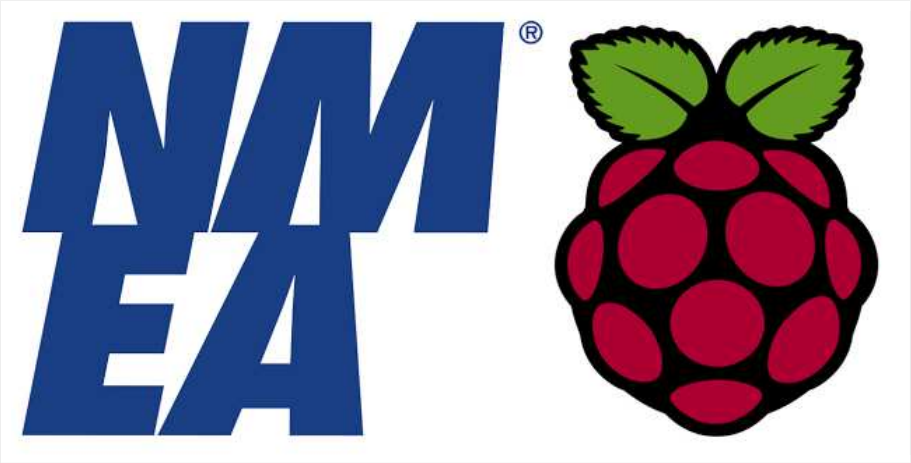
King Tide Sailing

Yachting for the Common Man.

Home | Raspberry Pi Wireless NMEA-0183 Multiplexer | NMEA-0183 and Arduino | NMEA-0183 and Raspberry Pi | Low-Cost Digital Compass | About King Tide Sailing

Tuesday, April 19, 2016

A Raspberry Pi Wireless NMEA-0183 Multiplexer (USB GPS, Digital Compass, and Sailboat Instrument Input)



Recommended Gear:

These are the exact devices that I ordered and am using. The big ticket items:

- Raspberry Pi 3 Model B (2016 version--you can use an older one, but you'll need a wifi dongle)
- GlobalSat BU-353-S4 USB GPS Receiver (but any USB GPS will work)
- Airmar DST800 Thru-Hull Smart Sensor Triducer (Depth, Temperature, and Water Speed Paddlewheel--but any regular NMEA sailboat instrument will work)
- MPU-9250 9-DOF Accelerometer, Gyro, and Tilt-Compensated Compass (only $12.50! many other IMUs will work though)

Smaller items that are necessary:

- NMEA-0183 to USB Converter (NMEA-0183 is really just RS-485)
- 12V to Micro-USB Power Supply (make sure you get the right cord orientation for your setup)
- Cables, Soldering Iron, and Solder for the MPU-9250
- Other random stuff

Step 1: Setup the Raspberry Pi

I'm running this with a brand new fresh install of Raspbian Jessie on April 18th, 2016. Before anything else, ensure you're software is all up to date by running these two commands from the terminal:

```
sudo apt-get update
sudo apt-get upgrade
```

Once this is done, open up Menu->Preferences->Raspberry Pi Configuration, and change your hostname or password if you wish. But the one thing we're looking for is under the Interfaces tab; click I2C Enabled. I also recommend configuring your Localisation tab to your location. Once you're done, click OK and reboot.

Next, we're going to begin the long process of installing all the necessary software (these are all for the MPU-9250 Digital Compass--if you're not using that, then you probably don't need these except you might need python-dev).

```
sudo apt-get install i2c-tools
sudo apt-get install cmake
sudo apt-get install python-dev
```

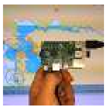
Blog Archive

- ▼ 2016 (7)
  - ▼ April (1)
    - A Raspberry Pi Wireless NMEA-0183 Multiplexer (USB...
  - ► February (5)
  - ► January (1)
- ► 2015 (10)

Popular Posts

How to Connect any NMEA-0183 Device to an Arduino

NMEA-0183 uses 12V signals to communicate with each other. The oldest instruments use RS-232, or a single wire that sends a high voltage...



How To Install OpenCPN on a Raspberry Pi

EDIT 11 FEB 2016: I would now recommend simply installing the OpenPlotter software found here (it's basically a fresh install ...

How to Setup the MPU-9150 9-Axis Accelerometer, Gyro, & Compass with an Arduino

The MPU-9150 9-Axis Accelerometer, Gyro, & Compass (hereafter referred to as just "The MPU") gives you a lot of info tha...

How to Setup the MPU-9250 on a Raspberry Pi

Step 1: Get the Right Gear You'll need to get the associated wires, jumper cables, solder, what-have you, in order to properly con...

How to Make an NMEA-0183 Tilt-Compensated Compass and Rate-Of-Turn

```
sudo apt-get install octave
```

When those are finished installing, connect your MPU-9250 (see this post for more information. In fact, you can connect all your devices--see this one for the USB GPS and this one for the DST800 or any other NMEA instrument using the USB to RS-485 converters I linked to above). After connecting, verify the Raspberry Pi can actually "see" the MPU-9250 by running:

```
sudo i2cdetect -y 1
```

and you should see a 68 somewhere in the grid. If not, well... you might have to go to RichardsTech's website to troubleshoot. Let's assume it is working. Great.

These next few steps are going to come fast, so keep up. We make a working directoy called 'kts' to store everything and to stay organized (keep in mind if you use my scripts, your directoy must also be called kts (or you can just change the scripts to reflect your name)). After making the directory, we download the RTIMULib from github and install its calibration program.

```
mkdir kts
cd kts
git clone https://github.com/richards-tech/RTIMULib2.git
cd RTIMULib2/Linux/RTIMULibCal
make -j4
sudo make install
```

This installs the calibration program, which is 100% necessary for the compass to work. Again, if you get lost, just mosey on over to his github page, or my earlier and expanded writeup on the MPU-92450 with the Raspberry Pi. However, I cannot reiterate this enough: **when you calibrate the compass, you MUST be working in the RTIMUEllipsoidFit folder--otherwise it WILL NOT WORK.** Not only that, but there must be a copy of the RTIMUEllipsoidFit folder in the directory **above** the script that is using the MPU-9250. If you follow my directions, this won't be a problem.

So let's do that.

```
cp -r /home/pi/kts/RTIMULib2/RTEllipsoidFit/ /home/pi/kts/
cd /home/pi/kts/RTEllipsoidFit/
RTIMULibCal
```

When you're on your boat, and your compass is installed where it will be used (I unfortunately cannot help you with this part), then run the calibration program RTIMULibCal. It doesn't make sense to calibrate it anywhere other than its intended final location. This creates a RTIMULib.ini file in the RTEllipsoidFit folder, which you then copy over to the scripts folder. Wait, the scripts folder? Yes, let's make that and download all the scripts used for this project from my github page:

```
cd ..

git clone https://github.com/OldCC/scripts.git

cd scripts
```

Now we have all the programs installed, we have all the folders made, and we have all the scripts.

Bonus Step: Setup a Wifi AP on the Raspberry Pi 3 Model B

This step took forever, mainly because there's a hundred tutorials out there on how to set up a wifi router with the Pi. That's not what I want to do. I just want a wireless access point to connect a bunch of things to, with no internet. I don't need the internet (yet). Fortunately, this is super super super easy to do, thanks to this awesome script that you just run once, edit one file, reboot, and you're good. Finally.

But that script actually didn't work for me, so I modified it slightly by making it a little less user friendly (sorry). I included this script in the scripts folder, if you downloaded it from github above. Set up the apsetup.sh by

```
sudo chmod +x apsetup.sh

sudo nano apsetup.sh
```

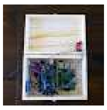
and change PASSWORD and NAME to whatever you want for your network. Then Control + C to save and exit. Next, run that script as the root user by

```
sudo ./apsetup.sh
```

This will take a minute or two to run through everything. When it's finished, we need to make it run at startup

Indicator

This simple, analogue compass costs $10. If we want to use this for electronic chart-plotting applications, then we need an electronic com...

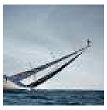


A Wireless Arduino NMEA-0183 Multiplexer

Let's put all this together. Using the previous posts with the "multiplexer" label, it's quite easy to set up an Ardui...

Bernard Moitessier's Last Sailboat Tamata

The Death of Bernard Moitessier's Joshua On the evening of Wednesday, December 8th, a determined couple stood on the sho...



Correcting NMEA-0183 Wind for Vessel Roll, Pitch, and Yaw

A simple wind anemometer only measures wind blowing orthogonal to its axis of rotation. If you take a normal anemometer and tilt it 90 ...

A Raspberry Pi Wireless NMEA-0183 Multiplexer (USB GPS, Digital Compass, and Sailboat Instrument Input)

Recommended Gear: These are the exact devices that I ordered and am using. The big ticket items: Raspberry Pi 3 Model B (2016 ver...

An Update on Calculating the Tidal Set and Drift

After a fresh round of ops testing, using the calibration for water speed here , I've found that the calculated VDR (Tidal Set and ...

Labels

Multiplexer (8) Arduino (7) NMEA-0183 (6) Raspberry Pi (6) DST-800 (2) OpenCPN (2) VDR Set and Drift (2) mpu-9150 (2) Articles (1) BU-353 USB GPS (1) Bernard Moitessier (1) Dampen (1) Degrees (1) MPU-9250 (1) Projects (1) Tamata (1) VHW (1) Waterspeed (1) compression post repair (1) diy (1) wind (1) all on its own. To do this, open this file:

```
sudo nano /etc/default/hostapd
```

and then turn this line here

```
#DAEMON_CONF=""
```

into this line (make note that I remove the the #)

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

Go ahead and reboot, and when it's back online, try to connect to your wifi network with your laptop or phone. If it connects, then you're good! But there won't be any internet. That's okay, because there will be NMEA-0183 data in just a little bit.

Step 2: Initialize the NMEA-0183 Instrument Scripts

Here is the monitor.py script, which runs at startup:

monitory.py

```
#!/usr/bin/env python

import serial
import operator
import time
import os
import sys
import socket
import select

# change the working directory to the scripts folder
os.chdir("home/pi/kts/scripts")

# log
hack = time.asctime( time.localtime(time.time()) )
log = open('log', 'w')
log.write("Monitor initialized: " + str(hack) + "\r\n")
log.close()

# determine GPS/DST ports
init = 0
while init == 0:

    # check port 0
    port0 = serial.Serial('/dev/ttyUSB0', 4800, timeout=3)
    port0_raw = port0.readline()
    hack = time.asctime( time.localtime(time.time()) )
    log = open('log', 'a')
    log.write("Checking serial port 0: " + str(hack) + "\r\n")
    log.close()
    if "*" in port0_raw:
        port0_split = port0_raw.split('*')
        port0_sentence = port0_split[0].strip('$')
        cs0 = port0_split[1][:-2]
        cs1 = format(reduce(operator.xor,map(ord,port0_sentence),0),'X')
        if len(cs1) == 1:
            cs1 = "0" + cs1

        # if it is a valid NMEA sentence
        if cs0 == cs1:
            port0_vars = port0_sentence.split(',')
            title = port0_vars[0]

            # if the GPS is connected to this port
            if title == "GPRMC":
                gps = "0"
                f = open('gps_port', 'w')
                f.write(gps)
                f.close()
                init = 1

            # if the DST is connected to this port
            if title == "SDDPT":
                gps = "1"
                f = open('gps_port', 'w')
                f.write(gps)
                f.close()
                init = 1
    port0.close()

    # check port 1
    port1 = serial.Serial('/dev/ttyUSB1', 4800, timeout=3)
    port1_raw = port1.readline()
    hack = time.asctime( time.localtime(time.time()) )
    log = open('log', 'a')
    log.write("Checking serial port 1: " + str(hack) + "\r\n")
    log.close()
    if "*" in port1_raw:
        port1_split = port1_raw.split('*')
        port1_sentence = port1_split[0].strip('$')
        cs0 = port1_split[1][:-2]
        cs1 = format(reduce(operator.xor,map(ord,port1_sentence),0),'X')
        if len(cs1) == 1:
            cs1 = "0" + cs1

        # if it is a valid NMEA sentence
        if cs0 == cs1:
            port1_vars = port1_sentence.split(',')
            title = port1_vars[0]

            # if the GPS is connected to this port
            if title == "GPRMC":
                gps = "1"
                f = open('gps_port', 'w')
                f.write(gps)
                f.close()
                init = 1

            # if the DST is connected to this port
            if title == "SDDPT":
                gps = "0"
                f = open('gps_port', 'w')
                f.write(gps)
                f.close()
                init = 1
    port1.close()

log = open('log', 'a')
log.write("GPS port is " + gps + ": " + str(hack) + "\r\n")
log.close()

# begin the instrument scripts
hack = time.asctime( time.localtime(time.time()) )
log = open('log', 'a')
log.write("Starting gps: " + str(hack) + "\r\n")
log.close()
os.system("python gps.py &")
hack = time.asctime( time.localtime(time.time()) )
log = open('log', 'a')
log.write("Starting imu: " + str(hack) + "\r\n")
log.close()
os.system("python imu.py &")
hack = time.asctime( time.localtime(time.time()) )
log = open('log', 'a')
log.write("Starting dst: " + str(hack) + "\r\n")
log.close()
os.system("python dst.py &")
hack = time.asctime( time.localtime(time.time()) )
log = open('log', 'a')
log.write("Starting kplex: " + str(hack) + "\r\n")
log.close()
os.system("sudo kplex &")

GPS_IP = "127.0.0.4"
GPS_PORT = 5005
gpssock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
gpssock.bind((GPS_IP, GPS_PORT))

IMU_IP = "127.0.0.5"
IMU_PORT = 5005
imusock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
imusock.bind((IMU_IP, IMU_PORT))

DST_IP = "127.0.0.6"
DST_PORT = 5005
dstsock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
dstsock.bind((DST_IP, DST_PORT))

hack = time.asctime( time.localtime(time.time()) )
log = open('log', 'a')
log.write("Starting loop: " + str(hack) + "\r\n" + "------------------------\r\n")
log.close()

gps_hack = time.time()
```

```
imu_hack = time.time()
dst_hack = time.time()
while True:

    # monitor gps.py
    gpsready = select.select([gpssock], [], [], .1)
    if gpsready [0]:
        data, addr = gpssock.recvfrom(1024)
        gps_hack = float(data)
    if time.time() - gps_hack > 10.0:
        hack = time.asctime( time.localtime(time.time()) )
        log = open('log', 'a')
        log.write("Restarting gps: " + str(hack) + "\r\n")
        log.close()
        os.system("pkill -9 -f gps.py")
        os.system("python gps.py &")
        gps_hack = time.time()


    # monitor imu.py
    imuready = select.select([imusock], [], [], .1)
    if imuready [0]:
        data, addr = imusock.recvfrom(1024)
        imu_hack = float(data)
    if time.time() - imu_hack > 10.0:
        hack = time.asctime( time.localtime(time.time()) )
        log = open('log', 'a')
        log.write("Restarting imu: " + str(hack) + "\r\n")
        log.close()
        os.system("pkill -9 -f imu.py")
        os.system("python imu.py &")
        imu_hack = time.time()

    # monitor dst.py
    dstready = select.select([dstsock], [], [], .1)
    if dstready [0]:
        data, addr = dstsock.recvfrom(1024)
        dst_hack = float(data)
    if time.time() - dst_hack > 10.0:
        hack = time.asctime( time.localtime(time.time()) )
        log = open('log', 'a')
        log.write("Restarting dst: " + str(hack) + "\r\n")
        log.close()
        os.system("pkill -9 -f dst.py")
        os.system("python dst.py &")
        dst_hack = time.time()
```

This script is quite useful, aside from starting the whole system. It first changes the working directory to the kts script folder, which is necessary for the imu script to access the right calibration data. Without this step, the MPU-9250 won't work correctly.

Throughout the monitor script, it writes down what it's doing with a timestamp to a log file to see what's happening with the script.

Next, it runs a quick routine to check which port number the USB GPS is plugged in to. I discovered that it seems to be quite random between port 0 and port 1, changing every time the Pi boots up. It writes the GPS port to a file, which is then read by the GPS and DST scripts. This also means that if you don't have a GPS plugged in and printing the GPRMC sentence, this whole system won't work. GPS is necessary.

After that, it executes the python scripts which then start running in the background (thanks to the "&" symbol), and then sets up the UDP ports to listen to each script's health monitor.

Every half second, each instrument script sends a timestamp which is read by this monitor script. If more than 10 seconds have gone by without receiving an update from the script (which means the script has failed for whatever reason), it will kill the process and restart it. It logs each action in the log file, which is helpful if I want to see if something is failing a lot.

Step 3: The GlobalSat BU-353-S4 USB GPS Receiver Python Script

Like I said, if there is no USB GPS plugged in, the monitor script won't work. But once it does, here's the GPS script:

gps.py

```
import sys
import serial
import math
import operator
import time
import socket
import os
```

```
GPS_IP = "127.0.0.1"
GPS_PORT = 5005

MON_IP = "127.0.0.4"
MON_PORT = 5005

mon = 0
log = time.time()

# determine gps port
f = open('gps_port', 'r')
gps_port = f.readline()
f.close()
ser = serial.Serial('/dev/ttyUSB' + gps_port, 4800, timeout=5)

while True:

    # health monitor
    hack = time.time()
    if hack - mon > .5:
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        sock.sendto(str(hack), (MON_IP, MON_PORT))
        mon = hack

    # log raw input
    gps_raw = ser.readline()
    f = open('gpsraw', 'a')
    f.write(gps_raw)
    f.close()
    if hack - log > 60:
        os.remove('gpsraw')
        f = open('gpsraw', 'w')
        f.write("Serial port " + gps_port + ": " + time.asctime( time.localtime(ti
        f.close()
        log = hack

    # checking to see if it's a valid NMEA sentence
    if "*" in gps_raw:
        gps_split = gps_raw.split('*')
        gps_sentence = gps_split[0].strip('$')
        cs0 = gps_split[1][:-2]
        cs1 = format(reduce(operator.xor,map(ord,gps_sentence),0),'X')
        if len(cs1) == 1:
            cs1 = "0" + cs1

        # if it is a valid NMEA sentence
        if cs0 == cs1:
            gps_vars = gps_sentence.split(',')
            title = gps_vars[0]

            # recommended minimum navigation sentence
            if title == "GPRMC":

                # heading from IMU
                try:
                    f = open('imu_bus', 'r')
                    line = f.readline()
                    f.close()
                    imu_split = line.split(',')
                    imu_hack = float(imu_split[0])
                    heading = float(imu_split[1])
                except ValueError:
                    f.close()
                    time.sleep(.03)
                    f = open('imu_bus', 'r')
                    line = f.readline()
                    f.close()
                    imu_split = line.split(',')
                    imu_hack = float(imu_split[0])
                    heading = float(imu_split[1])

                # if heading is from the last 3 seconds, and groundspeed less than
                # reset course to heading to eliminate low speed artifacts
                valid = gps_vars[2]
                course = float(gps_vars[8])
                groundspeed = float(gps_vars[7])
                if time.time() - imu_hack < 3.0 and groundspeed < 0.1:
                    course = heading

                # assemble the sentence with corrected course
                rmc = "GPRMC," + gps_vars[1] + ',' + gps_vars[2] + ',' + gps_vars[
                rmccs = format(reduce(operator.xor,map(ord,rmc),0),'X')
                if len(rmccs) == 1:
                        rmccs = "0" + rmccs
                gprmc = "$" + rmc + "*" + rmccs + "\r\n"

                # to gps bus
                f = open('gps_bus', 'w')
                f.write(str(time.time()) + ',' + valid + ',' + str(course)  + ',' +
                f.close()

                # to kplex
                sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
                sock.sendto(gprmc, (GPS_IP, GPS_PORT))
```

First, it determines which serial port the GPS is plugged in to. After that, it starts looping, and it sends the health monitor signal out every .5 seconds to the monitor.py script. It also logs the raw input from the GPS for the last 60 seconds (any and all input, regardless if it's a valid NMEA sentence). Next, it runs a quick routine to verify it really is a valid NMEA sentence, and only if it's the RMC one.

If it is, then it extracts the heading information from IMU (see next section), and if the groundspeed is less than .1 knots and heading is recent (from the last 3 seconds), then it replaces course with heading. Why? I've found that sitting at dock, even with a groundspeed of 0, the course still moves all over the place. This isn't really necessary, but it's slightly annoying since I'm a perfectionist and if I'm not moving, then the course should, by default, be the same as the boat's heading.

Then it assembles the whole sentence, writes it to the GPS bus file for the DST script, and prints it out to kplex via a UDP port.

Step 4: The MPU-9250 Tilt-Compensated Compass Python Script

You can read all about it in my previous post, but the big take away is that you *must calibrate it*. Otherwise it will be useless. Remember, run the calibration command *in* the RTIMUEllipsoidFit folder. This will produce a RTIMULib.ini file. Copy that file over to the kts scripts folder, so it's in the same directory as the imu.py script.

Here's the python script for the IMU:

imu.py

```
import sys, getopt

sys.path.append('.')
import RTIMU
import os.path
import time
import math
import operator
import socket
import os


IMU_IP = "127.0.0.2"
IMU_PORT = 5005

MON_IP = "127.0.0.5"
MON_PORT = 5005

SETTINGS_FILE = "RTIMULib"

s = RTIMU.Settings(SETTINGS_FILE)
imu = RTIMU.RTIMU(s)

# timers
t_print = time.time()
t_damp = time.time()
t_fail = time.time()
t_fail_timer = 0.0
t_shutdown = 0

if (not imu.IMUInit()):
    hack = time.time()
    imu_sentence = "$IIXDR,IMU_FAILED_TO_INITIALIZE*7C"
    if (hack - t_print) > 1.0:
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        sock.sendto(imu_sentence, (IMU_IP, IMU_PORT))
        t_print = hack
        t_shutdown += 1
        if t_shutdown > 9:
            sys.exit(1)

imu.setSlerpPower(0.02)
imu.setGyroEnable(True)
imu.setAccelEnable(True)
imu.setCompassEnable(True)

poll_interval = imu.IMUGetPollInterval()

# data variables
roll = 0.0
pitch = 0.0
yaw = 0.0
heading = 0.0
```

```
rollrate = 0.0
pitchrate = 0.0
yawrate = 0.0
magnetic_deviation = -13.73


# dampening variables
t_one = 0
t_three = 0
roll_total = 0.0
roll_run = [0] * 10
heading_cos_total = 0.0
heading_sin_total = 0.0
heading_cos_run = [0] * 30
heading_sin_run = [0] * 30

# sentences produces by the imu
iihdt0 = "$IIHDT,,T*0C"
iixdr0 = "$IIXDR,A,,D,ROLL,A,,D,PTCH,A,,D,RLLR,A,,D,PTCR,A,,D,YAWR*51"
iihdt = iihdt0
iixdr = iixdr0
freq = 1

while True:

  hack = time.time()

  # if it's been longer than 5 seconds since last print
  if (hack - t_damp) > 5.0:

      if (hack - t_fail) > 1.0:
          t_one = 0
          t_three = 0
          roll_total = 0.0
          roll_run = [0] * 10
          heading_cos_total = 0.0
          heading_sin_total = 0.0
          heading_cos_run = [0] * 30
          heading_sin_run = [0] * 30
          t_fail_timer += 1
          imu_sentence = "IIXDR,IMU_FAIL," + str(round(t_fail_timer / 60, 1))
          cs = format(reduce(operator.xor,map(ord,imu_sentence),0),'X')
          if len(cs) == 1:
                cs = "0" + cs
          imu_sentence = "$" + imu_sentence + "*" + cs
          sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
          sock.sendto(imu_sentence, (IMU_IP, IMU_PORT))
          t_fail = hack
          t_shutdown += 1

  if imu.IMURead():
    data = imu.getIMUData()
    fusionPose = data["fusionPose"]
    Gyro = data["gyro"]
    t_fail_timer = 0.0

    if (hack - t_damp) > .1:
        roll = round(math.degrees(fusionPose[0]), 1)
 pitch = round(math.degrees(fusionPose[1]), 1)
        yaw = round(math.degrees(fusionPose[2]), 1)
        rollrate = round(math.degrees(Gyro[0]), 1)
        pitchrate = round(math.degrees(Gyro[1]), 1)
        yawrate = round(math.degrees(Gyro[2]), 1)
        heading = yaw - magnetic_deviation
 if heading < 0.1:
            heading = heading + 360

        # Dampening functions
        roll_total = roll_total - roll_run[t_one]
        roll_run[t_one] = roll
        roll_total = roll_total + roll_run[t_one]
        roll = round(roll_total / 10, 1)
        heading_cos_total = heading_cos_total - heading_cos_run[t_three]
        heading_sin_total = heading_sin_total - heading_sin_run[t_three]
        heading_cos_run[t_three] = math.cos(math.radians(heading))
        heading_sin_run[t_three] = math.sin(math.radians(heading))
        heading_cos_total = heading_cos_total + heading_cos_run[t_three]
        heading_sin_total = heading_sin_total + heading_sin_run[t_three]
        heading = round(math.degrees(math.atan2(heading_sin_total/30,heading_cos_t
        if heading < 0.1:
            heading = heading + 360.0

        t_damp = hack
        t_one += 1
        if t_one == 10:
            t_one = 0
        t_three += 1
        if t_three == 30:
            t_three = 0

        if (hack - t_print) > 1:

            # health monitor
            sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            sock.sendto(str(hack), (MON_IP, MON_PORT))

            # iihdt true heading
            hdt = "IIHDT," + str(round(heading))[:-2] + ",T"
            hdtcs = format(reduce(operator.xor,map(ord,hdt),0),'X')
            if len(hdtcs) == 1:
                hdtcs = "0" + hdtcs
            iihdt = "$" + hdt + "*" + hdtcs

     # iixdr ahrs data
            xdr = "IIXDR,A," + str(int(round(roll))) + ",D,ROLL,A,"  + str(int(roun
            xdrcs = format(reduce(operator.xor,map(ord,xdr),0),'X')
            if len(xdrcs) == 1:
                xdrcs = "0" + xdrcs
            iixdr = "$" + xdr + "*" + xdrcs

            # assemble the sentence
            imu_sentence = iihdt + '\r\n' + iixdr + '\r\n'

            # to imu bus
            f = open('imu_bus', 'w')
            f.write(str(t_print) + ',' + str(heading) + ',' + str(roll)  + ',' + st
            f.close()

            # To kplex
            sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            sock.sendto(imu_sentence, (IMU_IP, IMU_PORT))

            t_print = hack

    time.sleep(poll_interval*1.0/1000.0)
```

Long story short--if it doesn't initialize, it will let you know via the XDR NMEA sentence, then shut down. The monitor will keep restarting it every 10 seconds. Also, if it stops receiving input for more than 5 seconds, it'll let you know that too.

Don't forget to update the magnetic deviation for your location.

It dampens out heading over three seconds, and roll over one second, and prints all available AHRS information (attitude, heading, and rates of change) via a sentence. Most of this is just for gee-whiz, but the heading is necessary, and the roll/pitch is useful for other functions in the next section.

Step 5: The Airmar DST800 Smart Sensor Python Script

dst.py

```
import serial
import math
import operator
import time
import socket
import os.path
import os

DST_IP = "127.0.0.3"
DST_PORT = 5005

MON_IP = "127.0.0.6"
MON_PORT = 5005

mon = 0
log = time.time()

# initialize for VDR
five = 0
setx_total = 0
setx_run = [0] * 5
sety_total = 0
sety_run = [0] * 5
drift_total = 0
drift_run = [0] * 5

# initialize for VLW
vlwfirst = 1
vlwinit = 0.0

# determine dst port
f = open('gps_port', 'r')
gps_port = f.readline()
```

```
f.close()
dst_port = "1"
if gps_port == "1":
    dst_port = "0"
ser = serial.Serial('/dev/ttyUSB' + dst_port, 4800, timeout=5)

while True:

    # health monitor
    hack = time.time()
    if hack - mon > .5:
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        sock.sendto(str(hack), (MON_IP, MON_PORT))
        mon = hack

    # log raw input
    dst_raw = ser.readline()
    f = open('dstraw', 'a')
    f.write(dst_raw)
    f.close()
    if hack - log > 60:
        os.remove('dstraw')
        f = open('dstraw', 'w')
        f.write("Serial port " + dst_port + ": " + time.asctime( time.localtime(tim
        f.close()
        log = hack

    # checking to see if it's a valid NMEA sentence
    if "*" in dst_raw:
        dst_split = dst_raw.split('*')
        dst_sentence = dst_split[0].strip('$')
        cs0 = dst_split[1][:-2]
        cs = format(reduce(operator.xor,map(ord,dst_sentence),0),'X')
        if len(cs) == 1:
            cs = "0" + cs

        # if it is a valid NMEA sentence
        if cs0 == cs:
            dst_vars = dst_sentence.split(',')
            title = dst_vars[0]

            # depth sentence
            if title == "SDDPT":

                # roll and pitch from imu
                try:
                    f = open('imu_bus', 'r')
                    line = f.readline()
                    f.close()
                    imu_split = line.split(',')
                    imu_hack = float(imu_split[0])
                    roll = float(imu_split[2])
                    pitch = float(imu_split[3])

                except ValueError:
                    time.sleep(.03)
                    f = open('imu_bus', 'r')
                    line = f.readline()
                    f.close()
                    imu_split = line.split(',')
                    imu_hack = float(imu_split[0])
                    roll = float(imu_split[2])
                    pitch = float(imu_split[3])

                # correct depth for 23 degree offset from centerline, but if roll/p
                # are from the last 3 seconds, correct depth for attitude
                depth = round(float(dst_vars[1])*math.cos(math.radians(23)),1)
                if time.time() - imu_hack < 3.0:
                   depth = round(float(dst_vars[1])*math.cos(math.radians(23-roll))

                # assemble the sentence with .5 meter offset from waterline
                dpt = "SDDPT," + str(depth) + ",0.50"
                dptcs = format(reduce(operator.xor,map(ord,dpt),0),'X')
                if len(dptcs) == 1:
                    dptcs = "0" + dptcs
                sddpt = "$" + dpt + "*" + dptcs + "\r\n"

                # to kplex
                sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
                sock.sendto(sddpt, (DST_IP, DST_PORT))

            # mean water temp sentence
            if title == "YXMTW":

                # convert to fahrenheit
                mtw = dst_vars[0] + "," + str(int(float(dst_vars[1]) * 9 / 5 + 32))
                cs = format(reduce(operator.xor,map(ord,mtw),0),'X')
                if len(cs) == 1:
```

```
        cs = "0" + cs
    yxmtw = "$" + mtw + "*" + cs + "\r\n"

    # to kplex
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.sendto(yxmtw, (DST_IP, DST_PORT))

# vessel waterspeed sentence and current set and drift
if title == "VWVHW":

    # heading and roll from imu
    try:
        f = open('imu_bus', 'r')
        line = f.readline()
        f.close()
        imu_split = line.split(',')
        imu_hack = float(imu_split[0])
        heading = float(imu_split[1])
        roll = float(imu_split[2])
    except ValueError:
        f.close()
        time.sleep(.03)
        f = open('imu_bus', 'r')
        line = f.readline()
        f.close()
        imu_split = line.split(',')
        imu_hack = float(imu_split[0])
        heading = float(imu_split[1])
        roll = float(imu_split[2])

    # course and groundspeed from gps
    try:
        f = open('gps_bus', 'r')
        line = f.readline()
        gps_split = line.split(',')
        f.close()
        gps_hack = float(gps_split[0])
        valid = gps_split[1]
        course = float(gps_split[2])
        groundspeed = float(gps_split[3])
    except ValueError:
        time.sleep(.03)
        f = open('gps_bus', 'r')
        line = f.readline()
        f.close()
        gps_split = line.split(',')
        gps_hack = float(gps_split[0])
        valid = gps_split[1]
        course = float(gps_split[2])
        groundspeed = float(gps_split[3])

    # calculate corrected waterspeed from heel and velocity
    waterspeed = float(dst_vars[5])
    sensor_angle = 23.0
    if time.time() - imu_hack < 3.0:
        sensor_angle = math.fabs(23.0-roll)
    five_knot_correction = -.02 * sensor_angle
    ten_knot_correction = sensor_angle * (.035 - .0065 * sensor_angle)
    if sensor_angle > 10.0:
        ten_knot_correction = -.03 * sensor_angle - 2
    if waterspeed < 5.0:
        waterspeed = round(waterspeed + (five_knot_correction * watersp
    else:
        waterspeed = round((waterspeed + (waterspeed * (ten_knot_correc

    # assemble the sentence
    vhw = "VWVHW,"
    if time.time() - imu_hack < 3.0:
        vhw = vhw + str(int(heading))
    else: vhw = vhw + ''
    vhw = vhw + ",T,,M," + str(waterspeed) + ",N,,K"
    vhwcs = format(reduce(operator.xor,map(ord,vhw),0),'X')
    if len(vhwcs) == 1:
        vhwcs = "0" + vhwcs
    vwvhw = "$" + vhw + "*" + vhwcs + "\r\n"

    # to kplex
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.sendto(vwvhw, (DST_IP, DST_PORT))

    # VDR set and drift
    if time.time() - imu_hack < 3.0 and time.time() - gps_hack < 3.0 ar

        heading_radians = math.radians(heading)
        course_radians = math.radians(course)
        set0 = course_radians - heading_radians
        if set0 < 0:
            set0 = set0 + 2 * math.pi
```

```
                    set_relative = math.pi - math.atan2(groundspeed * math.sin(set0
                    if waterspeed == 0 and groundspeed == 0:
                        set_relative = set0
                    set_radians = heading_radians + set_relative
                    if set_radians > (2 * math.pi):
                        set_radians = set_radians - (2 * math.pi)
                    drift = math.sqrt(pow(waterspeed,2) + pow(groundspeed,2) - 2 *

                    # dampen out set and drift over the last five readings
                    setx_total = setx_total - setx_run[five]
                    setx_run[five] = math.cos(set_radians)
                    setx_total = setx_total + setx_run[five]
                    setx_ave = setx_total / 5
                    sety_total = sety_total - sety_run[five]
                    sety_run[five] = math.sin(set_radians)
                    sety_total = sety_total + sety_run[five]
                    sety_ave = sety_total / 5
                    set_radians = math.atan2(sety_ave, setx_ave)
                    if set_radians < 0:
                        set_radians = set_radians + 2 * math.pi
                    set_true = math.degrees(set_radians)
                    set_apparent = set_true - heading
                    if set_apparent < 0:
                        set_apparent = set_apparent + 360

                    drift_total = drift_total - drift_run[five]
                    drift_run[five] = drift
                    drift_total = drift_total + drift_run[five]
                    drift = drift_total / 5
                    five = five + 1
                    if five > 4:
                        five = 0

                    # assemble the sentence
                    vdr = "IIVDR,A," + str(int(set_true)) + ",T,,,M," + str(round(d
                    vdrcs = format(reduce(operator.xor,map(ord,vdr),0),'X')
                    if len(vdrcs) == 1:
                        vdrcs = "0" + vdrcs
                    iivdr = "$" + vdr + "*" + vdrcs + "\r\n"

                    # ghost MWV sentence to show set/drift instead of wind
                    mwv = "IIMWV," + str(int(set_apparent)) + ",R," + str(round(dri
                    mwvcs = format(reduce(operator.xor,map(ord,mwv),0),'X')
                    if len(mwvcs) == 1:
                        mwvcs = "0" + mwvcs
                    iimwv = "$" + mwv + "*" + mwvcs + "\r\n"

                    # to kplex
                    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
                    sock.sendto(iivdr + iimwv, (DST_IP, DST_PORT))

            # voyage log sentence
            if title == "VWVLW":

                # calculate present trip total vs overall total
                if vlwfirst == 1:
                    vlwinit = dst_vars[1]
                    vlwfirst = 0
                trip = float(dst_vars[1]) - float(vlwinit)
                vlw = "VWVLW," + dst_vars[1] + ",N," + str(trip) + ",N"
                cs = format(reduce(operator.xor,map(ord,vlw),0),'X')
                if len(cs) == 1:
                    cs = "0" + cs
                vwvlw = "$" + vlw + "*" + cs  + "\r\n"

                # to kplex
                sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
                sock.sendto(vwvlw, (DST_IP, DST_PORT))
```

This is very similar to the GPS script, in that it finds the right port, checks for validity, and has a routine for each sentence. For the SDDPT sentence, it corrects for installation offset. My sensor is at a 23 degree angle from straight up and down, so the depth is skewed. Since we know the vessel pitch and roll from the previous section, we can correct the depth reading for the boat's attitude. Nice.

It converts the YXMTW water temperature to Fahrenheit, and it also calculates the correct VWVLW voyage log sentence. The DST only stores the total nautical mileage, and doesn't have a reset function. No problem, I built one in the script.

For the VWVHW vessel water speed sentence, it corrects the water speed reading from boat velocity and roll (see this post here for more on that). Not only that, but every time it receives a VWVHW sentence, and heading/course/groundspeed info is from the last 3 seconds, it calculates the VDR Set and Drift Sentence (see this post here). It dampens that out over the last 5 readings to give smooth data.

Since I don't have a wind anemometer, I also spoof the MWV sentence to reflect where the current is flowing towards, instead of the wind. That way, I can use my NKE Marine Electronics iOS app to show me what the current is doing just by looking at the wind circle.

Step 6: Setup a Wireless Kplex Multiplexer

Kplex is a fantastic tool, but to be totally honest, the documentation kind of sucks. It isn't written for computer morons like me, so I had to do a lot of testing and searching to figure it out. First, install it by doing this (in the kts directory of course):

```
git clone https://github.com/stripydog/kplex.git
cd kplex
make
sudo make install
```

Here is my kplex.conf file that accepts UDP inputs from the previous python scripts, and writes it all down to a log file, and prints it all to a TCP port that my laptop/phone can connect to. This kplex.conf file is also included in the github download from my page.

kplex.conf

```
# From GPS
[udp]
address=127.0.0.1
port=5005
direction=in

# From IMU
[udp]
address=127.0.0.2
port=5005
direction=in

# From DST
[udp]
address=127.0.0.3
port=5005
direction=in

# To TCP
[tcp]
mode=server
port=10110

# To Log File
[file]
filename=/home/pi/kts/kplex.output
direction=out
```

Simply save this as kplex.conf in the kts/scripts folder (where a copy already is if you downloaded my github) and copy it into the /etc/ directory with the following command in the terminal:

```
sudo cp kplex.conf /etc/
```

But we're not done. What address should you connect to on your phone or laptop? Find out with the following command:

```
ifconfig
```

and write down the number next to inet address of wla0 (mine is 10.0.0.1). When you add a connection in OpenCPN or the NKE Marine Electronics iOS app, use that number for the address, and use the port number that is in your kplex.conf file (if you used mine, the port number is 10110). Now, when you connect, you'll have a steady stream of NMEA data!

But let's make sure everything is working before we proceed. We're going the run the imu.py script, which will output NMEA data to the kplex port, and then run kplex as well with the raw text output:

```
python imu.py &

sudo kplex file:direction=out
```

That will output a text string of whatever kplex is receiving. Great. So now our instruments are outputting data, kplex is receiving, if you connect to the WiFi network and type in the inet address on your device (OpenCPN/NKE, with the port number specified in kplex.conf) you can receive over WiFi. So now, we just need to run the monitor script on its own so all this will happen automatically.

Step 7: Execute a Python Script at Startup on a Raspberry Pi

Save yourself some trouble, and just read this blog post here about it. I followed it, and have my results right here. Basically, it runs the init.sh script at startup, which executes the monitor.py script, which then starts everything.

init.sh

```
#!/bin/sh

### BEGIN INIT INFO
# Provides:          myservice
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Put a short description of the service here
# Description:       Put a long description of the service here
### END INIT INFO

# Change the next 3 lines to suit where you install your script and what you want t
DIR=/home/pi/kts/scripts
DAEMON=$DIR/monitor.py
DAEMON_NAME=monitor

# Add any command line options for your daemon here
DAEMON_OPTS=""

# This next line determines what user the script runs as.
# Root generally not recommended but necessary if you are using the Raspberry Pi GP
DAEMON_USER=root

# The process ID of the script when it runs is stored here:
PIDFILE=/var/run/$DAEMON_NAME.pid

. /lib/lsb/init-functions

do_start () {
    log_daemon_msg "Starting system $DAEMON_NAME daemon"
    start-stop-daemon --start --background --pidfile $PIDFILE --make-pidfile --user
    log_end_msg $?
}
do_stop () {
    log_daemon_msg "Stopping system $DAEMON_NAME daemon"
    start-stop-daemon --stop --pidfile $PIDFILE --retry 10
    log_end_msg $?
}

case "$1" in

    start|stop)
        do_${1}
        ;;

    restart|reload|force-reload)
        do_stop
        do_start
        ;;

    status)
        status_of_proc "$DAEMON_NAME" "$DAEMON" && exit 0 || exit $?
        ;;

    *)
        echo "Usage: /etc/init.d/$DAEMON_NAME {start|stop|restart|status}"
        exit 1
        ;;

esac
exit 0
```

You already have the monitor.py script from above, so now all that's left to do is make these two scripts "executable," and copy this init.sh to the right directory.

That's easy, all you have to run is the following commands, assuming you're in the kts/scripts folder:

```
sudo chmod a+x init.sh

sudo chmod a+x monitor.py

sudo cp init.sh /etc/init.d/

sudo update-rc.d init.sh defaults
```

Now it will execute init.sh at startup, which will execute monitor.py, which then executes all the instrument scripts and kplex.

Make note of two things: the very first line of monitor.py is "#!/usr/bin/env python" because it needs to know its a python script. Also, the first step in monitor.py is to change the working directory to the scripts folder for IMU calibration.

Final Thoughts

Now when you turn on your Raspberry Pi, all these scripts should start up and you'll receive good NMEA data through the wifi connection. If you want to check out the log files, you can ssh in with the same inet address we've been using:

```
ssh -Y pi@169.254.183.49

# Enter your password

cd kts/scripts

sudo nano log
```

or just nano whatever file you want to see. This way, you can debug/see what's up without a screen or a keyboard on your Pi when you're down on your boat with your laptop. Pretty useful, huh?

All in all, this is what we're looking at:

Raspberry Pi	$40
USB GPS	$30
MPU-9250	$12
DST800	$250
Cables/Misc	~$100
Total:	~$430

Not too bad for a wireless multiplexer, especially considering what it can do. Tilt-compensated compass. Data dampening. Wireless multiplexing. VDR Set and Drift. Correct paddle wheel readings. I am always interested in hearing what other applications people can find with the little device.

In another comment section, someone mentioned using it to automatically modulate a windlass based on depth--something that could be useful if you always want to drop out seven times the depth worth of chain for your anchor, or in his case, I believe it was fishing nets. Use the comments below to tell me your ideas.

Posted by Connor Olds at 10:17 AM

G+1 Recommend this on Google

Labels: Multiplexer, NMEA-0183, Raspberry Pi

6 comments:

Max44 May 30, 2016 at 4:55 PM

I looked at the link you provided for the depth sounder. How did you convert this from NMEA 2000 into NMEA 0183 Nice project.

Reply

Max44 May 30, 2016 at 4:56 PM

I looked at the link you provided for the depth sounder. How did you convert this from NMEA 2000 into NMEA 0183 Nice project.

Reply

Replies

Connor Olds June 2, 2016 at 12:09 PM

Thanks for pointing that out. I linked to the wrong model, so I've updated it to reflect the NMEA-0183 version (unfortunately, it's out of stock at the moment).

I do recall it being a little misleading, so I matched it up with the Garmin serial number found here (http://www.airmar.com/productdescription.html?id=110). The NMEA-0183 version has the Garmin serial 010-11051-10, while the NMEA-2000 has 010-11051-00 (note the last two digits). Amazon, while incredibly useful for online commerce, sometimes lacks appropriately descriptive titles. So while the title may say 0183 or 2000, always check the serial number in the advanced description.

Thanks for pointing that out!

Reply

sailor_max June 12, 2016 at 8:54 AM

Hello,
great project, thanks for the effort!!!

I do have a version of your system basically up and running but there is one issue, I couldn't solve.

Configuration:
RASPI 3 running Jessie
U-Blox NEO-6 GPS with Serial to USB converter (delivers NMEA strings on /dev/ttyUSB0)
Second serial-to-USB converter w/o any HW connected (this will take input from existing NMEA devices on the boat)
9 DOF IMU (http://www.ebay.de/itm/181877281002?_trksid=p2057872.m2749.l2649&ssPageName=STRK%3AMEBIDX%3AIT), delivers NMEA strings using the imu.py script.
No further HW connected yet.
SW installation as described above

My observations:
I had to change line 12 in monitor.py from os.chdir("home/pi/kts/scripts") to os.chdir("/home/pi/kts/scripts").
Your remark about
I2C read error from 104, 114 - Failed to read fifo count
then one of your cords probably became unplugged :/ (it happened to me).
In the "How to Setup the MPU-9250 on a Raspberry Pi" blog prevented me from beating the kids and burning the apartment (my cable was not unplugged but internally broken ;-( )

Without a second dummy USB converter, the script crashes

```
File "dst.py", line 38, in
ser = serial.Serial('/dev/ttyUSB' + dst_port, 4800, timeout=5)
File "/usr/lib/python2.7/dist-packages/serial/serialutil.py", line 261, in __init__ self.open()
File "/usr/lib/python2.7/dist-packages/serial/serialposix.py", line 278, in open raise SerialException("could not open port %s: %s" % (self._port, msg))
serial.serialutil.SerialException: could not open port /dev/ttyUSB1: [Errno 2] No such file or directory: '/dev/ttyUSB1'
Traceback (most recent call last): File "gps.py", line 83, in
course = float(gps_vars[8])
ValueError: could not convert string to float:
[2]+ Exit 1 python monitor.py
```

With the USB converter connected, the error message is as follows:

```
traceback (most recent call last):
File "gps.py", line 83, in
course = float(gps_vars[8])
ValueError: could not convert string to float:
```

and randomly

```
File "gps.py", line 96, in
f = open('gps_bus', 'w')
IOError: [Errno 13] Permission denied: 'gps_bus'
Traceback (most recent call last):
File "gps.py", line 83, in ………………….
```

As an output from kplex, I do only receive IMU data. GPS is missing completely.
"ps –ax" shows monitor.py, imu.py and kplex running, no gps.py.

Does this tell you anything? Your feedback is very much appreciated!

Matthias

Reply

Replies

Connor Olds June 15, 2016 at 9:26 PM

Okay, it *sounds* like what's happening here is that the names/directories of the USB devices are changing. When you plug a USB device into the Pi, it automatically assigns it (at random) USB0 or USB1 or USB2 or USB3, depending on how many are plugged in. If you only have one USB device plugged in, then it will only assign it USB0. When I say USB0, I mean the path: /dev/ttyUSB0 vs /dev/ttyUSB1

So if you have two USB devices plugged in, you will have them randomly assigned to USB0 and USB1 every time the Pi powers up. This makes the script damn near impossible to use. This is why I included a very clunky way in init.py to wait for USB reception from either of the ports, then write down which port that is, then start the dst.py and gps.py, which then reads the USB port which was determined in init.py.

This is not a good way to do this.

A better way, which I have not updated this post to reflect yet, is to assign each physical USB plugin a specific port. It's not THAT complicated to do, but my RPi box is down on my boat, so I can't update this post until I get that back (another two weeks or so).

Basically, with the setup above (as of right now), you'll always need a USB GPS plugged in, otherwise it won't work properly.

As for your other issue (could not convert string to float)... it's trying to open the gps_bus file to write the groundspeed and course to that file, which is then read by the dst.py script. However, typically whenever I got that "Permission denied" error, as you did, I had to delete the gps_bus file, restart the whole system, and then try again from a clean slate.

I hope this helps. If it doesn't, please post a little more detailed sequence of events to getting that second error and I'll see if I can help more (again... I might have to wait two weeks until I get the box back up here).

Reply

Unknown June 16, 2016 at 8:43 AM

Thanks for the reply and hints. I'll try to delete and restart and keep you posted on the result.
regards Matthias

Reply

Enter your comment...

Comment as: Bernard Mayer Fig **Sign out**

Publish **Preview** □ Notify me

Home Older Post

Subscribe to: Post Comments (Atom)

Simple template. Powered by Blogger.