Instrumentation

Beau Trepp

Electrical, Electronic and Computer Engineering

University Of Western Australia

A thesis submitted for the degree of

Bachelor of Computer Science/ Bachelor of Electronic Engineering

2011 November

Abstract

Put your abstract or summary here, if your university requires it.

Acknowledgements

During work on this project, I recieved

TODO thank

ZeroMQ guys, Gumstix Guys

Ian Hooper Jonathan something?. Thomas

REV Team

Contents

List of Figures **v**

List of Tables **vii**

Glossary **ix**

1 Introduction **1**

1.1 Electric Vehicles 1

1.1.1 Pollution 1

1.1.2 Rising Fuel Prices 2

1.2 Embedded Systems 2

1.2.1 History 2

1.2.2 Telemetry 2

1.2.3 Modularity 2

2 Aims of the project **3**

2.1 Final Aim 3

2.2 Preliminary aims 3

3 Literature Review **5**

4 System Design **7**

4.1 GPS 7

4.1.1 Hardware 7

4.1.2 Drivers 7

4.1.3 Design 9

5 Windowing Toolkit **11**

5.1 Pages 11

5.2 Redraw Performance 11

5.2.1 Refresh on Arrival 11

5.2.1.1 Message speed greater than redraw rate 13

5.2.2 Add to Queue 13

5.2.3 Redrawing the Screen 13

5.2.4 Redraw rate 14

5.2.5 Batch Redraw 15

5.2.5.1 Maximum batch size 15

5.2.5.2 Small batch size 15

6 Interface **17**

6.1 Overview Panel 17

6.2 Battery 18

6.3 Maps 18

6.4 Trip Meter 18

6.5 Trip Meter 18

6.6 Trip Meter 18

6.7 About 18

7 Discussion **19**

8 Materials & methods **21**

References **23**

List of Figures

5.1 Processing message flow chart 12
5.2 Appending to Queue 14

6.1 Panel showing other panels 17

List of Tables

GLOSSARY

Glossary

BMS	Battery Management System
CPU	Central Processing Unit
GPS	Global Positioning System
TCP	Transmission Control Protocol
ZMQ	Zero MQ

GLOSSARY

1

Introduction

1.1 Electric Vehicles

There are many motivating factors behind the development of electric cars. These vehicles utilze new technologies and are represent humanity moving forward in both imagination and respect for the environment.

1.1.1 Pollution

Electric vehicles are advantageous over traditional ICE vehicles as they operate with zero emissions. These vehicles have no exhaust, so therefore have no emissions. While this does not make them completely pollutant free, it does help limit and control the emissions being produced by the act of transport. It is important to remember when discussing electric vehicles that the components must be manufactured using industrial processes and the act of generation electricty. This does not making them truly carbon neutral, but helps limit the sources of pollution. It is much more easier to manage the pollution produced from one power plant, than that from thousands upon millions of vehicles. (1)

1.1.2 Rising Fuel Prices

1.2 Embedded Systems

1.2.1 History

1.2.2 Telemetry

1.2.3 Modularity

2

Aims of the project

2.1 Final Aim

The Ultimate goal of the project is to investigate the viability of distrubted systems in a automotive environment. This will culminate into a completed system, provided data logging functionality and a user interface to view the live data. (2)

2.2 Preliminary aims

Preliminary aims of the project are to.

a minimal embedded systems

2. Develop GPS capability

3. Develop BMS capabilty

this data into a user display

his data to be reviewed later

3

Literature Review

4

System Design

4.1 GPS

4.1.1 Hardware

A vital part of the data logging and user-interface of the software is finding the cars current location. This is done by the use of a off-the self GPS unit. Currently the system is using a Qstar MODEL NEEDED [CITATION NEEDED] usb equipped GPS receiver. This receiver operates at a rate of 10hz [CITATION NEEDED], though it can be set to operate at a slower frequency of 1hz. For the purposes of recording positional data, along with estimating the vehicles current speed, the unit should run as fast as possible. The extra precision is useful for the data-logging aspect, with no negative effects on the user-display aspect.

The GPS device can be enumerated as a standard serial port. This is beneficial as it can be used on any device that has the correct drivers and a available usb port. As it appears as a normal serial port, it can be queried using standard system routines. This allows the program that reads the device to operate any custom knowledge of the device it is connected to, aside from the serial parameters to make the connection.

4.1.2 Drivers

While the Eyebot M6 has hardware usb support, it was not immediately compatible with the GPS sensor. Various versions of usb-serial drivers where tried (see figure X.X) each with their own problems. The main cause of this difficultly was the out-dated Linux kernel being run in the system. This was kernel version number 2.6.17 and was released in 2006, which is 5 years old as of writing [OSNEWS citation]. This was a major cause of incompatibilities, as the GPS receiver was manufactured a significant time after this kernel was written. The drivers had no clue as to what the usb product keys were, nor the specific quirks that the devices may have had.

The first driver attempted was the generic usb-serial driver, included as a kernel module in 2.6.17. This drivers success would mean that the sensor and program could be easily installed in just about any machine running Linux. The driver would have matured after the 2.6.17 kernel, and newer kernels would have support by default. This is beneficial to the system as it would require the least amount of configuration and setup if the GPS program was set to run in a different machine.

Sadly this driver did not perform correctly with the GPS device. While the driver was able to be loaded into the kernel without any errors, it caused problems when trying to associate with the GPS. The device appeared to use bulk endpoints [CITE/EXPLAIN], which were unsupported by the generic driver. This caused strange symptoms in the operating system. The main symptom of an incorrect driver was the generation of the /dev/ttyUSB0 device. This availability of this device implies that a tty is available to read/write from. Due to the incompatibility of the driver, this serial port would never report any bytes to be read, which is why it is unsuitable for use with this device. Customizing the generic driver to support this device would be unfeasible because it is unlikely that newer versions of this driver would support the device. This leads to the situation that if the GPS program is ported to a different machine, a custom version of the generic usb serial drivers would have to be ported as well.

As the GPS device did not work with the generic serial drivers, alternative drivers were investigated in order to support this device. Experiments indicated that this device was automatically detected and loaded in a newer kernel. This was kernel 2.X.X running on a x86 Intel machine. This functioned correctly and was able to communicate with the GPS device at the full rate of 10hz. The driver used by this kernel was called cdc-acm. Further investigation showed that this driver could be included as a kernel module for the gumstix platform.

This driver was not immediately compatible with the device. This was because the device was manufactured after the kernel [CITE ME]. As such the driver did not recognize the manufacturer ID and product ID of the GPS[see figure X.X]. The driver was then modified to include this information and re-deployed to the eye-bot. This was successful in creating the virtual serial device inside /dev, and also in allowing data to be read from this device.

While the cdc-acm driver was able to be loaded and functioned, it still contained errors. If the system was under intense CPU load, the program may not run quick enough to remove all the data from the serial port buffer.[CITE TTY/SERIAL BUFFERS]. This would cause the operating system to throttle the port. Examination of the driver source code reveals that new information is dropped while this driver is throttled[cdc-acm/2.6.17-arm]. This is acceptable behaviour in this instance, however this driver had a race condition. If the TTY was throttled under certain conditions, it would be unable to un-throttle later on. This cause the TTY to drain its buffer and never accept any new data from the GPS even if its buffer was empty. [SOURCE CODE ANALYSIS SECTION HERE]. The driver was further modified to include spin-locks, a primitive kernel locking technique, in order to prevent this situation. The driver is now able to run for extended periods of time without locking up, enabling a reliable GPS reporting mechanism to be developed.

4.1.3 Design

Development of the GPS reporting component was done in C. This was chosen as it is a relatively low level language, with wide support. It is simpler to understand than more complicated object oriented style languages. This makes it a good choice for the GPS reporting mechanism, as it only has to do one task. In order to ensure that the code can be easily modified by future programmers, the structure of this program is simple. It runs in only one-thread, aside from the back ground ZeroMQ threads, and thus requires no concurrency management.

This first iteration of this code used blocking ports and read a single byte at a time. This was the style used to match existing examples [CITE PREVIOUS THESIS]. This was a functional design however it did lead to some problems. One problem with this approach was excessive throttling. The process would be woken up every time one character could be read, and would only remove one character from the buffer, even if there were hundreds waiting. This is a bad situation, as the process will continually be awoken to do trivial work. This steals cpu-time from another process, and caused the program to appear sluggish. This design was also in-sufficient as blocked the process while attempting to read from the port. This would cause the program to appear to hang if no data was available. It made it difficult to diagnose errors with this program. Figure X.X shows the process logic of the first iteration of the GPS controller.

The design was refined in order to support bulk-reads and non-blocking operation. This fixes the two problems with the first approach. Rather than reading one character at a time, the program now reads as many as possible and stores the information into its own circular buffer. This allows the serial port to be purged as quickly as possible. It also has the added feature of allowing the program to decide how to discard messages in the case that it cannot keep up with the GPS.

Table X.X shows the protocol for the GPS message when transmitted over the network. The protocol uses a binary format instead of an ASCII based one. This reduces the space/data transmitted over the link, which helps reduce cost and improve speed. The motivation for ASCII based protocols is that control characters can be used to help synchronize the data. As this design uses ZeroMQ in order to manage the flow of data, such control characters are unnecessary. All values are transmitter in network order, this is big-endian order so that the most significant byte is transmitted first.

5

Windowing Toolkit

5.1 Pages

5.2 Redraw Performance

Due to the limited resources of the system, care must be taken when attempting to update the screen. As as windowing system was already being developed, it was desirable to have this system absract these actions away from the programmer. By using the toolkit the programmer can trigger when an element should be draw, and specify the drawing code to draw the element. The absraction means that the developed code will never be called directly by the programmer, it is all taken care of by the underlying mechanisms.

5.2.1 Refresh on Arrival

Whenever a new message is recieved by the user interface, it would seem appropriate to process that message instantaneously. Figure 5.1 shows the flow of this methodology. A message is recieved, then the system processes the message, redraws the screen, and then starts over again. While this is a working solution, it does present problems in regards to performance and the overall usability of the system.

The performance penalty in such a design is not immediately obvious. Data that is recieved should be displayed instantanouelsy. Inspecting Figure 5.1 further does help highlight the problem that occurs in this situation. While the system is processing or displaying the message, it is unable to process anymore messages. This can be a

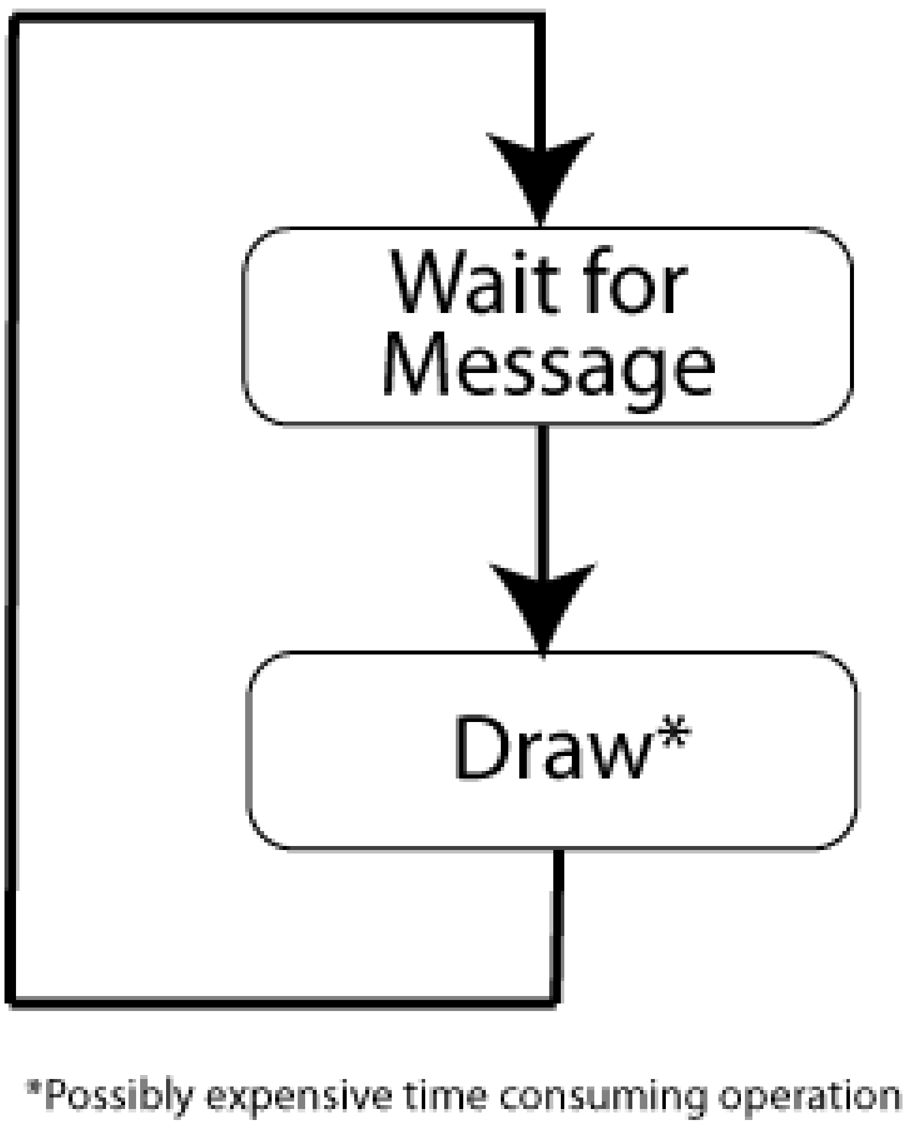

Figure 5.1: Processing message flow chart - This figure shows the naive approach to processing and displaying recieved message data

problem when the source of the messages is generating messages faster than the device can process.

5.2.1.1 Message speed greater than redraw rate

In situations such as the BMS module, the program was able to sufficiently cope with the input. This lead to all the messages being instantly removed from the network layer when they arrrived. Thus the program was always in the state of "waiting for a message". However another important sensor caused problems with this method. This sensor was the one designed to read the GPS signals. Messages containing GPS information where generated at a rate of 10 messages per second, or 10hz. This speed led to the situation where when a new message arrived, the process was still in the "Draw" state. This would cause the message to be delayed on the network layer.

The rate of input regarding the GPS module was constant, these messages would continually build up on the network layer. Any new messages that were transmitted would be dropped when they were attempted to be sent. While this is acceptable, eventually space would be cleared for new messages, it would lead to alot of new messages to be dropped. The other issue that occured here, was the new messages appeared at the back of the "Queue". Until all the older messages were processed, the new ones would not be seen. This is expected from how ZeroMQ functions (**?**)zeroMQ_internals. However, this meant that there was a significant delay until new data was seen.

The delay that occured was proportional to the size of the ZeroMQ Message Queue (**?**)zeroMQ_internals. This delay could be up to a couple of seconds, which is unacceptable for live feedback to the driver. Reducing the size of the Message Queue helped alleviate the problem, however the screen would still attempt to redraw as fast as the messages where recieved. This also led to the interface portion of the system hogging the CPU. Thus an alternate method of dealing with screen updates was developed.

5.2.2 Add to Queue

5.2.3 Redrawing the Screen

The previous section discussed the problems that occured with allowing the screen to update whenever a message was recieved. The biggest performance penalty that

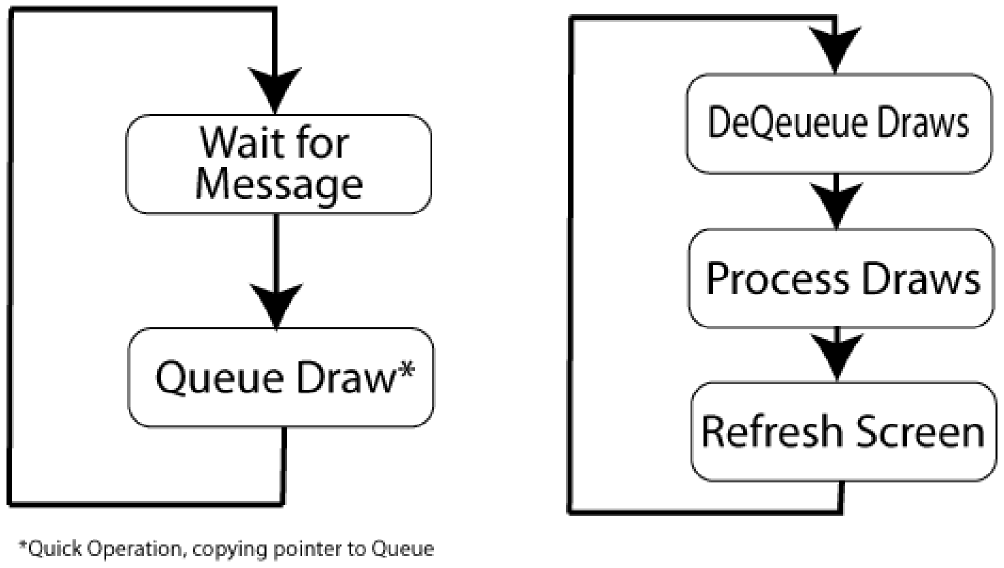

Figure 5.2: Appending to Queue - This figure shows the flow of appending screen refreshes to a queue

occured was not from the actual processing of the data, but from having to display it on the screen. All of the processing is relatively trivial computation wise. It is the transfer of variables into various memory locations so they can be accessed when the screen is displayed. Copying a variable itself is trivial, however processing that variable for display on screen is incredibly intense.

5.2.4 Redraw rate

Inspection of the EyeLin library source code showed that the library was using a simple framebuffer in order to interact with the screen (**?**)frame_buffer. This framebuffer method stored the entire screen state as an 24bit image. By default, whenever one pixel was changed in this image, all the data was copied to the framebuffer again. This was incredibly wasteful, and contributed to the large delays in redrawing the screen. Many items on the screen need to be redrawn together, for instance, a digit display has three or more digits that may change from it's last appearance. By default the library would redraw three times if the number changed by a large amount. This caused alot of performance issues in previous projects using these libraries see

5.2.5 Batch Redraw

In order to offset this problem the way in which the queue processed draws was modified. The thread that processed the queue would attempting to dequeue as much items as possible and process them as a batch. This gives more performance than attempting to redraw the screen for every change.

5.2.5.1 Maximum batch size

Attempting to remove as many items as possible is a problem in a multi-threaded system. Consider the case where a thread is adding elements, the producer thread, at the same time the drawing thread is removing them, the consumer thread. If the producer thread is operating faster than the consumer thread, the consumer thread will always be removing elements from the queue. Thus the consumer thread has a maximum amount of elements it will process at a time. This garuntees that the consumer thread will always refresh the screen.

5.2.5.2 Small batch size

Another issue that can occur with processing drawing events in a batch format is that no new draw events may be generated. Consider that the thread is attempting to remove a certain number of redraw events, however there may only be half present inside the queue. If , for whatever reason, no more redraw events are added, the queue will wait forever attempting to remove them. This is undesirable, as some transitions may only trigger a few redraw events and do nothing more. A good example of this is a static page, like the sponsor page. It only has a few elements, the buttons and the sponsor logos, and is not dynamically updated in response to any data. If the thread was waiting for more draw events, they would never be recieved. The solution to this problem is to continue on with the drawing actions if the queue is ever empty. This prevents the thread for waiting for more events, and helps garuntee the constant refreshing of the screen.

6

Interface

6.1 Overview Panel

To facilitate easy navigation a panel was designed that shows all the different panels available. This panel is shown in figure 6.1. It contains 8 different aspects of the program, with room available for 12. This will allow the system to be extended in the future.

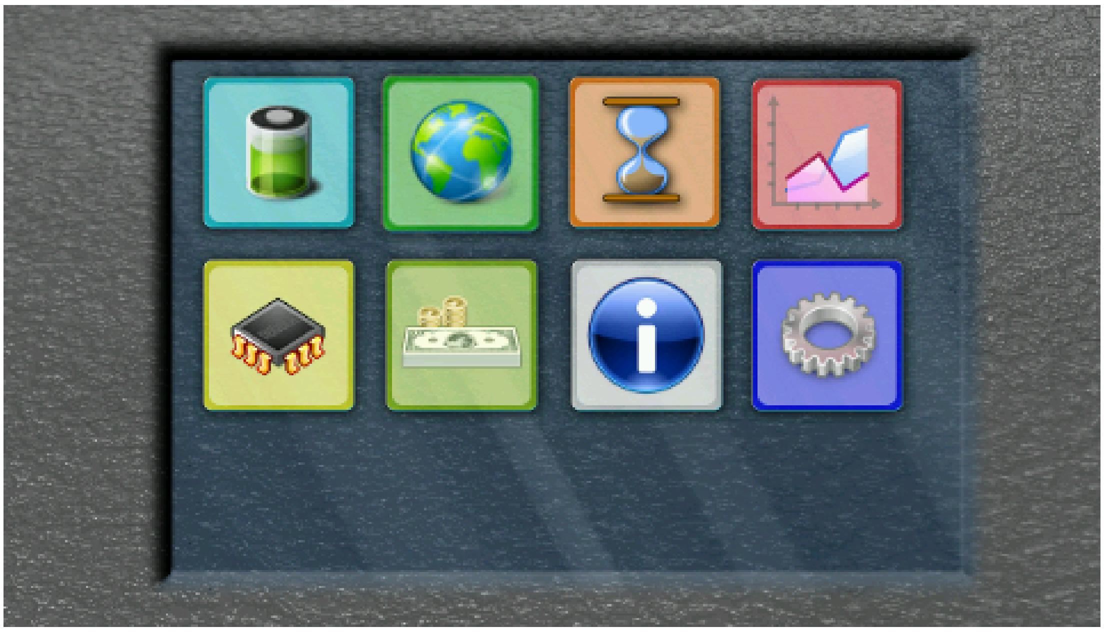

Figure 6.1: Panel showing other panels - Shortcuts to different aspects of the program. (Battery, Maps, Trip Meter, Accelerometer, Arduino, Savings, About, Options)

6.2 Battery

6.3 Maps

6.4 Trip Meter

6.5 Trip Meter

6.6 Trip Meter

6.7 About

7

Discussion

8

Materials & methods

[2] Kathleen Potosnak. **Modular implementation benefits developers, users. (separating user interface from rest of computer program)**. *IEEE Software*, **6**(3):91+, 1989. 3

References

[1] Thom Holwerda. **Linux 2.6.17 Released**, June 2006. 1

Declaration

I herewith declare that I have produced this paper without the prohibited assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in identical or similar form to any other German or foreign examination board.

The thesis work was conducted from XXX to YYY under the supervision of PI at ZZZ.

CITY,