Final Internship Report

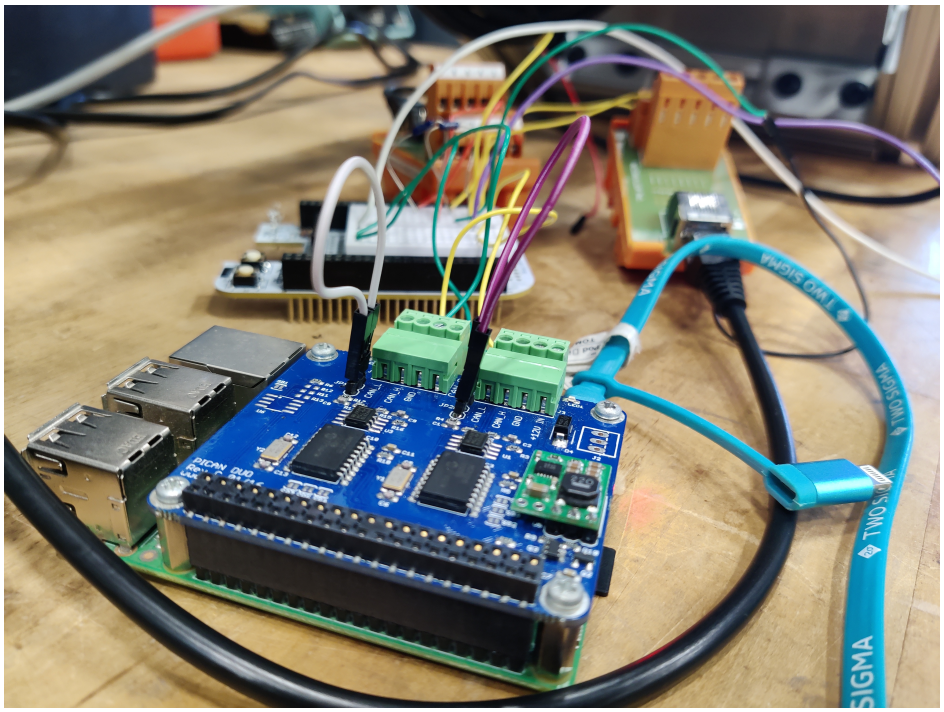
Arjun Vedantham
Argonne National Laboratory
Mentor: Kevin Stutenberg

Summer 2022

Mentor Signature: Kevin Stutenberg

Abstract

The Advanced Mobility Technology Laboratory at Argonne National Laboratory conducts testing of vehicles on a ”chassis dynamometer” - a device used to simulate a road surface during a driving cycle. In order to drive the car remotely, they use a robotic driver system. The existing control scheme for this robotic driver system uses an analog control signal to control two electromagnetically propelled linear actuators, but this requires manual calibration before each test cycle and provides no direct feedback data. The purpose of this project was to migrate control of the robotic driver over to a digital, CAN protocol based control system. After investigating new control methods, we developed a ”CAN2Serial” software package, which is capable of accepting CAN commands from the testing equipment and translating it into commands sent over an RS232 serial bus to the actuators. Additionally, CAN2Serial is able to simultaneously retrieve actuator position data. This allows for more precise, repeatable, and direct control over a vehicle during a test cycle.



CAN2Serial testing configuration

1 Introduction

This summer, I was tasked to come up with a CAN based control system for controlling the robotic driver system used during chassis dynamometer testing at the Advanced Mobility Technology Laboratory at Argonne National Laboratory. I was excited about this project because I have been interested in robotics for a long time, and because I wanted to apply what I had learned in my robotics programming classes to a real world problem.

2 Description of the Research Project

The Advanced Mobility Technology Laboratory is a group within the Transportation and Power Systems division of Argonne National Laboratory, performing in depth testing of cars, trucks, and other vehicles using ”vehicle in the loop” capabilities to quantify energy use simulated environments. To conduct vehicle in the loop testing, AMTL uses a chassis dynamometer, where a car is mounted on rollers that simulate a driving surface. AMTL uses either direct communication with the vehicle, or a robotic driver to press against the vehicle's pedals in order to drive during dynamometer testing. However, the robotic driver uses an analog control signal, which required manual recalibration before each run, and did not provide any feedback about the actuator position. My project was to develop a system for controlling the robot driver over a CAN connection, which is a low level, message based protocol frequently used to communicate between different components in modern vehicles.

3 Contributions Made to the Research Project

I started developing this new control system by reaching out to the manufacturers of the amplifier modules used with the linear actuators that made up the core of the robot driver. After speaking with them, I determined that there were two methods of developing this new control system. The first involved using the Copley CANOpen C++ library, written by Copley Controls, the manufac-

turers of the Xenus XTL-230 amplifiers that were used in the robot driver. This allowed for control of the actuators over CANOpen, an industry standard protocol frequently used in these kinds of industrial automation cases. However, using this software would require an expensive license fee and a restrictive usage agreement. The other approach involved using the serial library provided by the manufacturer, which was freely available and did not have a license restriction. This would involve sending messages to the amplifiers over a built-in RS232 serial interface located on the front panel of the amplifier module. After giving a short presentation on the available options to my mentor and some of the other researchers at AMTL, we decided to build out a software package capable of converting CAN messages from the dynamometer data acqusition controller into messages that the amplifiers could understand.

After deciding on this path, I began exploring how to send and receive CAN messages. I originally started with a Waveshare CAN/RS485 shield, which is a printed circuit board that could be mounted on top of a Beaglebone Black, a popular standard single board computer. However, the Waveshare shield had poor community support, and was plagued by old, unmaintained software packages. Shortly after this, I switched over to using a Raspberry Pi and a PiCAN 2 Duo shield, both of which had much better documentation and community support. In addition to the Raspberry Pi, I also used an Arduino Leonardo with a Sparkfun CAN shield on top as a stand-in for the data acquisition controller system typically used by AMTL during vehicle testing.

Once these devices were able to communicate over a CAN network, I moved on to developing software for the actuators themselves. Through the serial API, I wrote a program in that would read and send commands from a text file, essentially a ”script” for the amplifiers to follow. Once this was working, I start working on combining this functionality with the CAN message work that I had previously completed. Soon, the amplifiers were able to move the actuators in response to a CAN message that encoded the new position requested by an operator.

However, this software had to do more than just send translated CAN messages to the amplifiers. In addition to this functionality, this software (called ”CAN2Serial”) had to poll - or regularly request - the amplifiers for the current position of the actuators. This was because the serial API had no mechanism for the amplifiers to send a message on their own - all communication was in response to a sent request. It also had to be able to detect any dangerous fault cases that could occur during operation.

Initially, I started by just setting up a script to repeatedly poll the amplifiers for their position. According to my mentor, the desired update frequency would be 100 Hz, so CAN2Serial would have to send a position request, receive a response, and send the resulting value over the CAN bus in less than 10 ms. However, this process took well over ten times as long, due to a bug in the standard serial read/write functions provided by the manufacturers of this amplifier. After resolving this issue, the CAN2Serial software was able to retrieve and send position updates in less than 10 ms, and was able to process incoming position commands in 20 ms, well within operational limits.

After this was working, I moved onto combining each of these individual tasks into a single program. Originally, I tried to use multithreading to run these tasks concurrently. However, this posed a problem - thread scheduling is not deterministic, meaning that programmers do not have exact control over when a thread runs. This was problematic, because it meant that incoming CAN commands could go unprocessed until the next time the OS scheduler placed that thread on the CPU for runtime, causing unacceptable delays in command processing. After thinking through alternatives, I rewrote the multithreaded code using an asynchronous multitasking technique, which removed the need for thread preemption and time intensive context switches. This resulted in the CAN2Serial package being much more responsive to new position commands while simultaneously reducing the downtime in between position updates sent to the data acquisition controller.

Finally, I implemented fault handling. Originally, this software was designed to quit as soon

as it ran into a critical error, but after doing some limited tests with some of the other researchers at AMTL, I quickly realized that this would not be conducive for long testing runs. Instead, I implemented a new fault handler that would stop CAN2Serial safely, continuously transmit an error message on the CAN interface, and await error reset commands. Now, instead of having to log into the Raspberry Pi and manually restart CAN2Serial through the command line, an operator could simply ”reset” each of the errors by sending a CAN messge to the Pi indicating which errors should be dismissed. Once all of the detected errors have been dismissed, CAN2Serial automatically reboots itself and the amplifiers, allowing the operator to safely recover from an error state without extensive intervention.

After developing these features, I worked with the other researchers in the AMTL group to test this software on a 2019 Hyundai Sonata Hybrid that was mounted on AMTL's two wheel chassis dynamometer. I also gave a presentation to the other researchers detailing my work throughout the summer.

4 New Skills & Knowledge Gained

Through this summer, I was able to apply much of the skills that I had learned in classes (particularly my concurrent programming class), as well as the skills that I learned as part of UMDLoop, an undergraduate engineering team at my school, the University of Maryland at College Park. This included refining my skills with socket programming and basic digital electronics. In addition, I gained new insight into developing an asynchronous, concurrent software package for use in a real-time computing environment. I also learned a little bit about the scientific computing stack used by AMTL, including dSPACE and LabView.

5 Research Experience Impact

I have been interested in pursuing graduate school to perform this kind of research and development work, and my time at Argonne National Laboratory has only reinforced that interest. I really enjoyed working in this capacity, and I have only become more certain about pursuing a graduate school education in computer science and robotics.

6 Relevance to the mission of DOE

This research is critical to the mission of DOE as it focuses on improving the research environment needed to make efficient and autonomous vehicles a reality. AMTL's research is essential to developing autonomous driving systems that eliminate many of the inefficiencies currently associated with road-based transportation, and my project this summer will help to make that research easier, faster, and more accurate.

7 Acknowledgements

Thank you to my mentor this summer, Kevin Stutenberg, for his guidance and support throughout this summer. Additionally, thank you to Miriam di Russo, who has been essential in evaluating CAN2Serial's functionality in both tabletop and dynamometer test configurations.

References

[1] Xenus XTL-230 Manual, Copley Controls, 2018. https://www.copleycontrols.com/wp-content/uploads/2018/02/XTL-XSJ-Xenus_User_Guide-Manual.pdf

[2] ASCII Programmer's Manual, Copley Controls, 2018. http://www.copleycontrols.com/wp-content/uploads/2018/02/All-ASCII_Programmers_Guide-Manual.pdf

[3] BCM2385 Peripherals, Broadcom, 2012. https://datasheets.raspberrypi.com/bcm2835/bcm2835-peripherals.pdf