Alexander Hansen

Vehicle Telemetry Communications

Abstract

In prototype vehicles, a lot of information is gathered from sensors place all over the vehicle, and multiple CPU's may be necessary to process the data because of location constraints. A reliable and predictable method of communication between these processing units and the CPU which handles and stores the telemetry is necessary. The telemetry system must also be able to perform basic math on the raw data, or throw out unnecessary data.

Introduction

Using an Ethernet network and real time tasks, I was able to accomplish a predictable communication stream between two CPUs. Ethernet was chosen because of its reliability and error handling built into its protocol. Real time tasks were chosen because it improves reliability of the messages and provides a predictable timing between messages. It also ensures that the operating system has enough time to write information to a CSV file each time data is read from the stream.

Technical Background

I got the idea behind this project from problems that I encountered while working on the University of Missouri's Hydrogen Car Team. The fuel cells we used provided large amounts of data, much of it which was useless to us, and other custom hardware to supplement this data. A method was needed to communicate between our custom hardware and fuel cells to a telemetry system. This project shows how two CPU's can communicate, one providing data, and the other receiving, processing, and storing it. Real Time Tasks are used to ensure the predictability of the program. It is worth the sacrifice in speed to know that data comes in at well defined intervals. The network uses sockets, which are a special type of pipes, to exchange information between CPUs. Pipes are data structures that allow information to be passed between tasks on an operating system, on a higher level where the tasks are on different hard wares, they become known as Sockets.

Implementation

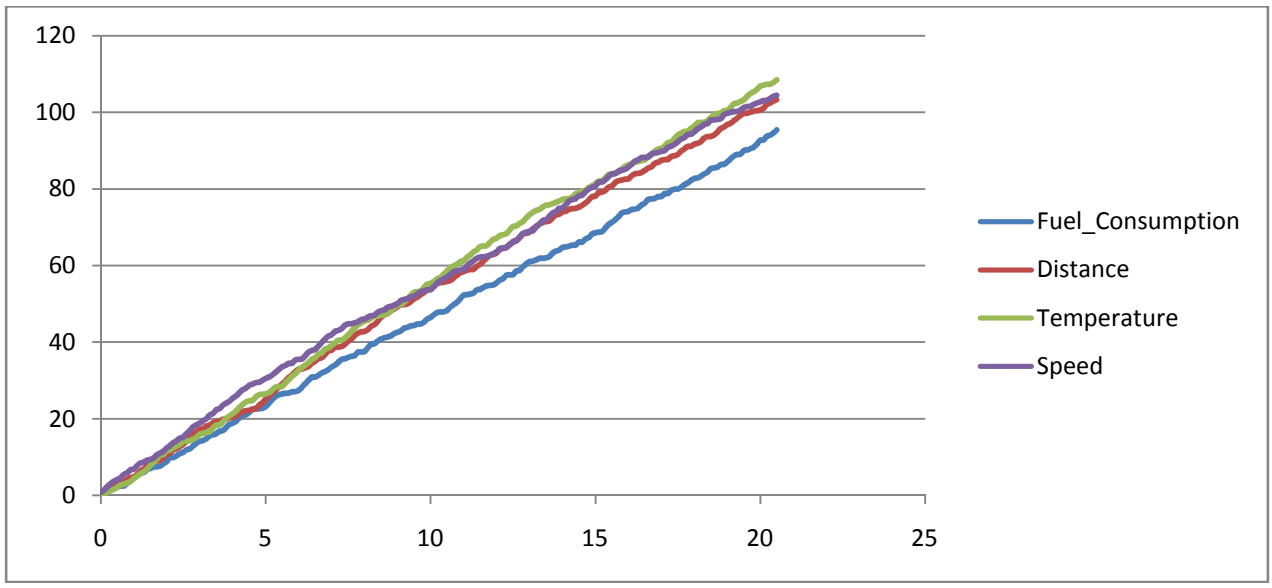
Two TS-7250 development boards are used as platforms for each part of this project, one board randomly generates data and sends it to the other, while the second board receives the data, processes it, and saves it to a CSV file on the board. The data generation software is the simplest, as its only purpose is to broadcast 9 pieces of floating point information one at a time over the network. The data is simulated by adding a random number to a base value which updates over time. It then waits for the receiving board to send a request for more data. The receiving board is more complicated, since it is where most of the work is done. There are

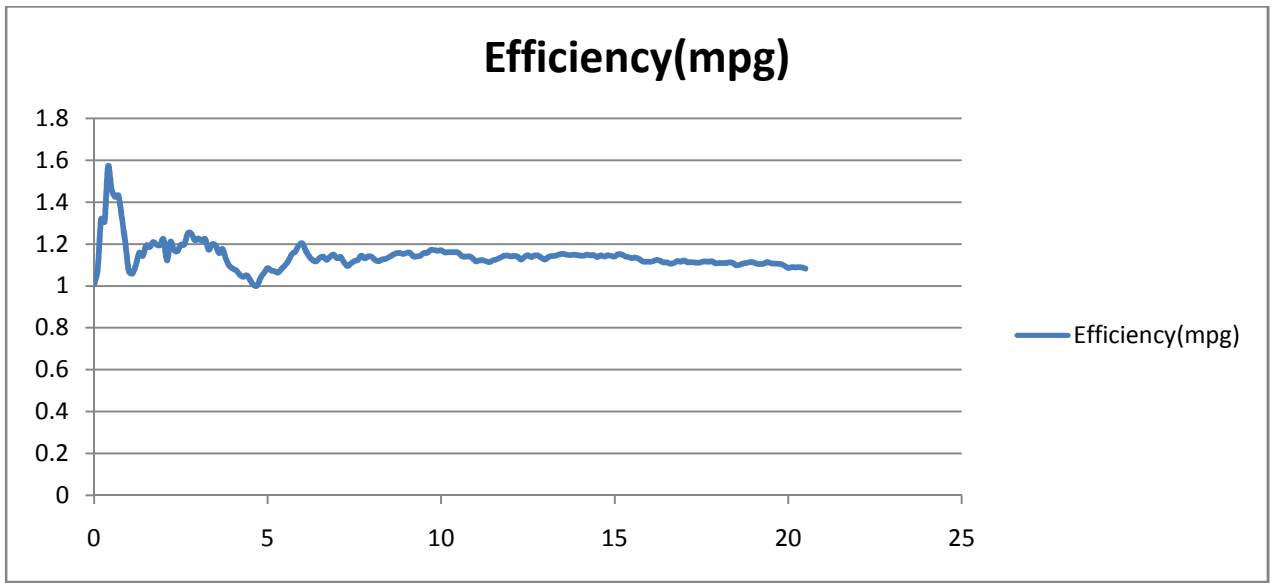
three real time tasks that handle the operation of the program, all which loop infinitely and have a period of 100ms. All three tasks share information using two circular buffers, one which holds the raw data, and the other holds the processed data. The first task is given 50ms to run, and it receives the data from the other board. The second task discards the data that isn't necessary and performs simple math on the important data to make a new value, i.e. computing the instantaneous fuel efficiency, and it is given 20ms to run. The third task is given the remaining 30ms to run and stores the data in a CSV format so that it can easily be imported to a spreadsheet program, like Microsoft Excel. The process opens and closes the file each time a line is written to ensure that data is not lost in the event of a program crash. This increases the time the process needs to run, but it drastically improves the reliability of the program.

Results

After figuring out the timing constraints, the project worked effectively. The first task on the reception board is allocated more time that is necessary, but that is to allow the possibility of adding more data boards in the future. I had originally wanted to collect data at every 10ms instead of 100ms, but 3ms was not enough time to open the CSV value, write to it, and close it again. 10 data points per second is however more than plenty for proper analysis. The three figures below show a typical output from the program which was run for about 20 seconds. The first figure, the table, shows the first 10 lines of the CSV file. The next two figures shows graphs made after being imported into Microsoft Excel, the first graphs showing the 4 values taken from the raw stream, and the next graph showing the mpg calculation performed by the telemetry software.

Fuel_Consumption	Distance	Temperature	Speed	Efficiency(mpg)
0.517239	0.521587	0.073563	0.780904	1.008406
1.280071	1.379125	0.359979	1.458818	1.077381
1.530601	2.019994	0.792069	2.30226	1.319739
1.849083	2.415457	1.23083	3.142329	1.3063
2.043142	3.208514	1.59886	3.719089	1.570383
2.320887	3.397773	2.100528	4.162866	1.463998
2.412992	3.43932	2.716074	4.650434	1.425334
2.459638	3.521778	2.860319	5.490138	1.431827
3.16678	4.1925	3.186431	5.911701	1.3239





Conclusions

The method is reasonable if the proper hardware and software architecture is already implemented on the boards generating and creating data. The real time constrains are not necessary to make the Ethernet communications reliable, it only serves to produce a predictable time between readings. If simpler hardware and software is implemented, a CAN network would likely be implemented, in which case a real time constrains may be necessary to improve reliability. The software is only capable as is to communicate between 2 boards, but the code allows for more to be added relatively easily.