SP-13
ELECTRIC VEHICLE TEAM
FINAL REPORT DRAFT
JAMES RIVETT
2:00 PM SECTION
https://ksu_evt.gitlab.io/

Table of Contents

Abstract

In 2021, roadway fatalities per capita reached the highest they have been since 2005 [Person]. With a 10.5% increase in motor vehicle-related deaths from the previous year, many companies from both the automotive and technical field are proposing computer-driven autonomy as a solution. As is the case with many other applications, computers can make much faster decisions than humans can. In addition, they often make better decisions due to the depth and breadth of information they can access and process in real time. On the surface, it is intuitive that this ideology should be applicable to motor vehicle transportation. That said, the technology isn't quite there yet. If history has taught us anything, especially in the automotive field, it is that competition drives innovation. The inter-garage turmoil and outright desire to win from racing teams throughout motorsport history are all responsible for the technology we use to get from point A to point B every day. The goal of the Electric Vehicle Team this year is to improve our autonomous go-kart 'Voltron' in hopes to defend our EVGP title this year at Purdue University.

Background

ROS2/VNC Environment

When solving problems with robotics, developers (ourselves included) often utilize a wide variety of technologies on every level of the stack. On the hardware levels, different components communicate and output data using a wide variety of protocols and physical connections. On the software levels, using this data from each component in a meaningful way is often difficult because they come in at different rates and formats.

The Robot Operating System (ROS2) framework aims to consolidate the use of these components. The term 'Operating System' in ROS2 is a misnomer in this case because it does not run on the bare metal of the robot's computer as one might expect. Instead, it runs as a regular process on an x86 machine. This allows it to serve as a 'Puppet Master' in the sense that it can handle all communication, control, and synchronization between the different hardware and software on our stack via a publisher/subscriber system.

While this is extremely useful for our application, as stated before, ROS2 runs on an x86 machine, which raises some issues. Since it uses many low-level libraries to seamlessly communicate with various hardware and software, ROS2 always needs to have access to the EXACT same set of libraries and dependencies wherever it is run. Because of this, ROS2 only runs on a specific LTS release of Ubuntu; one that is guaranteed to have all of these necessary libraries.

The issue is this: not everyone on EVT has the excess hardware, Linux experience, or even time to install the specific version of Ubuntu needed to run and develop for ROS2. To remedy this, the team developed a docker-based workspace that allowed users of any Linux OS to develop for Voltron's software stack. It is a container (essentially a special Virtual Machine with direct access to the machine's hardware) running the correct version of Ubuntu that has special GUI and hardware acceleration capabilities via an open-source project called x11Docker. The only issue with this approach is that x11Docker only works on Unix-Based systems, and many of the tools we use inside the container only work on Linux systems specifically.

To remedy this issue, it was decided that the first deliverable was a new workspace image that re-implemented all of the functionality we needed from x11Docker, but in ways that were compatible with docker on other platforms. If we could achieve this, it would allow developers on any operating system to write ROS2 code for Voltron's software stack.

Steering System Improvement

Our current autonomous system is chiefly controlled using Model Predictive Control (MPC). This model considers multiple observed state variables in addition to the current input state [Rawlings] to forecast which control commands would bring the kart closest to the line it is trying to follow. The inherent issue here is that our steering system is controlled by a commanded number of motor rotations, but our MPC outputs a steering angle.

Once the MPC predicts the optimal control commands, there is a separate controller that actually sends those commands to the steering system. As it is currently, that control unit is what handles the translation from steering angle to motor position. This is a point of inefficiency because this control unit is detached from the x86 PC. It is instead running on a Teensy Microcontroller, which has limited

processing power. Instead of wasting resources on that chip, we plan to instead do the translation on the master x86 PC.

Most importantly, this translation function is currently just an estimate; there was no empirical data that motivated it and is therefore not as accurate as it could be. To remedy this, we plan to study the effective angle of the steering joint using motion capture technology, fit a function to that data, then use its inverse to get a motor position as the output when a requested angle is given as an input. This means the resulting steering angle from the control unit should be much closer to what MPC deemed optimal.

In addition to a good translation from steering angle to motor pose, we also don't have much information on the general capabilities of the steering system. Factors such as Ackerman ratio, maximum steering rate, and alignment all have a large impact on the dynamics of how the kart performs on a hotlap.

One semantic issue of this study is that Voltron has Ackermann Steering. This is an intentional design decision of many steering systems across various applications of motor vehicles. This term describes the phenomenon in which the inside wheel steers more sharply relative to the outside wheel. This is preferable because the outside wheel will have to travel a wider radius, and therefore longer distance than the inside wheel. [Simionescu] This means, however, that the angle of one steering joint is not guaranteed to match that of the other joint at any point in the steering range. In order to calculate the effective steering trajectory produced by both wheels in tandem, the average of the two angles is taken.

Requirements

ROS2/VNC Workspace Container

- There must be a streamlined process for starting the VNC-based workspace that is similar to, or just as easy as the old one
    - This process must automatically start a VNC server inside the container so the user can connect to an actual desktop environment right out of the box
    - This process must be documented and reproducible on any machine
- All functionalities must work seamlessly given Host OS is UNIX
    - X11Docker assumes Linux, not just UNIX which leaves out Mac
    - Can be run under WSL if the host is using Windows
- All functionalities/tools of the old workspace itself must be present in the new one
    - ROS2
    - ROS2 Dependencies
    - RVIS
    - Gazebo
    - Python
- All functionalities of X11Docker that were necessary for the old workspace must be present in the new one
    - Persistent user/UID
    - Persistent home directory
    - SSH identity forwarding
    - Automatic dependency installation for our modules

    - The container doesn't need native performance, but hardware acceleration for things like RVIS and Gazebo would be nice

Steering System Improvement

- Gather positional data of the steering joint at various steering motor poses
    - This will be done via a 'recorder' node
    - This node will automatically advance the steering motor's position by small increments
    - Between each increment, the node will also record the quaternion data from the motion capture software
- Flatten quaternion data from the motion capture software into a 2-D yaw value
    - Move the pivot point of the rigid body onto the axis around which the steering joint actually rotates in real space (where it's attached to the frame)
    - Assume an isometric top-down view
        - This allows to ignore the camber and caster
    - The observed rotation can now be seen as the effective steering angle on a plane parallel to the ground plane
- Fit a function onto that data
    - X (input) is the motor pose we commanded
    - Y (output) is the resulting steering angle for each motor pose
- Invert that function
    - X (input) is the steering angle MPC deems optimal
    - Y (output) is the motor position that the control unit needs to command in order to achieve that steering angle
- Mirror and average
    - We only have time/resources to do the study on one of the steering joints
    - We need positional data from both so we can average them to get the effective steering angle
    - Luckily, our steering system is designed to be symmetrical, so we can simply mirror the data from one to get the other
- Replace the 'guestimate' translation function with the one that was fit from the angle data
- Move the translation process off the microcontroller and on to the x86 main PC

Development

ROS2/VNC Workspace Container

The first action item for development of the new workspace container was getting the VNC server installed in the container by default. This is a bit difficult because the server we are using (TurboVNC) does not have a listing in the official apt repositories. This means we must install it from a .deb package. This may not seem like an issue at first, but the goal of this new container is to have an easy startup, so we don't want to make the users download the VNC server package. In addition, we want to ensure that the correct version of the VNC server gets installed in the container. The solution route we chose was to upload the .deb package into the workspace repository on GitLab via git-LFS (more on that in version control section). Uploading large files to git repositories will often cause a leak in size due to the entire file being recorded in history every time it changes.

With the VNC server installed in the container image, it was time to begin building the QOL for the workspace. In order to gain an idea of the things that needed to be implemented, I used the new workspace environment exclusively when experimenting with the other parts of Voltron's stack. This exposed issues such as the default terminal not being set correctly, missing functionalities due to the image being a minimized version of Ubuntu, and even some configuration issues with the VNC server itself.

Once everything inside the container was in a usable state, it was time to implement all of the housekeeping that x11Docker used to handle. Since the plan was to allow users to interact with the container in a fashion similar to a traditional VM, we had to extend Docker's functionality in order to provide this experience.

One of the first items was a persistent home directory; this would allow the user to return to their local work after shutting down and restarting the container image, even if that specific instance was terminated. I did this by virtue of Docker volumes which can be bind mounted into the container at any directory path desired. When the container can carry the home directory into each instance of the container, it allows other things than just files to persist as well, which we'll cover here.

As aforementioned, persistence of the environment variables and user configuration is often desirable as well; that was one of the core functionalities implemented in x11Docker. Since most of a Linux user's data is stored in their home folder, we were able to simply name the Docker volume in tandem with the uname, then ensure a new user with that uname and UID gets created each time the container starts.

One last major functionality I had to re-implement from x11Docker was SSH agent forwarding. When developing for and testing Voltron, we often have to use SSH to connect to the kart PC. We also use many tools that make use of SSH to get data from the web such as Git. Luckily for me, this was an already-documented process, so I simply had to implement it in our own workspace. I would simply create a bind mount of the SSH authentication socket from the host machine into the container just like the home directory. After the container is started, Docker then mounts the persistent home directory and creates a symlink inside of it that points to that authentication socket bind mount. This means any SSH commands inside the container will be executed with the same identities as the host machine.

Steering System Improvement

As mentioned in the background information section, one of the most useful pieces of information on our steering system that we could have would be a more direct translation from the real position of the steering joint to the commanded position of the motor that drives it. Luckily, the motor we use to drive the steering system is controlled by an ODrive, a controller that exploits an encoder in order to treat a regular motor like a servo, giving us direct control over how many rotations it should complete. ODrive also provides a Python library that not only facilitates the control of this system, but offers many debugging and statistics tools as well. This is the tool we use to command the steering system during normal operation, but we can also automate the movement of the steering system during testing. [Appendix, 'Steering Mocap Study Videos']

With the positional data of the motor handled, we now need a way to record the effective yaw of the steering joint relative to the ground plane. While there are many physical tools we could use to measure this data, things like the camber and caster of the steering joint make it difficult. This is because camber and caster, by definition, change the orientation of the steering axis to provide more cornering grip.

MPC outputs a steering trajectory that is agnostic to camber and caster, so we cannot simply use a protractor fixed to the steering joint; the yaw measurements need to be parallel to the ground plane. To remedy this, we can employ motion capture technology to record the positional data of the steering joint, then measure the yaw from an isometric top-down view to see the effective steering angle parallel to the ground plane. [Appendix, 'Steering Mocap Study Videos']

We can extend the software that automates the moving of the steering motor such that it also records the positional data from the motion capture software. This ensures synchrony between both measurements. If we wait until the process finishes recording the positional data before advancing to the next steering position, we can be sure that each entry for motor positions does in fact correlate with the motion capture data recorded at the same timestamp.

Implementing the recording of the motion capture software was actually the easier part of the process. When it came time to perform the study, we ran into a few different issues. First being simply that we were new to the software. Much of the measurement accuracy depends on how well we set up the rigid body and pivot point; doing that in the software wasn't entirely intuitive and the documentation was quite sparse given how much money we spend on it. Regardless, after much tribulation, we got our steering joint correctly defined as a 3-D rigid body in the software.

When we ran the automated study, we got some extremely noisy data [Appendix, Fig. 5a] in certain parts of the sweep from full-left lock to full-right lock, and we experienced that noise at the same points every time. This was due to multiple reasons mainly relating to the infrared cameras' ability to see the reflector balls we placed on the steering joint. Our plate that mounts the Velodyne LIDAR system was obstructing some of the cameras in the array at certain points throughout the sweep. To remedy this, we simply removed the plate. Voltron is also predominantly made of sheet aluminum which is extremely reflective. So reflective in fact, that it is enough to surpass the noise/signal ratio that the motion capture software uses to detect reflectors. To remedy this issue, we disabled all lights in the room except for the infrared ones on the cameras themselves, and covered the reflective parts of the kart with extra pieces of clothing that the present team members had on hand.

Analysis/Results

ROS2/VNC Workspace Container

With the container completely working on Linux, I was ready to begin testing it under other operating systems. Because of Windows' popularity among the team members, the most common host OS would be Ubuntu under WSL. This is the first case we tested, and it immediately presented some strange issues. The environment was working perfectly on the same version of Ubuntu running on bare metal, but the connecting to the VNC server under WSL would only result in a black screen for the client. [Appendix, Fig. 2]

The debugging process took about two weeks and after solving a multitude of problems with the Xserver as well as the VNC server itself, we still got nothing but a black screen when connecting to the VNC server. Since many of the Windows users on the team were already working on projects that didn't explicitly require the ROS2 workspace, we decided to abandon efforts on the VNC/ROS2 workspace and move on to the steering study. While we didn't achieve the main goal of platform-agnosticism, I learned a lot about Docker during the process and we can still run the VNC server container on the EVT server

for people to connect to remotely. It's not running on their machine, but Windows and Mac users do have a workspace to use if they need it.

Steering System Improvement

It was noisy at first [Appendix, Fig. 5] but to our surprise, after implementing these remedies, the data we got from the study had very little noise. [Appendix, Fig. 4] The raw output from the motion capture recorder was quaternion data, so another team member wrote a script that flattened that quaternion data as if we were viewing it from the isometric top-down view mentioned before. After this flattening, we were left with 6 scatter plots for each run where X was the input motor command and Y was the resulting steering joint yaw in radians.

With the raw data gathered, the hard part of the study is over. Now, there are three major steps left: mirroring the data to mimic the other steering joint, taking the average between those two data sets, then fitting a function to that. Once that is done, the inverse of that function can be used to translate a desired effective steering angle to a motor pose.

Project Planning and Management

Communication

Most team communication for EVT is handled through Discord; a free chat app that was originally meant for gamers but has vastly expanded its functionality due to various other communities making use of it to keep in touch. EVT has a server with different channels for departments within the team. I mostly work with the software department, and therefore spend most of my time in that channel. To keep things orderly inside that channel, we create threads that pertain to different topics or projects within the software department. Discord also allows administrators to assign roles and nicknames to people so it's easy to get someone's attention in a day-to-day chat.

Meetings

Meetings (on all levels) are generally held at Wilder Communications Center on the Marietta Campus. Meeting in person not only allows for better interpersonal communication, but also enables us to physically model or draw the things we're talking about. This is important because everyone involved has very different backgrounds and we all interpret ideas very differently. Most meeting notes are stored on the BookStack instance running on the server in Wilder:

https://wiki.ksuevt.org

Other things discussed in meetings often get diagrammed/drawn on whiteboards. If something is relevant to the project we're working on, we're responsible for documenting the knowledge. Usually, this simply constitutes taking a picture of the whiteboard [Appendix, Fig. 1].

Trello/GitLab Issues

Prior to this semester, all tasking and scheduling was handled through Trello.

https://trello.com/w/ksuevt

Most departments on EVT still do use Trello, but Software has pivoted to using GitLab issues as we can more closely integrate it with our pull requests and code commits (More on this in the next section).

https://gitlab.com/groups/KSU_EVT/autonomous-software/-/issues

Version Control

Version Control was handled through a GitLab organization. This is not only because we can have multiple projects (repositories) for the different modules of the software stack, but also because it provides us container and artifact registries with which we can tag and release working components for the stack.
https://gitlab.com/KSU_EVT/autonomous-software

We also elected to implement git-lfs (Large File Storage) for storing the .deb file of the VNC server in the workspace repository. This is because tracking each change to a large binary file would quickly bloat the git history; so much so that it could be in magnitudes of gigabytes within a few changes. Git-lfs instead stores a reference hook to the large file stored on a different server, then simply curls it when checking out a branch with an lfs-tracked file.

https://git-lfs.github.com/

Test Plan/Test Report

VNC/ROS2 Workspace Container

Test Plan:

- Run the workspace container on the following Operating Systems:
    - Ubuntu on Windows via WSL
    - MacOS
- Connect to the VNC Server that was started in the container
- Perform standard development operations to get a short-term result
- Allow team members to use the workspace for extended periods of time to uncover bugs

Test Report:

- Everything fine under Ubuntu [Appendix, Fig. 2]
- Attempted to run the workspace on Windows with WSL
- Container started properly and everything is functioning, VNC Server starts as well
- Connecting to the server only warrants a black screen [Appendix, Fig. 3]
- Same result on MacOS
- After weeks of debugging, most members that would've used it moved on to projects that do not require direct development with ROS2, so we abandoned the project

Steering System Improvement

Test Plan:

- Minimize errors (Likely MSE/RMSE) from fitted function
- Verify accuracy with motion capture software
    - Ask control unit for steering angles
    - Use motion capture isometric view to ensure the steering angle is what we requested
- Verify usability with MPC

    - No changes to MPC node necessary
    - Run same testing procedure as usual for MPC
    - Record times around standardized GPS circle
    - Compare with baseline time using old control function

Test Report:

- Not finished implementing yet, so no tests have been run
- Despite the semester being over, competition is in May
- I plan to stay involved with EVT as an Alumni not only to tie up these loose ends, but to help compete and defend the EVGP title

Appendix

Whiteboard Capture of VNC/ROS2 Container Architecture

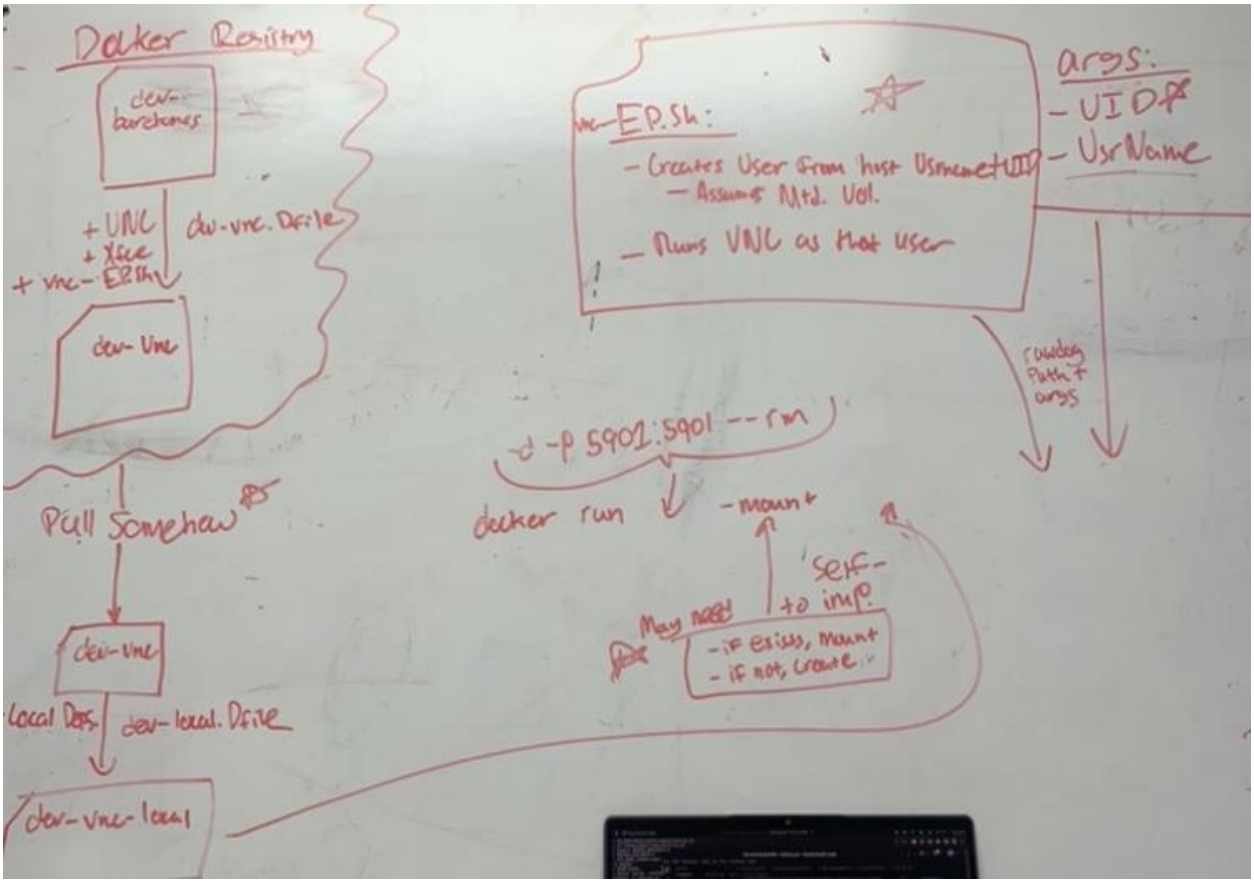

Figure 1: VNC/ROS2 Workspace Container Architecture

VNC/ROS2 Workspace Container Testing

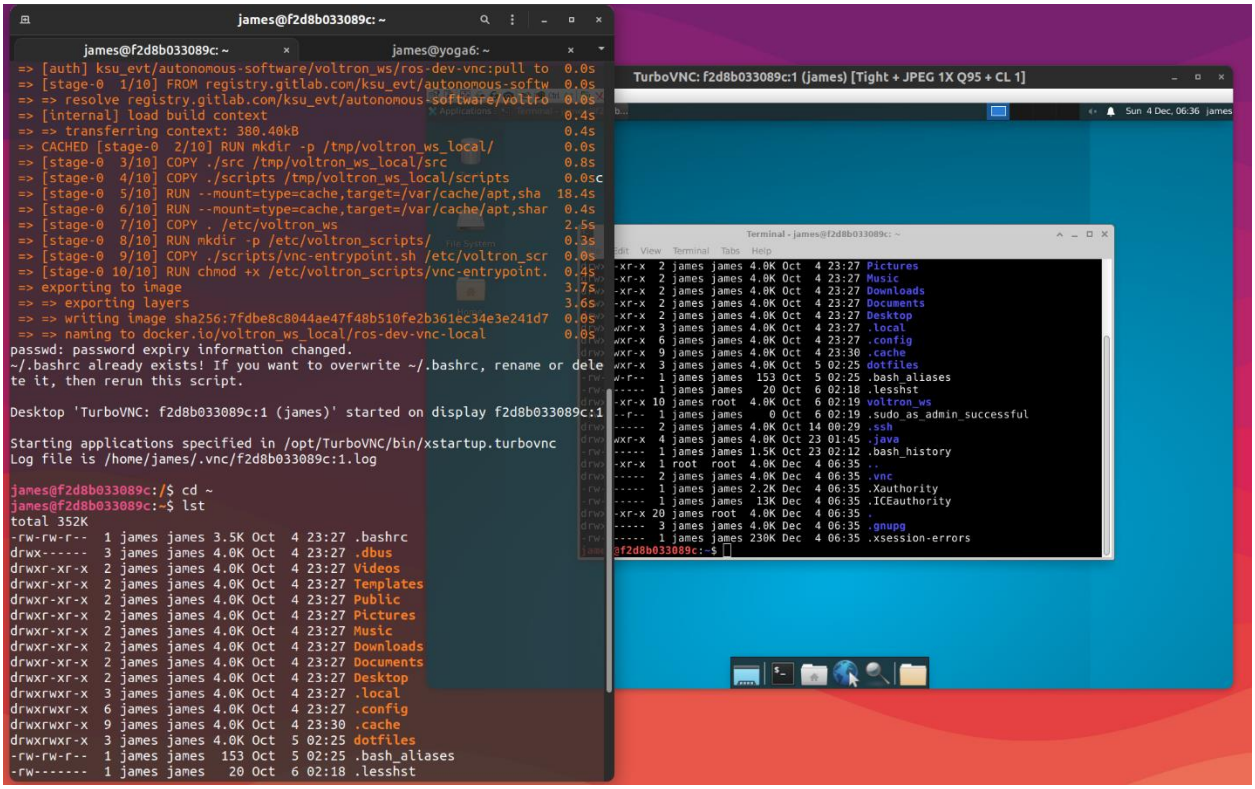

Figure 2: Container runs fine under Unbuntu

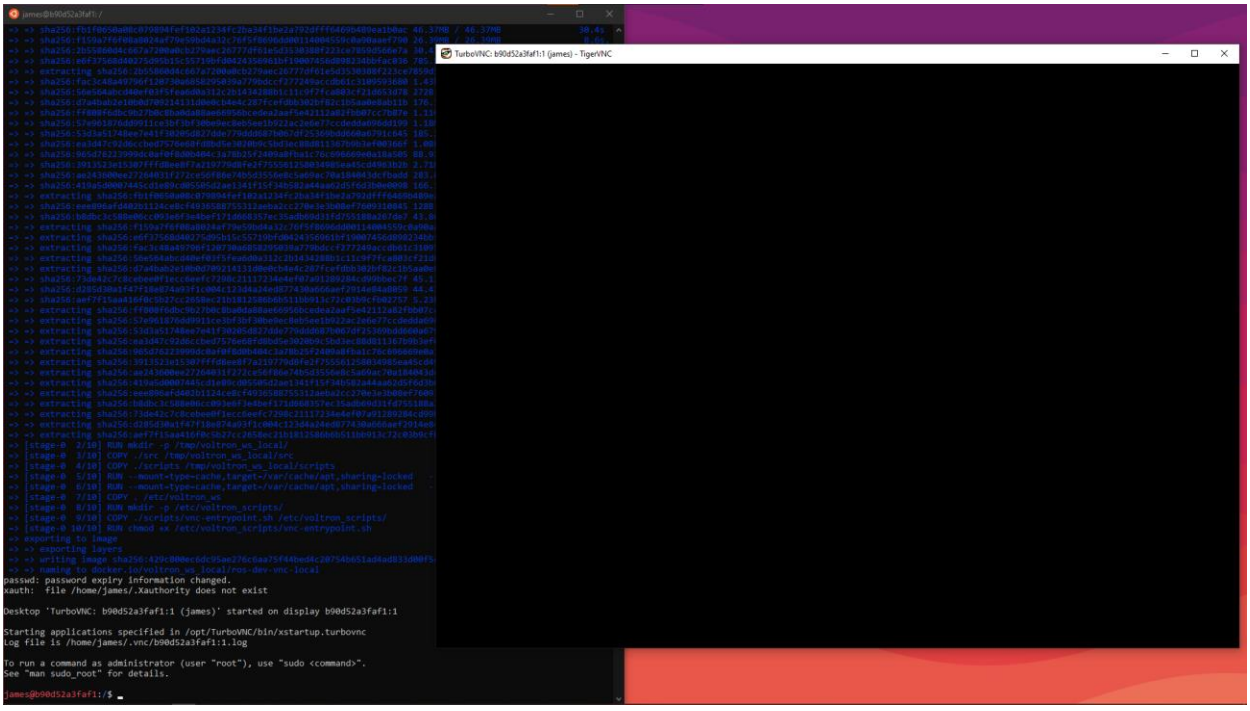

Figure 3: Running the container under WSL only yields a black screen when connecting to the VNC server

Steering Mocap Study Videos

https://kennesawedu-my.sharepoint.com/:v:/g/personal/jrivett_students_kennesaw_edu/EWcieE2t9QNAlApJ4Sk6x3sB9uhrt3kFvJYEg6kUxHUp0g?e=BMkczC

https://kennesawedu-my.sharepoint.com/:v:/g/personal/jrivett_students_kennesaw_edu/EQE4lv-nmkRFo1AxqdMEgdoBnopMH_JEJPvk7ec5aofNBw?e=lemH6o

https://kennesawedu-my.sharepoint.com/:v:/g/personal/jrivett_students_kennesaw_edu/EePMKtFtKmxDiScwYbZ5gh4BFHplYJQM-hPNK1CUoL85Ag?e=l40fpI

Steering Mocap Study Data

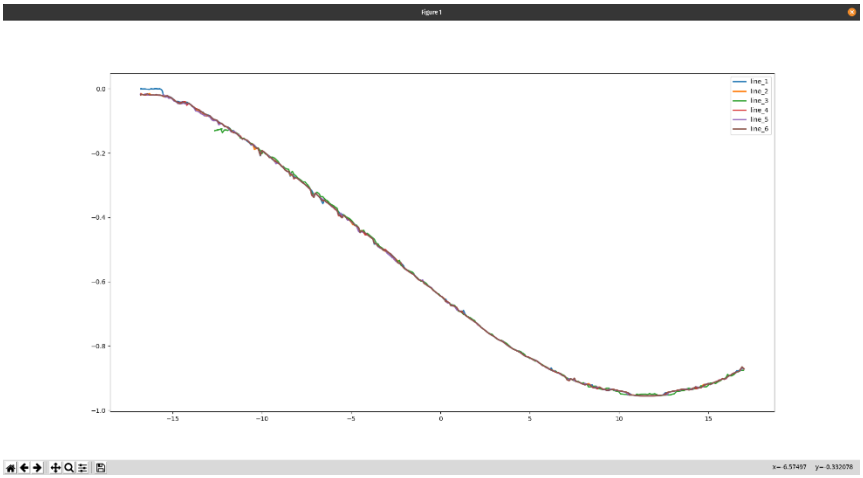

Figure 4: Flattened Yaw Data From Study

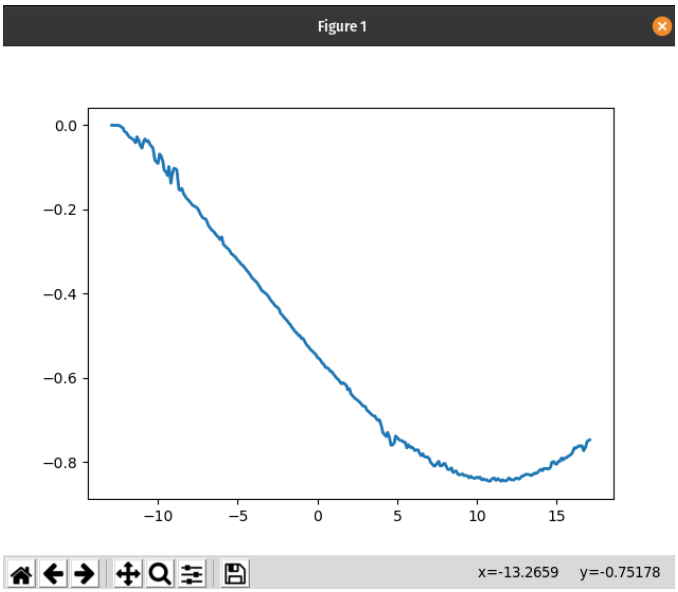

Figure 5: Motion Capture Data before De-Noising Measures

REFERENCES

Person, and David Shepardson. “U.S. Traffic Deaths in 2021 Jump to Highest Number since 2005.” Reuters, Thomson Reuters, 17 May 2022, https://www.reuters.com/world/us/us-traffic-deaths-jump-105-2021-highest-number-since-2005-2022-05-17/.

J. B. Rawlings, "Tutorial overview of model predictive control," in IEEE Control Systems Magazine, vol. 20, no. 3, pp. 38-52, June 2000, doi: 10.1109/37.845037.

Simionescu, P.A, and D Beale. “Optimum Synthesis of the Four-Bar Function Generator in Its Symmetric Embodiment: The Ackermann Steering Linkage.” Mechanism and Machine Theory, vol. 37, no. 12, 2002, pp. 1487–1504., https://doi.org/10.1016/s0094-114x(02)00071-x.