Owen Popplestone

Professor Peter Walsh

CSCI 310 Intro to Graphical User Interfaces – Directed Studies

January 12, 2013

Engine Instrumentation Monitoring via OpenCPN's Plugin API and WXWidgets

Abstract

There is a need to extend the functionality of laptop based charting software to manage other systems on a boat.

OpenCPN[1] is an open source chart plotter navigating system that can be extended with a plugin API. The purpose of the project is to extend the functionality of OpenCPN to provide engine and systems data to a screen for the user.

Many different approaches to instrument data acquisition were considered. Sensors were hooked up to an Arduino[2], a Raspberry Pi[3], and a Dataq Instruments DI-155 Data Acquisition Starter Kit[4]. The decision was made to use the DI-155 to gather the data, and the Raspberry Pi as the device behind the scenes using soft real time programming to supply data to the OpenCPN program.

1. Introduction

The goal of this project is to read data from sensors in the engine compartment and view derived information on the laptop. The author and Professor Walsh discussed microprocessor boards such as the Arduino, the Motorolla HC11, and the Raspberry Pi. The decision was made to work collaboratively on this project with Professor Walsh doing the real time back end, (forming and sending the data) and the author doing the graphical front end (read, manipulate, and display). The collection of data is completely independent of the GUI. This allows for future development of trend graphs with the ability to log all readings to a database external to OpenCPN. The functionality of the project is similar to the OpenCPN's built in “Dashboard” plugin. Although the idea was there, the dashboard did not have any of the engine systems measurements that were desired. Since engine systems don't change much, the dynamic functionality of the dashboard was not followed.

The project requirements were to:

1. Use OpenCPN on linux. That requirement led to the use of the WXWidgets GUI platform.

2. Use a Dataq Instruments DI-155 data acquisition box to take measurements from the engine.

3. Use the perl module "AnyEvent" [5] on the Raspberry Pi to form and transmit the data.

The Raspberry Pi would then forward data on to OpenCPN running on another laptop via the wireless network.

Section 2 gives an overview of the whole project and how the multiple systems fit together. Section 3 details how to download the OpenCPN source code and compile the program and the Engine Instrumentation Plugin. Section 4 describes WXWidgets and how the decisions were made regarding what the GUI was going to look like and its behaviour. Section 5 concludes with how the project turned out and what the plans for the future are.

2. The Big Picture

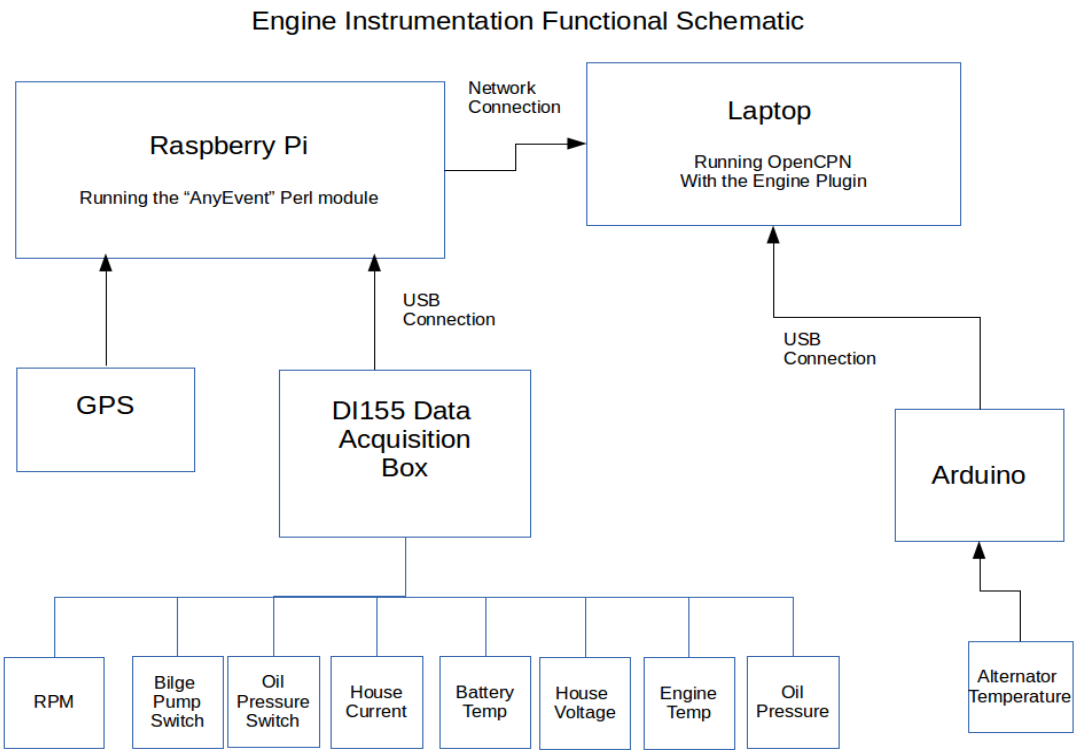

Figure 1 – The Overall System

The system (Figure 1) works by taking signals from the engine using a DI-155 box to capture the data. The box is capable of four analog inputs and four digital inputs. The analog inputs are capable of +- 2.5 to +- 50V full scale range at 13 bits resolution. Analog inputs are used for things such as oil pressure, voltage, and temperature. The four digital inputs could be used for things such as bilge and oil alarms. There are options to use two of the digital inputs as a counter for things such as RPM. The DI-155 box is connected to the Raspberry Pi using a usb cable.

The Raspberry Pi is where the brains of the data collection system are. The Pi is running linux and the "AnyEvent" event based programming tool. AnyEvent waits for data to come in from a sensor. AnyEvent then packages up and sends the data in a string to OpenCPN on UDP port 7070. The strings are styled after the National Marine Electronics Association (NMEA) 0183 Protocol[6]. As an example, the alternator temperature string is given below:

(Note: See appendix for full instrument sentence specification

```
        1   2 3          4          5
        |   | |          |          |
  $--ALT,x.x,C,hhmmss.ss,ddmmyy*hh<CR><LF>
```

Field Number:

1) Degrees
2) Unit of Measurement, Celsius
3) UTC Time – used to check whether the data is stale or not.
4) UTC Date
5) Checksum

The decision to add UTC time is not necessarily the NMEA standard for some instruments nor is it a requirement of all typical OpenCPN Dashboard instruments. The decision was made to enhance the reliability and trust in the data by synchronizing between the instruments and computers running OpenCPN and the Engine Instrumentation Plugin. Instrument time comes from the user supplied GPS

interfaced with the system.

3. OpenCPN and Build

OpenCPN is an open source cross platform chart plotting program and instrument navigation system that is from programmers all over the world. The author has successfully used the program for one year of cruising.

To build OpenCPN on a linux box from source:

1. Gather all dependant packages with apt-get. On Ubuntu use sudo apt-get. Debian is su root then apt-get.
2. sudo apt-get install libgtk2.0-dev gettext git-core cmake gpsd gpsd-clients libgps-dev build-essential wx-common libwxgtk2.8-dev libglu1-mesa-dev libgtk2.0-dev wx2.8-headers libbz2-dev libtinyxml-dev libportaudio2 portaudio19-dev
3. Clone the OpenCPN project from Github:
   git clone git://github.com/OpenCPN/OpenCPN.git
4. cd opencpn - *unless already in this directory.*
5. *cd plugins. - Download and extract the engine_pi source code (this project) and then place in this directory.*
6. *cd ..*
7. mkdir build
8. cd build
9. cmake ../
10. make
11. sudo make install

The instructions can also be found at:

http://opencpn.org/ocpn/compiling_source_linux

The user can then launch OpenCPN. At start up, or by clicking the wrench icon the user can select the plugin manager. Click enable to have the Engine Instrumentation plugin show on the toolbar. When using OpenCPN just click the Engine icon to have the data displayed. There are some choices as to where the data comes from. In the main settings page of OpenCPN the user can choose where the data stream is coming from. This project receives the data on UDP port 7070.

As OpenCPN is open source, there were times when the author had to debug other people's code as well. The project was downloaded as source from it's git repository a number of times when there were compile problems to start with a clean build. (github.com/OpenCPN/OpenCPN.git) Using the latest source code introduced new problems from other people working on the project. This made for some challenges when working on the Engine Instrumentation GUI as other code had to be debugged as well. Future development work will be from a compiled release instead.

4. WXWidgets / GTK graphics libraries

OpenCPN uses the WXWidgets GUI framework. WXWidgets is open source and cross platform. It will run on Microsoft Windows, Linux, and MAC. It is a wrapper for each OS's native GUI API. OpenCPN uses WXWidgets 2.8, and users have to be careful to realize this. Under Linux WXWidgets is the wrapper for GTK+ using the native API wherever possible and adding to it if needed.[7,8]

The author and Professor Walsh conducted a needs analysis of the GUI widgets. The requirements for the project were to have a widget with a dial and an “Alarm Light” style widget for the bilge pump and low oil pressure alarm. In the future, a digital display will be added and when the database storage is implemented, a graph trend display widget will be built.

4.1 Prototype Programming With GTK

To prove that these concepts would work the first prototype back end was to build a NMEA0183 sentence on the Arduino to pass strings to OpenCPN. That worked using the rudder angle sensor in the existing dashboard. The next step was to do a prototype on the Raspberry Pi using GTK with an Arduino feeding NMEA0183 strings. The simple prototype was a success, feeding data to a text box. The author chose GTK as the prototype because of WXWidgets using GTK in the linux environment.

4.2 Alarm panel widget:

Some research was done into how OpenCPN widgets handled alarm situations. The closest widget that was found was the light indicating GPS status. The light turned green for good GPS data and red for bad GPS data. This methodology did not work for the bilge pump and oil pressure alarms. Users are accustomed to alarms similar to the alarms in their car. The alarm light is muted when the conditions are normal. Therefore the alarm light does not draw any attention to itself as it is the basic grey colour. When an alarm condition exists, the light turns red. When the data is stale the alarm turns black as per the dial widget specification. There are no sounds to indicate the alarm signals at this time, only the visual indicators.

4.3 Dial widget:

The dial widget was utilized from the built-in “Dashboard Plugin” with some minor changes.

The arrow should stay inside fixed limits so if the data is lower than the minimum dial value it is set at the minimum dial value. If the value is higher than the maximum dial value it is set at the maximum dial value. If the data is at a warning level, the needle will point to a yellow area on the dial. If the data is at an alarm level it points to a red area. If the data is older than 15 seconds or if the GPS time stamp is off by plus or minus 15 seconds the needle will change from orange to black and will stay that way until the data is not stale. This last specification does not match with OpenCPN's style of making the

needle disappear when data is “lost”. As the Engine Instrumentation Plugin is being more strict with timing of data, the decision was made that rather than make the needle disappear, the needle will turn from the normal orange colour to black. This allows for a user to see that they might not have time set up properly. The user will still see movement in data, but will notice that the data is showing as stale. The user can then troubleshoot their system to find out why data is coming in indicated as stale.

4.4 Future Trend Widget

The trend widget has not been implemented (the Raspberry Pi is not yet logging data) but will be defined here to follow similar behaviour as the other widgets. This was defined early on as the next item to be implemented. The trend widget will take on the format of popular paper strip chart styles. Like the Dial widget, values exceeding the maximum and minimum of the trend will get changed to indicate the maximum and minimum trend values. A yellow line will indicate a warning level. A red line will indicate an alarm level. The data line on the chart will be the same as the dial orange needle colour. If data is stale (as specified above) then that section of the chart line will be indicated in black. The background will be in white. The user will be able to select the time frame of the trend (2 min, 5 min, 10 min, 30 min, 1 hr, 6 hrs, 12 hrs, 24hrs).

4.5 Future Digital Widget

For older eyes or if the laptop is run from the navigation station down below, the dial can be a little hard to read. The digital widget will provide an easy to read, large black font on light grey background. If the data gets to a predetermined warning level, the background will turn yellow. If the data gets to an alarm level, the background will turn red. If the data goes stale (as specified above), the background will turn black

4.6 Engine Instrumentation Plugin

The main GUI Engine Instrumentation Plugin (Figure 2) of this project was to model after similar plugins for OpenCPN. Parameters such as layout, consistent and correct use of colour, and data validity were considered and planned for.[9] After analyzing a number of the widgets and plugins, it was discovered that many do not follow design guidelines. The guidelines were left mostly to the programmer's discretion. The only thing that was fairly well defined was the Plugin API itself, not what the GUI should look like. As an example, some plugins close on the x button on the top of the screen. Others use the plugin toolbar icon to shut off. The decision was made that most users would expect the close button functionality, so users of Engine Management Plugin are given both options to close the plugin.

Moving from the initial GTK prototypes to the OpenCPN WXWidgets plugin API was a fairly steep learning curve for the author. After working with demo example of the plugin API for some time it was discovered that the demo was not supported any more. There was significant commented out code and comments such as "the following commented out line causes a core dump”. As well, compilation was being done at the plugin directory itself, rather that from the top of the OpenCPN directory tree. Others were compiling from the top down rather that just the plugin. This led to errors in other plugins' makefiles when compiling in the sub-directory. All compilation by the author is now done from the top of the tree until other's cmakefiles are corrected.

Figure 2 The Engine Instrumentation Plugin

5. Conclusion and Future Work

In conclusion, the goals that were set at the start of this project have been met. In some ways it would have been much easier to program the GUI outside of the OpenCPN/WXWidgets framework and just stick with something like GTK. There are however some aspects of OpenCPN which made this project easier as well. The NMEA 0183 code was well developed and was easy to use and add to. OpenCPN has a great plugin that allows recording and replay of data, and the ability to read from the serial port and the network as well. This made prototyping very easy. Dial instruments were mostly recycled from another plugin with modifications to suit the Engine Instrumentation Plugin.

The open source OpenCPN program had its issues however. When the main code repository's git master branch was downloaded, errors were introduced that others had let slip through. This led to the debugging of other code in the program.

Future goals would be to look into making a generic NMEA instrument string so that a user could hook any instrument up to OpenCPN. Configuring the string output of the instrument would generate an instrument if the user desired. The plugin should also have the ability to shrink and grow depending on how many instruments the user would like if generic instruments were available. On the back end side, growth of the project would see data stored in a database for later use or for trend graphs. Although this project is not on the bleeding edge of technology, there is a trend with pleasure boat users to have more access to data like this.[10] This project makes low cost instrumentation within the reach of the average boater. The DI-155/ Raspberry Pi system costs approx $200 US whereas a system such as Actisense's EMU-1 is closer to $500 US[11]. With these results the author has been happy with the outcome of the project and believes it meets all of the stated initial needs.

References:

1. OpenCPN - Chart plotting software. www.opencpn.org

2. Arduino – An open source electronics prototyping platform. http://www.arduino.cc/

3. Raspberry Pi – A small single board computer running linux. http://www.raspberrypi.org/

4. Dataq Instruments DI-155 Data Acquisition Starter Kit

   http://www.dataq.com/products/startkit/di155.htm

5. AnyEvent - A perl framework for event based programming.

   http://search.cpan.org/~mlehmann/AnyEvent-7.07/

6. National Marine Electronics Association – NMEA0183 Standard

   http://www.nmea.org/content/nmea_standards/nmea_0183_v_410.asp

7. WXWidgets 2014 *Overview*. http://wxwidgets.org/about/ Accesed March 2014

8. GTK+ A multi-platform toolkit for creating graphical user interfaces. www.gtk.org

9.*Five Practical Elements Of Effective SCADA Graphics* By Dan Roessler and Lisa Garrison February 2013 http://www.pipelineandgasjournal.com/five-practical-elements-effective-scada-graphics Accessed March 2014

10. *Marine Mobile and Apps* by Terri Hodgson Canadian Yachting West March 2014

11. Actisense EMU-1 Engine Monitoring Unit

http://www.actisense.com/products/nmea-2000/emu1.html

Appendix:

Engine instrumetation sentences:

ERM - RevolutionsPerMinute

```
          1 2          3        4
          | |          |        |
  $--ERM,x,hhmmss.ss,ddmmyy*hh<CR><LF>
```

Field Number:

1) Revolutions

2) UTC Time

3) UTC Date

4) Checksum

ALT - Alternator Temperature

```
          1    2 3            4           5
          |    | |            |           |
```

$--ALT,x.x,C,hhmmss.ss,ddmmyy*hh<CR><LF>

Field Number:

1) Degrees

2) Unit of Measurement, Celcius

3) UTC Time

4) UTC Date

5) Checksum

ALC – Alternator Current

```
          1   2             3           4
          |   |             |           |
```

$--ALC,x.x,hhmmss.ss,ddmmyy*hh<CR><LF>

Field Number:

1) AMPS

2) UTC Time

3) UTC Date

4) Checksum

BTP – Battery Temperature

```
        1    2 3          4          5
        |    | |          |          |
$--BTP,x.x,C,hhmmss.ss,ddmmyy*hh<CR><LF>
```

Field Number:

1) Degrees

2) Unit of Measurement, Celcius

3) UTC Time

4) UTC Date

5) Checksum

RUC – House Run Current

```
        1   2            3          4
        |   |            |          |
$--RUC,x.x,hhmmss.ss,ddmmyy*hh<CR><LF>
```

Field Number:

1) AMPS

2) UTC Time

3) UTC Date

4) Checksum

ETP – Engine Temperature

```
          1    2  3            4           5
          |    |  |            |           |
$--ETP,x.x,C,hhmmss.ss,ddmmyy*hh<CR><LF>
```

Field Number:

1) Degrees

2) Unit of Measurement, Celcius

3) UTC Time

4) UTC Date

5) Checksum

OIP – Oil Pressure Sender

```
          1    2 3            4           5
          |    | |            |           |
$--OIP,x.x,C,hhmmss.ss,ddmmyy*hh<CR><LF>
```

Field Number:

1) Pressure

2) Unit of Measurement, PSI/ Kpa

3) UTC Time

4) UTC Date

5) Checksum

BIP – Bilge Pump Switch

```
        1 2          3        4
        | |          |        |
$--BIP,x,hhmmss.ss,ddmmyy*hh<CR><LF>
```

Field Number:

1) Boolean 1 for on 0 for off

3) UTC Time

4) UTC Date

5) Checksum

OIS – Oil Pressure Switch

```
        1 2          3        4
        | |          |        |
$--BIP,x,hhmmss.ss,ddmmyy*hh<CR><LF>
```

Field Number:

1) Boolean 1 for on 0 for off

2) UTC Time

3) UTC Date

4) Checksum

HOV - House Battery Voltage

```
          1   2            3          4
          |   |            |          |
$--HOV,x.x,hhmmss.ss,ddmmyy*hh<CR><LF>
```

Field Number:

1) Voltage

2) UTC Time

3) UTC Date

4) Checksum