol. 178, p. 105760, 2020. [Online]. Available: https://doi.org/10.1016/j.compag.2020.105760

[4] "rosbag format specification." [Online]. Available: http://wiki.ros.org/Bags/Format

[5] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: https://doi.org/10.1126/scirobotics.abm6074

[6] "Implementation of rosbags in ros 2." [Online]. Available: https://github.com/ros2/design/blob/ros2bags/articles/rosbags.md#implementation-of-rosbags-in-ros-2

[7] "Mcap." [Online]. Available: https://mcap.dev/

[8] "foxglove." [Online]. Available: https://foxglove.dev/

[9] "Vision replay system (vrs)." [Online]. Available: https://github.com/facebookresearch/vrs

[10] N. Iqbal, M. Niemeyer, J. C. Krause, and J. Hertzberg, "Scalable evaluation pipeline of cnn-based perception for robotic sensor data under different environment conditions," in *2023 IEEE European Conference on Mobile Robots (ECMR-2023)*. Coimbra, Portugal: IEEE, 9 2023.

[11] "Datumaro." [Online]. Available: https://github.com/openvinotoolkit/datumaro

[12] A. Collette, *Python and HDF5*, 1st ed. Beijiing: O'Reilly, 2014.

[13] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson, "An overview of the hdf5 technology suite and its applications," in *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, ser. AD '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 36–47. [Online]. Available: https://doi.org/10.1145/1966895.1966900

[14] A. Bashyal, P. V. Gemmeren, S. Sehrish, K. Knoepfel, S. Byna, and Q. Kang, "Data storage for hep experiments in the era of high-performance computing," 2022.

[15] T. Wiemann, F. Igelbrink, S. Pütz, and J. Hertzberg, "A file structure and reference data set for high resolution hyperspectral 3d point clouds," *IFAC-PapersOnLine*, vol. 52, no. 8, pp. 403–408, 2019, 10th IFAC Symposium on Intelligent Autonomous Vehicles IAV 2019. [Online]. Available: https://doi.org/10.1016/j.ifacol.2019.08.101

[16] "Bag Database." [Online]. Available: https://github.com/swri-robotics/bag-database

[17] M. Yuan, "Use of a three-domain repesentation to enhance gis support for complex spatiotemporal queries," *Transactions in GIS*, vol. 3, no. 2, pp. 137–159, 1999. [Online]. Available: https://doi.org/10.1111/1467-9671.00012

[18] P. Fankhauser and M. Hutter, "A universal grid map library: Implementation and use case for rough terrain navigation," in *Robot Operating System (ROS): The Complete Reference (Volume 1)*, A. Koubaa, Ed. Cham: Springer International Publishing,

2016, pp. 99–120. [Online]. Available: https://doi.org/10.1007/978-3-319-26054-9_5

[19] H. Roodaki, M. Dehyadegari, and M. N. Bojnordi, "G-arrays: Geometric arrays for efficient point cloud processing," in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 1925–1929. [Online]. Available: https://doi.org/10.1109/ICASSP39728.2021.9413902

[20] Prometheus, "Prometheus - Monitoring system & time series database." [Online]. Available: https://prometheus.io/

[21] "InfluxDB: Open Source Time Series Database." [Online]. Available: https://www.influxdata.com/

[22] T. Krajník, J. P. Fentanes, J. M. Santos, and T. Duckett, "Fremen: Frequency map enhancement for long-term mobile robot autonomy in changing environments," *IEEE Transactions on Robotics*, vol. 33, no. 4, pp. 964–977, 2017. [Online]. Available: https://doi.org/10.1109/TRO.2017.2665664

[23] S. Macenski, D. Tsai, and M. Feinberg, "Spatio-temporal voxel layer: A view on robot perception for the dynamic world," *International Journal of Advanced Robotic Systems*, vol. 17, no. 2, p. 1729881420910530, 2020. [Online]. Available: https://doi.org/10.1177/1729881420910530

[24] H. Deeken, T. Wiemann, and J. Hertzberg, "A Spatio-Semantic Model for Agricultural Environments and Machines," in *Recent Trends and Future Technology in Applied Intelligence*, M. Mouhoub, S. Sadaoui, O. Ait Mohamed, and M. Ali, Eds. Cham: Springer International Publishing, 2018, vol. 10868, pp. 589–600. [Online]. Available: https://doi.org/10.1007/978-3-319-92058-0_57

[25] H. Denken, T. Wiemann, and J. Hertzberg, "Grounding semantic maps in spatial databases," *Robotics and Autonomous Systems*, vol. 105, pp. 146–165, July 2018. [Online]. Available: https://doi.org/10.1016/j.robot.2018.03.011

[26] "Zstandard - fast real-time compression algorithm." [Online]. Available: https://github.com/facebook/zstd