Badlog: Simple and Modular Data Logging for Robotics

Dominik Winecki, Team 1014 Bad Robot

Abstract

Robots do weird things; especially when they are built in six weeks. To perform reliably finding the source of these problems is essential. Data logging can be the best tool in any software teams arsenal to identify the source of a problem.

Here we present badlog: a small collection of tools to provide useful and capable data logging while posing minimal risk to FRC teams.

The badlog Framework

The badlog framework is our solution to data logging. In short, it allows us to create interactive HTML displays with our data. It has helped our teams during competitions and has proven to be invaluable for identifying flaws and analyzing failures. All of our logs are hosted at `https://badrobots1014.github.io/logs/`; they serve as an example of what this system can produce.

The framework consists of of two parts: the run-time (badlog) and the visualization tools (badlogvis). Both of these are connected with a common format for logged data (bag files). This architecture takes inspiration from the rostopic/rosbag model used in Robot Operating System (ROS). Because of the split, many separate use cases can be implemented.

Objectives

FRC has special requirements for logging that differ from what is seen in many other fields. These are the objectives targeted for this system:

1. **Speed and Reliability** The absolute worst case scenario for any logging system is if it interferes with the performance or function of the robot. For simplicity, there should be no unneeded computation or writes, minimal configuration, and no networking.
2. **Simplicity** A FRC build happens in six weeks, and at competitions development speed often takes precedence over quality. A simple design allows minimal interference and therefore keeps itself running.
3. **Quick Visualization** Getting the data is only half the challenge and unnecessary if it goes unused. An ideal solution would allow any team member, regardless of experience, to find out what went wrong.

badlog logging tool

Badlog is a Java library for data logging. It can log two types of data: values and topics. Values are any key-value string pair that is known at initialization. Topics are a constant stream of numeric data.

Examples of values include match number, PID constants, or detected controllers. Examples of topics include motor outputs, current draw, and gyroscope data. Each topic has a name, unit, method to retrieve data, and attributes.

The source code for badlog is open and available at `https://github.com/dominikWin/badlog`

For comments, suggestions, or help with implementing this framework for your team, please feel free to contact me directly, or reach out to any members of Team 1014.

badlogvis visualization tool

badlogvis is a visualization tool that takes bag files (or CSV files) and transforms them into readable HTML files. It also uses different attributes to perform calculations after the fact and adds these derived graphs. It is open source and available at `https://github.com/dominikWin/badlogvis`

Example usage

This is a simple walk-through of a basic application of badlog. This example shows the entire API, and a real-world implementation is at `https://github.com/BadRobots1014/BadRobot2018`.

```
BadLog log;

public void robotInit() {
        log = BadLog.init("test.bag");

        BadLog.createValue("Match Number",
                "" + DriverStation.getInstance().getMatchNumber());

        BadLog.createTopic("Match Time", "s",
                        () -> DriverStation.getInstance().getMatchTime());

        BadLog.createTopicSubscriber("Random Numbers",
                BadLog.UNITLESS, DataInferMode.DEFAULT, "integrate");

        // Subsystems can add their own topics and values
        drivetrain = new Drivetrain();

        log.finalize();
}

public void periodic() {
        BadLog.publish("Random Numbers", Math.random());

        log.updateTopics();
        log.log();
}
```

This example shows the three basic types of logging that can be done.

- A value is created for the Match Number.
- A topic for Match Time is created with a lambda function to get the data at a later time.
- A subscribed topic is created for Random Numbers.

Every topic has a unit associated with it. This is only so charts can be annotated; it does not change any calculation. The `BadLog.UNITLESS` constant ("ul") is provided if needed.

Subscribed topics use a Publisher-Subscriber pattern and require instruction on what to do when no data is provided. `DataInferMode.DEFAULT` uses a default value of -1 when not set. `DataInferMode.LAST` uses the last set value. The infer last mode allows for potentially expensive operations to be done less often than the general log update period. An example application of inter last mode is for the process RAM usage where the user runs it once per second.

Attributes

In the example, the "Random Numbers" topic has a single attribute of "integrate". Any number of attributes can be provided for a topic. What is done with the attributes depends on the application, the logging runtime doesn't use their values directly. Here the "integrate" attribute is provided, which badlogvis recognizes as an order to append an additional graph with the integral of this topic.

Attributes allow for post-processors to know properties of a topic. They are a modular way to extend this format. Some examples of ways they can be used are mathematical operations (like badlogvis provides), hooks (like piping the data through Matlab/Python to create something else), or setup parameters.

For example, in badlogvis, the most useful attribute has been "xaxis". When put on a topic it becomes the x-axis for all other graphs, and it uses the unit provided from time to have accurately derived units for other topics. For example, if the topic for Power Draw (J) has the "integrate" topic badlogvis will create a new graph with $J \cdot s$ as the unit.

Disk Space

Logged data can take up a lot of disk space. On standard FRC hardware the storage is filled up after a few hours of constantly logging with around 50 topics. There are a few ways that the used storage space can be decreased without losing data quality.

First, badlog uses a function for converting doubles to strings for logging (which can be set with `setDoubleToStringFunction`). The length of the stored data is proportional to file size. By default badlog uses printf format "%.5g".

If a session has a defined x-axis for the then another option is to reduce logging frequency when not activated. For example, if a robot is disabled, rather than logging each periodic update, the log may only update once every 250 ms. This can be achieved by only calling the `updateTopics` and `log` methods when needed. Old subscribed data are overwritten and all calculations using time continue to work.

Another technique that works well is lazy initialization. By delaying the initialization procedure until a Driver Station is connected (or, as a backup, entering non-disabled state) fewer log files are created and more information is available to be logged. Many variables are unavailable until a Driver Station is connected such as field orientation, controllers, and accurate system time.

Bag File Specification

Bag files are the standard way to represent log data. Their design had four goals:

1. **No lock in** Sometimes projects fail. In the worst case bag files can be converted to CSV.
2. **Power-off tolerant** The standard way to turn off an FRC robot is to kill its power supply. Files need to survive unexpected shutdown.
3. **Simple** There is no need to recreate the wheel. Existing standards and technologies exist, and they should be used.
4. **One File** There should never be more than one file per match. Once there are two files there are synchronization issues, so stick to one.

CSV is the only major data format that can take a sudden cut-off so it is used for the topic data. Values and topic meta-data are stored in a one line JSON header. This solution allows for a bag file to be converted to a CSV file by deleting the first line while still allowing for extensibility and a Key-Value store.

The bag file from the previous example would produce the following bag file (JSON expanded for readability, would be one line):

```
{
    "values": [
        {
            "name": "Match Number",
            "value": "8"
        }
    ],
    "topics": [
        {
            "name": "Match Time",
            "unit": "s",
```

```
            "attrs": []
        },
        {
            "name": "Random Numbers",
            "unit": "ul",
            "attrs": ["integrate"]
        }
    ]
}
```

This solution solves each of the requirements. One limitation is that all of the values need to be known at initialization so the header can be created. A fix would be to add a column with JSON data, but this would add overhead so there is no direct support since user implementation is trivial.

Badlogvis

Badlogvis is a program that takes bag files (or CSV) and generates HTML. It serves as a useful tool to instantly view data and a reference implementation for a consumer of bag files. See `https://badrobots1014.github.io/logs/` for files created with badlogvis.

Badlogvis was created with many of the same design principals of the previous standards. It is open source and available at `https://github.com/dominikWin/badlogvis`.

Badlogvis Features

- **One File** Each output is a single HTML file. There are no external links so it can be viewed offline. Because of this it can be used locally, shared (Slack works well), and hosted online. The badlogvis program itself is also a one-file executable, with all dependencies embedded within the binary.
- **Interactive** Selecting a time frame on one graph syncs to all graphs, and lines in graphs can be hidden. Each graph can export an image of itself.
- **Fast** Outputs handle well with many hundreds of thousands of data points, even on smartphones, due to WebGL accelerated charts.
- **Attributes** Many different attributes are used to control outputs. Most common calculations can be done after the fact, and multiple topics can be joined to a single graph. All generated data (called virtual topics) annotated with square brackets.
- **No data loss** Each HTML file has a full copy of the CSV portion of the bag file that can be downloaded. There is no reason to keep the bag files after they have been converted to HTML.

Future Expansion

There are many additional utilities that can be created with this architecture. None of these have been fully implemented but could be done so without much trouble.

The rosbag system that badlog was based on was designed to replay values live into different systems, and thus this is a supported feature of badlog. Live charts can be added to match videos to watch the data live; however, this system is not fully implemented.

Another use case (that thankfully wasn't needed) is automated data analysis. Bag files could be fed into a Matlab/R/Python/Julia script and automatically gather statistics or search for known bad behavior. Badlogvis originally was intended to support calling hooks like this and embedding their results, but this was never needed.

Team 1014 uses Java, but porting badlog to C++ would not be difficult. If a port is created then those robots would have access to badlogvis and any other utilities.