UW EE 499 (2 credit) Research Report
Spring 2021

Data Analysis & Visualization of Rosbags

Author: Paul Chung
Date: June 10th, 2021

UNIVERSITY of WASHINGTON

I. Table of Contents:

II. Table of Figures:

1.0 - Competition Background Information:

The University of Washington's EcoCAR, a research organization composed of undergraduate and graduate students, is a part of a four-year EcoCAR Mobility Challenge against 12 other universities. Teams apply propulsion systems, connected and automated vehicle technology, energy efficiency, and consumer appeal to a 2019 Chevrolet Blazer. This event is sponsored by General Motors, Mathworks, DOE, and Argonne National Laboratory and runs from 2018- 2022 [1].

This quarter, the Sensor Fusion Team continued testing and improving the Multi-Object Tracking (MOT) algorithm. Last quarter was focused on using simulated data in MATLAB to fine tune the MOT algorithm; however, this quarter was focused on shifting to in-vehicle testing, which included the use of rosbags. One of the competition deliverables was to turn in data collected from vehicle testing. The in-vehicle data tests and improves the MOT algorithm, which ultimately improves the vehicle's Accelerated Cruise Control (ACC). The EcoCAR Mobility Challenge requires SAE Level 2 automation, which refers to a vehicle having acceleration and steering with the driver still engaged with the environment. Having an exceptional MOT algorithm ensures success in the EcoCAR Mobility Challenge.

2.0 – Project Introduction:

Beginning my third quarter of joining UW's EcoCAR organization, the Sensor Fusion CAV Lead, Aaron Wu, asked me to step up and be a team lead for the Sensor Fusion Data Team. From a high level, CAV primarily focuses on programming autonomous driving systems and vehicle communication. To develop and improve autonomous driving systems, the Sensor Fusion Data Team focuses on processing raw sensor data to use for the vehicle's MOT and ACC. As a lead, my project for the quarter consisted of three objectives. The first was to learn how to process and prepare rosbags to test the functionality and accuracy of the MOT algorithm. To clarify, rosbags contain data from TANK, which includes raw sensor data and sensor fusion results. Rosbag data is collected through real-world tests (i.e. Target vehicle completely stopped 250 m ahead). The second objective was to create data visualizations highlighting the accuracy of the data in the collected rosbags. Creating a data visualization ensures the effectiveness of the different radars, algorithms, and cameras on the vehicle during the real-world tests. The third objective was to develop leadership skills to lead a team of three other engineers.

3.0 - Project Description:

3.1 – Processing Rosbags

Rosbags are collected in a process known as Vehicle-In-the-Loop (VIL) testing. VIL testing evaluates the sensor fusion on the vehicle and can only be conducted during planned driving tests. Once the rosbags record the driving test, there is a process of filtering and preparing the data for useful information. As mentioned earlier, rosbag data is from the

TANK, which contains the raw sensor data and sensor fusion results. Viewing the raw sensor data and sensor fusion results consists of three steps.

First, rosbags must be reindexed. Based on our method used to record rosbags, rosbags are not "closed" properly and are in an "active" state; thus, this requires reindexing rosbags. Reindexing repairs broken bag files- typically if a bag was not "closed" effectively [2]. We can reindex by entering the below command line in the same folder as the rosbag (Fig. 1).

```
rosbag reindex [NAME].bag.active
```

Fig. 1. [NAME] is the name of the rosbag.

Next, a rosbag must be filtered. A rosbag starts recording whenever the car is turned on. As a result, a single rosbag contains data from various conducted tests; thus, we must extract the desired data, which contains the timestamps for its respective tests (Fig. 2) We must convert the times from Pacific Standard Time to UNIX timestamp to properly filter the rosbag.

```
rosbag filter [NAME].bag [NEW NAME].bag "t.secs >= [START TIME] and t.secs <= [END TIME]"
```

Fig. 2. The start times and end times refer to the time at which the test was conducted.

Finally, a rosbag must be logged to a properly formatted text file (Fig. 3). Within the text file contains rows of numbers with five columns (Fig. 4). A single row contains data in the order of time, x-position, y-position, x-velocity, and y-velocity. Each row differs by 100 ms because a rosbag collects data every 100 ms.

```
roslaunch data_logging log_bag.launch
```

Fig. 3. Command-line to log rosbag to .txt file.

```
10.00 82.625 61.625 0.0 0.0
10.10 82.625 61.625 0.0 0.0
10.20 82.625 61.625 0.0 0.0
10.30 82.625 61.625 0.0 0.0
10.40 82.625 61.75 0.0 0.0
10.50 82.625 61.75 0.0 0.0
```

Fig. 4. Result of running command-line to log rosbag data to .txt file. The above picture shows the sensor detections from 10.00 to 10.50 seconds.

Once we are able to complete the three commands, the rosbag has been properly processed and is now able to be analyzed further. If we run "rosbag info [NAME].bag," we will be able to see all the various types and topics that each rosbag contains (Fig. 5).

```
paulchung@ubuntu:~/CAV_Main/catkin_ws/src/motcan_bringup/rosbags$ rosbag info A25-2021-5-23.bag
path:        A25-2021-5-23.bag
version:     2.0
duration:    33.0s
start:       May 23 2021 11:18:50.00 (1621793930.00)
end:         May 23 2021 11:19:22.99 (1621793962.99)
size:        28.0 MB
messages:    98855
compression: none [36/36 chunks]
types:       can_msgs/Frame             [64ae5cebf967dc6aae4e78f5683a5b25]
             cavros_msgs/AlgorithmStats [47ddacc275153196d67ef443a5ffb36a]
             cavros_msgs/DetectPacket   [5608742733f0e50f86f3298bf1f15f12]
             cavros_msgs/MobilEye       [7ca5c5a806883095723a7c9aad789a00]
             cavros_msgs/SyncedData     [c9167db14384b96503838fb6aa2e0cf2]
             cavros_msgs/TrackingArrays [27f189a5022d02382fafb54e7ffd7805]
             rosgraph_msgs/Log          [acffd30cd6b6de30f120938c17c593fb]
topics:      /algorithm_stats       330 msgs    : cavros_msgs/AlgorithmStats
             /cam_det_data          330 msgs    : cavros_msgs/TrackingArrays
             /camera_data           339 msgs    : cavros_msgs/MobilEye
             /can_rx               1719 msgs    : can_msgs/Frame
             /can_rx_shi             29 msgs    : can_msgs/Frame
             /can_tx              29106 msgs    : can_msgs/Frame
             /can_tx_ni            6069 msgs    : can_msgs/Frame
             /cluster_data          660 msgs    : cavros_msgs/TrackingArrays
             /front_det_data        330 msgs    : cavros_msgs/TrackingArrays
             /radar_front_data      378 msgs    : cavros_msgs/DetectPacket
             /rosout              29452 msgs    : rosgraph_msgs/Log
             /rosout_agg          29453 msgs    : rosgraph_msgs/Log
             /synchronized_data     330 msgs    : cavros_msgs/SyncedData
             /tracking_data         330 msgs    : cavros_msgs/TrackingArrays
```

Fig. 5. Screenshot of one of seven tests from rosbag recorded on May 23rd, 2021.

3.2 – Bird's-Eye Plotting

With the properly processed rosbags, we needed data visualizations to see the raw sensor data and sensor fusion results. Initially, I used Python and Jupyter Notebooks to plot the .txt files of x-position vs. time, y-position vs. time, x-velocity vs. time, and y-velocity vs. time; however, the data visualization was not helpful in understanding how well the rosbag was recorded. Thus, after discussion with Sensor Fusion Lead, Aaron Wu, we found it best to implement a real-time plot of the data from the rosbags. Specifically, we wanted to implement /synchronized_data and /tracking_data as shown in Fig. 5. To better visualize the recorded rosbag data, we planned to implement a real-time bird's-eye plot of the sensor detections.

3.3 – Project Management

Processing rosbags and developing visualizations for the Sensor Fusion (SF) Data Team requires the guidance of a project manager. Thus, although this was my third quarter as an undergraduate researcher for UW's EcoCAR, I was promoted as the SF Data Team lead. The SF Data Team primarily focuses on processing and visualizing sensor data to improve the team vehicle's Multi-Object Tracking. As subteam lead, I was responsible for a

team of three engineers, which included delegating tasks, leading subteam meetings, working with sensor data, and developing software. Although in a work-from-home environment, I was excited to showcase my leadership capabilities and help further UW's efforts in the EcoCAR Mobility Challenge.

4.0 - Results:

4.1 – Processing Rosbags

Since this was the first quarter the Sensor Fusion Data Team was exposed to rosbags, we primarily spent time learning a new tech stack. We used VMWare, Ubuntu, Python, C++, and downloaded Robot Operating System (ROS) dependencies in order to fully complete our workspace.

The first couple weeks of the quarter was spent configuring a workspace and understanding how to process rosbags. Once comfortable with the workflow, I processed rosbags from different test dates to help CAV co-leads see how well the rosbags were recorded. Data submissions for the competition deliverables were due on May 26th; thus, to ensure the most up-to-date data, the CAV co-leads conducted driving tests the weekend before, and the SF Data Team and I reindexed and filtered the rosbags for the competition deliverables.

The most difficult part of processing rosbags was fully configuring the workspace; however, the process itself was not too difficult. I was glad to be able to take part in turning in the competition deliverables.

4.2 – Bird's-Eye Plotting

Prior to spring quarter, the SF team used the .txt files from logging rosbags to view its data. However, using the .txt files were not very helpful in visualizing sensor detections. Thus, the CAV co-leads found it most useful to implement a bird's-eye plot of the sensor detections. Most of my quarter was spent working on this data visualization.

Initially, I spent time learning how to develop software in MATLAB. After having taken classes in Java, Python, and C, it did not take much time to learn. Next, I spent time learning the message data structures and data types that are transmitted in rosbags. The most important data type from the rosbag was /synchronized_data, which included a sensor detection's x-position, y-position, x-velocity, and y-velocity every 100 ms from four different radars (front, right, left, rear) in addition to a front camera. I was able to create a prototype labeled rosbagDataViz.m that was modeled after mot_node.cpp, which parses a rosbag in C++. Initially, I perceived I was able to implement the bird's-eye plot because I was able to view multiple detections that were played in real-time; however, I later learned my implementation in rosbagDataViz.m did not parse all the detections that were recorded in the rosbag.

Thus, I created a bird's-eye implementation that parsed all the possible detections in /synchronized_data called plotDetections.m. However, this newly created file had detections that were scattered, repeated, and completely useless as seen in this GIF. Instead, for the third implementation, I only used the data type /cam_det_data, which was the data from only the front camera of the vehicle. It is important to note /synchronized_data contains data from four radars (front, right, left, rear) as well as the front camera, also known as /cam_det_data. Since /synchronized_data is difficult to parse, the team found it to be easier to parse /cam_det_data as seen in plotting_cam_det_data.m. On my third implementation, I was able to plot the detections smoothly in real-time as seen in this GIF. Since /cam_det/data had an identical data structure as /front_det_data, the front radar, and /tracking_data, which are the confirmed tracks the MOT algorithm recognizes, I was able to create plotting_data.m, which plots all three options. As seen through this GIF, the /front_det_data detections are not very good due to difficulties configuring the front radar on the vehicle. Thus, in instances where one wants different settings, I implemented the option to play different types of detections, playback speed, and bird's-eye plot dimensions.

4.3 – Project Management

This was my first quarter as a lead for research. As lead, I was the intermediary between the CAV co-leads and the SF Data Team. For the first two weeks, I answered any questions related to processing rosbags and resolved troubleshooting errors. Starting Week 3 of the quarter, I created starter code to implement bird's-eye plotting for rosbags. The link to the specification can be found here. I created the file rosbagDataViz.m as a model for the SF Data Team. Originally, this file was only able to parse the detections and not implement the bird's-eye plotting. I gave the team the task to implement the bird's eye plot in real-time; however, most of their time was spent understanding the code, and I ended up implementing the real-time plotting feature myself. As mentioned earlier, rosbagDataViz.m proved to be an unsuccessful file; however, I found this task to be a success, considering the team familiarized themselves with the existing code base, and I was humbled by my first mistake as lead. After successfully creating a real-time bird's-eye plot for /cam_det_data, I published the file on CAV's GitHub as a model for the other team members and answered any questions regarding the code, assigned questions, and any troubleshooting issues. Thus, my teammates were able to understand the code and would be able to implement more features if needed.

5.0 - Additional Work Remaining:

After creating the data visualization, there is little work to be done. There are more data types that the CAV co-leads want to implement. The different data types could be helpful in informing the team about the vehicle's MOT; however, the implementation should

not be difficult considering the data structures of the data types will be identical. Furthermore, my research teammate, Jared Yen, was able to implement a synchronized side-by-side of the bird's-eye plotting as well as the video from the same rosbag file. The file plotting_data.m had all the features fully implemented, and SF Data Team's quarterly project was complete. We were able to create a real-time bird's-eye plot of data collected from vehicle testing.

6.0 - Conclusion:

With the Year 3 deliverables turned in and SF Data Team's quarterly project complete, Spring 2021 should be viewed as an accomplishment. I was able to apply concepts learned from courses at UW- basic command-line tools, Git, GitHub, technical documentation, data structures and algorithms, and object-oriented programming. Additionally, I was able to expand my knowledge on new technologies- Robot Operating System, VMWare, Ubuntu, MATLAB, and C++. I am extremely thankful for the opportunity to research a topic I am passionate about and proud of the progress my teammates and I made this quarter.

7.0 - References:

[1] “EcoCar Mobility Challenge,” *AVTC I Advanced Vehicle Technology Competitions*. [Online]. Available: https://avtcseries.org/ecocar-mobility-challenge/. [Accessed: 01-Jun-2021].

[2] “Wiki,” *ros.org*. [Online]. Available: http://wiki.ros.org/rosbag/Commandline#reindex. [Accessed: 02-Jun-2021].