SubjuGator Navigation Tool Report

Administrative

Team Name: Project 3 33
Team Members: Alex Perez, Daniel Shmul, Charles Richardson
GitHub: https://github.com/danielshmul/DSA_Project3
Video: https://youtu.be/pTGa-WKWWC8

Extended and Refined Proposal

Problem

Robot Operating System (ROS - a framework for writing robot software) bags are recordings of live or simulated data that can be used to analyze the behavior of robots or autonomous machines. The Machine Intelligence Lab (MIL) at the University of Florida heavily deals with ROS bags so as to analyze data for their autonomous submarine. However, it seems ROS is limited on the statistical analysis methods that it offers, which can make it hard to analyze the recordings.

Motivation

Sometimes these ROS bags are filled with hundreds of thousands of lines of data that can be difficult to analyze and digest. In fact, just five seconds can record up to 1000 data points. There are some tools that help us graph desired values, however, most involve having to replay the recording, which can get tedious or time consuming. Additionally, if you choose to do any other statistics on the data, you must parse through the raw data yourself. We want to make this whole process easier.

Features

We want to create a preliminary tool that takes a ROS bag and allows for various statistical options geared towards helping MIL members analyze their robot's data which will speed up the development, debugging, and analysis curve. The problem will have been solved when we can extract the following from bags of data:

- Max/Min positional data
- Max/Min linear velocity
- Max/Min angular velocity
- Average linear/angular velocity
- Data at a point in time

Data

We used data that was generated from the MIL submarine's simulation. Given a twenty minute simulation, the simulation will produce the required amount of data (in the form of floats) that will contain:

Odometry data

a. Position (x, y, z)
b. Linear velocity (x, y, z)

c. Angular velocity (x, y, z)

For the purposes of this project, we focused on these features.

Tools

ROS is used to help obtain the data set (the simulation was built within ROS). Once we get the csv data set, everything is done in C++ and through a CLI. The command prompt is used to interface with the program.

Data Structures/Algorithms implemented

A hashmap with open addressing, using a linear probing method and O(1) average time complexity methods was implemented specifically for this project. As a secondary data structure, and something to compare the hashmap to, we refactored an AVL tree (meeting/surpassing the requirements of project 1) to work with our main data class. Algorithms included finding the min/max of velocities, average of velocities, and returning the data at a certain point in time.

Data Structures/Algorithms used

The team made full use of both the hashmap, and the AVL tree. Given that the AVL tree came from project 1, all of the team members put together their source code files and we determined whose worked most efficiently. Once that was determined, we sifted through the code and removed unnecessary outputs that were only needed for Project 1 submission purposes. We also added functions to find significant points in the data. This process removed about 100 lines of code in total. Removing approximately 300 and adding about 200. Finding the min/max was composed of an inorder traversal of the AVL tree and a sequential accessing of elements in the hashmap. The average was an extension of this functionality. Finally, finding the data at a certain point consisted of a traversal of the AVL tree and an accessing of the element in the hashmap.

Distribution of Responsibility and Roles

Fundamentally, our project had two categories: data structures, and algorithms. Charles Richardson took the role of data structure development, contributing his AVL tree source code and developing the hashmap to store and organize the data. Daniel Shmul and Alex Perez oversaw the sorting and accessing algorithms which allow the program to access minimum, maximum, average values, etc. Additionally, Alex Perez took most of the debugging responsibility when errors could not be solved by the other team members.

Analysis

Any changes the group made after the proposal? The rationale behind the changes.

The team redistributed its roles as the project developed. Initially, Daniel Shmul and Alex Perez were responsible for the back end of the project, while Charles Richardson developed the front end. The plan was to have a GUI with buttons and graphs that would map the movement of the SubjuGator. However, due to a lack of planning and a magnitude of conflicting schedules, the project start date was pushed back by quite a bit, which forced the team to be much more

conservative with features. Upon coming to terms with each other and acknowledging the mistake, the team decided to use the terminal for output and put full effort into developing a comprehensive back end. This was when Charles Richardson pivoted to develop the data structures for the project, while Alex Perez and Daniel Shmul continued planning and developing the outputs.

Complexity analysis of the major functions/features you implemented in terms of Big O for the worst case

The data structures we implemented both run at a very optimal time complexity. To begin, the hashmap is not implemented under best practice (it uses a linear collision probe), but it runs under a second for tens of thousands of data points, so it fits the needs for the project. On average, the hashmap runs in O(1) time complexity. On the other hand, the avl tree we implemented only trails behind the hashmap in computing time by a little, running in O(NlogN) where N is the number of elements in the tree.

The worst case for the AVL tree is far more likely than the hashmap, in the context of this project. Since data values are inserted sequentially. The hashmap will always insert values right next to each other, avoiding the need to probe at all, until the size gets very close to capacity. Unfortunately, for the AVL tree, since the values are inserted sequentially, the tree will need to rebalance itself often, which takes more time. Fortunately, the rebalancing is O(1), so it does not take too much more time, nonetheless, time is time.

The time complexity of the algorithms is as follows. The find min/max functionality runs in O(n), where n is the number of data points, for both the data structures. For the AVL tree this consists of an inorder traversal of all elements in O(n) where n is the number of elements in the tree and for the hashmap it would consist of n*O(1) for accessing the elements in the hashmap, similarly resulting in O(n) runtime. Since the average functionality is an extension of the find min/max algorithm, it would similarly run in O(n), where n is the number of data points, for both the AVL tree and hashmap.

Reflection

As a group, how was the overall experience for the project?

This project was a phenomenal project to give the team a sense of collaborative programming. ‘In a curriculum of individual-based assignments, this project made a big difference on my outlook of the major.’ (Charles) By using our collaborative resources - Zoom and Github - the team was able to complete the project in a matter of weeks by sitting ‘together’ and working out the problems, bugs and structures. With no concern for plagiarizing each other, or meeting any specific outputs, the team's creativity and productivity was able to flourish.

Did you have any challenges? If so, describe

Apart from the challenges described in the first part of the Analysis section above, most of the other challenges pertained to the code we wrote. For example, the original hashmap had a pseudo random permutation probe which would create an array of equal size to the container array and increment through it when looking for a place to insert a value. While testing the

hashmap, the team realized that the program crashed after a few thousand inserts. We tirelessly debugged the code, and could not find the source. Under time pressure, we decided to switch to a linear probing function, which would sacrifice time efficiency, but worked nonetheless. Upon making this change, we noticed that the hashmap worked. Not only was it fast, but it was still faster than the AVL tree. Other challenges included combining our code for the AVL tree to achieve the best runtime and generalize the data structure to contain our odom wrapper class (contains data for each point). Additionally, developing and debugging the algorithms posed another challenge which presented some roadblocks but was ultimately achieved.

If you were to start once again as a group, any changes you would make to the project and/or workflow?

In future development cycles, we plan to change the rate at which we complete work, from an exponentially increasing to a constant rate throughout the timespan. It is not enjoyable to have so much work last minute, proper time management will help the team get there. Additionally, doing plenty of research ahead of time will help the team meet the initial goals we develop. If we were to do more research on building a GUI when Charles first came up with the idea, then it would be possible that the project would have one.

Comment on what each of the members learned through this process.

Charlie's Learnings: Leading a team is not about assigning roles, it is about letting people assign their own roles, and helping them fulfil their roles as well as they can by doing your own as well as you can. Github is not as confusing as it seems upfront, and it's actually very helpful. Plans always help when they are done early.

Daniel's Learnings: I gained a lot of experience with github which was very useful throughout the project and will come in handy for future projects and collaborations. I really like the experience of working in a group and the team dynamic was really good. Coming out of this project I have more experience with time management and leadership when working with others.

Alex's Learnings: I gained some more experience using git and github, and I learned more in depth how exactly a hashmap works when implemented in C++. This project also helped practice and learn going through the process of reading and understanding another person's code, which is a very valuable skill to have in industry.

References

- Machine Intelligence Lab software was used to obtain the data for this project.