Evaluation of robotics data recording file formats

Oct 6, 2021
John Hurliman
Foxglove Technologies Inc

The transition from ROS 1 to ROS 2 involves switching recording file formats from the classic “rosbag” (.bag) file format to a new serialization format. The rosbag2 recording software includes a pluggable interface to support different serialization methods, with the current default using SQLite (.db3) files. Non-ROS robotics frameworks each use their own recording format, leading to further fragmentation. This document reviews desirable properties of a robotics data recording format (including those considered during ROS 2 development) and evaluates existing and proposed solutions against those requirements.

Problem Statement

The goal of recording robotics data is to store and analyze or replay the data later to understand what environmental inputs were observed, what actions were taken, and some granularity of the internal state to understand what decisions were made and why. This goal is usually accomplished by recording multiple streams of data as packetized messages with high-resolution timestamps. Example data streams include camera images, LIDAR scans, pose estimates, planning decisions, and actuator commands. The streams of data vary significantly in size, frequency, compression, and schema complexity. The following desirable properties have been collected from Karsten Knese’s rosbag2 design proposal and from interviewing robotics companies using a variety of approaches to data recording and data storage.

Serialization Agnostic

The robotics world has not standardized on a single message serialization format. ROS1 included its own serialization format, while ROS2 uses CDR as the default but includes a pluggable middleware layer to support other serializations. Non-ROS robots may be using Protocol Buffers, CBOR, JSON, or a proprietary format. A generic recording container must include support for different serialization formats to support the breadth of choices for message serialization in use today.

Determinism

Playing back a recording should be a deterministic process. The ordering of messages in a single stream and across streams must be well defined to enable this. This ordering is generally achieved by associating high-resolution timestamps with each message and iterating across all streams concurrently in timestamp order during playback. There are many different concepts of time in a recording: the time a message was originally published, the time when it was sent by

the publisher (this can be very different for messages that are held in a queue and replayed), when the data recording process received it, or when the recorder wrote it to disk. Using a consistent definition of time is the most critical requirement while storing multiple timestamps can enable additional use cases.

Self-Describing

The design of robotic systems is often evolving. Data streams are added and removed, message publishing frequencies are tuned up and down, and schemas change. For a recording file to be useful as long-term storage, it must be self-describing. That is, the decoding of the container itself must be understandable by looking at a file signature, and the serialization of individual messages must either be self-describing (such as JSON or CBOR) or described by a schema embedded in the recording for each stream. Parsing a recording should not require access to external schemas, a specific codebase revision, or any other external resources.

Big and Small Data

Some sensor streams can produce gigabytes of data per second. While splitting files along the time axis is always possible, the recording container should support at least hundreds of gigabytes before forcing a file split. The per-message overhead should be low enough to support small messages at 1kHz+ frequency, and large individual messages should be possible to support uncompressed video feeds.

Write-Optimized

For a file format to be suitable for recording, it must be optimized for write throughput. An append-only file format is desirable for performance, to enable streaming data as it is written, and for crash safety. It should be possible to make I/O and CPU/memory tradeoffs by either compressing data at write time or writing uncompressed data. Things to avoid: expensive index creation, expensive working memory requirements, and expensive transcoding at write time.

No A Priori Channel Knowledge

Most robotics frameworks use a publish-subscribe (pub/sub) messaging system to send messages over channels, also known as topics. It is not always possible to know the complete list of channels that should be recorded ahead of time or the schemas for each channel before the channel starts publishing messages. For a recorder to support append-only, it must be possible to write new channel information and schemas midway through writing, after writing has already begun for other channels.

Corruption Resilient

Robots operate in chaotic environments. The most critical recording files are often associated with an unpredictable event such as a collision or system failure where data integrity cannot be

guaranteed. Some possible mitigations are splitting the data into smaller chunks, adding chunk-level checksumming, and a documented process on how to recover partial data from a truncated or otherwise corrupted recording.

Read-Optimized

There are many kinds of read-optimization, and often there is a direct tradeoff between read-optimization and write optimization. A recording format should be optimized for write throughput first and foremost. However, the generated files should be readable without loading the entire file into memory, parseable in different environments (desktop, server, web), and support efficient random access and range requests if possible. Although it is possible to use different file formats for recording and archiving with a transcoding process, if a file format can support high write throughput and speed up these read operations, it simplifies the development ecosystem. Tooling can then converge around a single file format.

Standards Compatible

If a universal robotics recording file format is adopted, it should be eligible to become a standard. The standard could take the form of an Internet Engineering Task Force RFC or a normalized specification through another standards body. Typical requirements include real-world usage in multiple places, complete documentation, no patent encumberment, and at least two independent implementations.

Existing Robotics Recording Formats

We can compare this set of requirements and desirable properties against existing file formats designed as robot recording containers.

rosbag1

The original ROS1 .bag file format meets several of our requirements. It is deterministic, self-describing, supports big and small data, somewhat write-optimized (although not append-only), does not require a priori channel knowledge, somewhat corruption resistant (no checksums, but data can be recovered at the chunk level and the index can be rebuilt), read-optimized via an index, and standards compatible. Its major shortcoming is that it is not serialization agnostic.

rosbag2

Whereas rosbag1 refers to both a file format definition and two reference implementations (C++ and Python), rosbag2 is a C API with a plugin interface for implementations, plus a default implementation using SQLite as the on-disk format. Since this document is focused on serialization formats, rosbag2 will be defined as the default SQLite implementation. This format

is serialization agnostic, deterministic, but not (yet) self-describing, although there is ongoing work to address that. It does not require a priori channel knowledge, is corruption resistant, and is read-optimized except in the web use case. The significant shortcomings are around write-optimization (not append-only, the index is updated per row insert), lack of compression support at write time beyond the individual message level, and requiring the entire file to be read into memory in the web use case (via sql.js). SQLite also poses challenges to standardization. The Internet Engineering Task Force, W3C, and other standards bodies require any specification to have two independent implementations to graduate to a standard. SQLite has only ever had a single implementation that is cross-compiled to different platforms, which caused the Web SQL Database proposal to reach an impasse.

Length-delimited Protobufs

From interviewing various robotics companies, the most popular serialization format outside of ROS is to write protobuf messages to disk with a simple length delimiter or short message header. If timestamps are included in every message, this can be deterministic, but the initial .proto schemas must be bundled as well to meet the self-describing criteria. This approach does not require a priori channel knowledge and supports big and small data. It does not provide any explicit corruption resistance, although a truncated final message can sometimes be detected by a serialization failure. It is a write-optimized append-only approach, and compression can be supported by piping the write output through a compressor. Protobufs have been submitted as an Internet-Draft (I-D) standard. The significant shortcomings are that by nature of defining around a single serialization format, it is not serialization agnostic, and that without a well-defined standard, the files produced by different implementations will be incompatible. The lack of read-optimization excludes its use in playback or analysis tools that require random seeking.

Apollo Cyber RT

Cyber RT includes its own recording file format. However, the documentation of the format itself (opposed to the software APIs surrounding it) is scarce. It is likely similar to Length-delimited Protobufs for the purpose of this discussion.

Existing Big Data Formats

Outside of robotic recording file formats, we can evaluate common “big data” containers for suitability to this problem.

Avro - Native Types

Avro is a row-based storage format that requires each row to follow a single schema. This requires a union type of all possible message schemas that need to be rewritten as new channels are added into a container or each channel to be written to a separate file. Support for

different serialization formats either requires mapping each serialization format to Avro data types, which may be a lossy process.

Avro - Opaque Bytes

Another way of using the Avro format is to define the schema as a single field containing an opaque array of bytes, which supports different serialization formats in a common container. This approach meets many of the desired requirements, but it defeats the most commonly desired benefit of using an existing container format: the ability to ingest data into a third-party system and have that system understand the data in a typed way. A recording format built on Avro would still require standardizing how and where to store content types and per-channel information such as schemas. Regardless, Avro containers storing opaque byte messages may be one possible solution.

HDF5

HDF5 predates Avro and suffers from similar limitations while also missing support for deeply nested data via schemas. Using HDF5 as strictly a wrapper around opaque byte messages would require a more complex file format for no additional gain.

Parquet

Parquet is a columnar data store that is not suitable for random access to messages and is not optimized for replay. While Parquet can play a valuable role in robotics for analysis and long-term storage, its lack of suitability for real-time recording and message playback excludes it from our search for a recording file format.

Next Steps

By comparing existing robotics recording formats with generic big data container formats, we can see there are practical advantages to designing a recording format specific to the robotics use case. However, none of the existing recording formats meet all the requirements and desirable traits laid out in this document. Many of the proposed approaches (rosbag2 SQLite, Avro with opaque buffer) follow an anti-pattern of using an existing schema-based format but only storing opaque binary blobs, negating the promise of interoperability from using an existing format. In light of this evaluation, I would like to propose a file format tailored to the requirements of robotics recording. I will introduce this file format in a follow-up proposal. The goal is to continue collecting requirements and desirable traits and collaboratively craft an industry standard backed by multiple implementations.