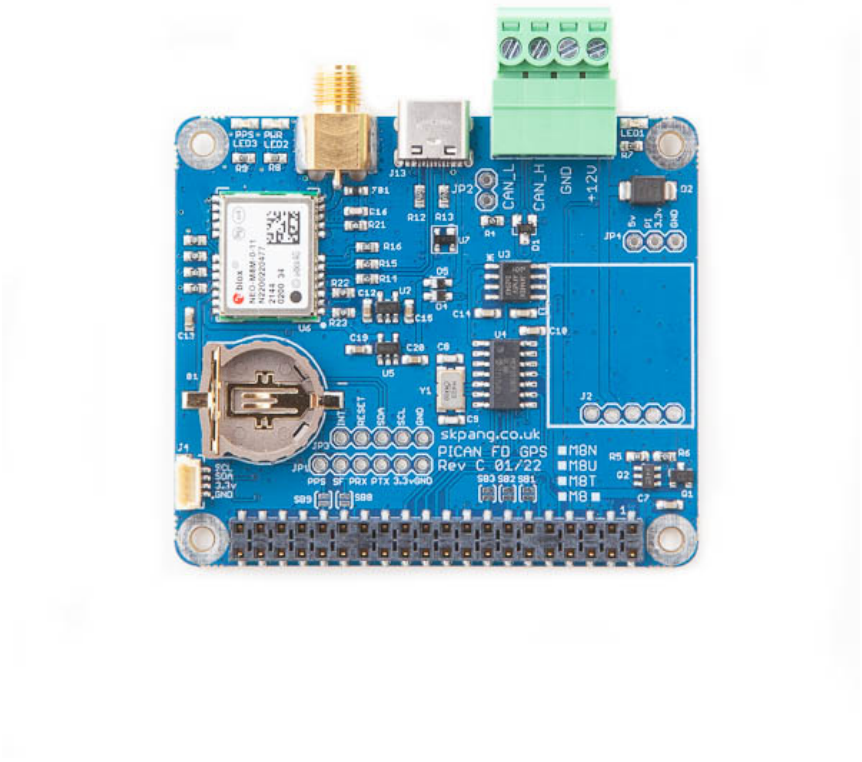


PiCAN FD and GNSS/GPS

With NEO-M8U

Untethered Dead Reckoning (UDR) with 3D sensors

USER GUIDE V1.0

Product name	PICAN FD and GPS Board for Raspberry Pi
Model number	RSP-PICANFD-NEO-M8U
Manufacturer	SK Pang Electronics Ltd

Contents

Table of Contents

1 Introduction

This PiCAN FD GPS/GNSS board provides Classic CAN and CAN FD Raspberry Pi. It uses the Microchip MCP2518FD CAN controller. GPS/GNSS is provided by ublox NEO-M8U Untethered Dead Reckoning (UDR) modules.

The improved CAN FD extends the length of the data section to up to 64 bytes per frame and a data rate of up to 8 Mbps.

Easy to install SocketCAN driver. Programming can be done in C or Python.

1.1 CAN Features

- Arbitration Bit Rate upto 1Mbps
- Data Bit Rate up to 8Mbps
- CAN FD Controller modes
- Mixed CAN2.0B and CANFD mode
- Conforms to ISO11898-1:2015
- High speed SPI Interface
- CAN connection via 4way plug in terminal
- LED indicator on GPIO04
- Four fixing holes, comply with Pi Hat standard
- SocketCAN driver, appears as can0 and can1 to application
- Interrupt RX on GPIO25

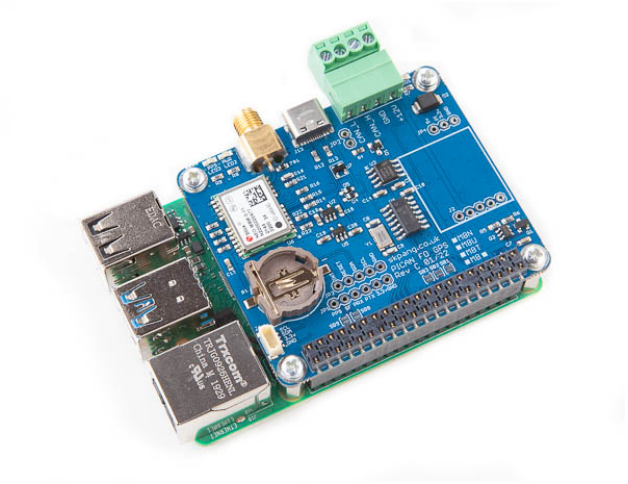
1.2 GPS GNSS Features

- U-blox NEO-M8U Untethered Dead Reckoning (UDR) module
- 72-channel u-blox M8 engine GPS/QZSS L1 C/A, GLONASS L10F, BeiDou B1 SBAS L1 C/A: WAAS, EGNOS, MSAS Galileo-ready E1B/C (NEO-M8N)
- Position accuracy 2.0 m CEP
- Acquisition
    - Cold starts: 26 s
    - Aided starts: 2 s
- Reacquisition: 1.5 s
- Sensitivity
    - Tracking & Nav: –167 dBm
    - Cold starts: –148 dBm
    - Hot starts: –156 dBm
- Assistance AssistNow GNSS Online
- AssistNow GNSS Offline (up to 35 days)
- AssistNow Autonomous (up to 6 days)
- OMA SUPL & 3GPP compliant
- Oscillator TCXO
- RTC crystal Built-In
- Anti jamming Active CW detection and removal.
- Extra onboard SAW band pass filter
- Memory Flash
- Supported antennas Active and passive
- Odometer Travelled distance
- Data-logger For position, velocity, and time

- Communicate to the Pi via UART or I2C
- PPS output to GPIO06
- Can work as a standalone using on board USB-C connection

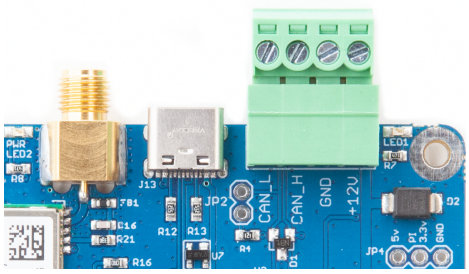
2 Hardware Installation

Before installing the board make sure the Raspberry is switched off. Carefully align the 40way connector on top of the Pi. Use spacer and screw (optional items) to secure the board.



2.1 Screw Terminals

The CAN connections are made via the 4way screw terminals.



J4	Function
1	CAN_L
2	CAN_H
3	GND
4	n/c

Note : The +12v In is only used on the PiCAN board with SMPS option fitted.

2.2 120Ω Terminator

There is a 120Ω fitted to the board. To use the terminator solder a 2way header pin to JP1 and JP2 then insert a jumper.

2.3 LED

There is a red LED fitted to the board. This is connected to GPIO04.

2.4 Not Fitted Items

Switch mode power supply, this is a 5v module that can power the Pi. It has an input voltage range of 7v to 24v.

3 Software Installation

It is best to start with a brand new Raspbian image. Download the latest from:

After first time boot up, do an update and upgrade first.

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

Add these lines to the end of file:

```
enable_uart=1
dtparam=spi=on
dtparam=i2c_arm=on
dtoverlay=mcp251xfd,spi0-0,interrupt=25
```

Reboot Pi:

```
sudo reboot
```

4 CAN-Bus Usage

4.1 Installing CAN Utils

Install the CAN utils by:

```
sudo apt-get install can-utils
```

4.2 Bring Up the Interface

You can now bring the CAN interface up with CAN 2.0B at 500kbps:

```
sudo /sbin/ip link set can0 up type can bitrate 500000
```

or CAN FD at 500kpbs / 2Mbps. Use copy and paste to a terminal.

```
sudo /sbin/ip link set can0 up type can bitrate 500000 dbitrate 2000000 fd on sample-point .8 dsample-point .8
```

Connect the PiCAN FD LIN board to your CAN network.

To send a CAN 2.0 message use :

```
cansend can0 7DF#0201050000000000
```

This will send a CAN ID of 7DF. Data 02 01 05 – coolant temperature request.

To send a CAN FD message with BRS use :

```
cansend can0 7df##15555555555555555
```

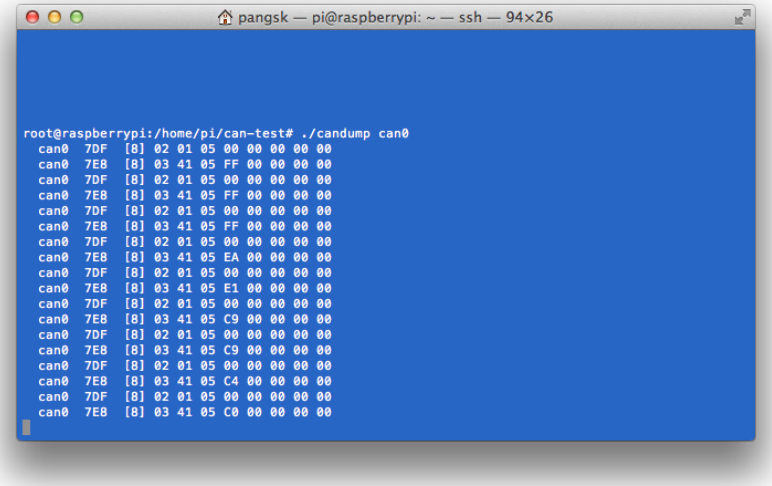
To send a CAN FD message with no BRS use :

```
cansend can0 7df##05555555555555555
```

Connect the PiCAN to a CAN-bus network and monitor traffic by using command:

```
candump can0
```

You should see something like this:



5 Python Installation and Use

Ensure the driver for PiCAN FD is installed and working correctly first.

Clone the pythonCan repository by:

```
git clone https://github.com/hardbyte/python-can
```

```
cd python-can
```

```
sudo python3 setup.py install
```

Check there is no error been displayed.

Bring up the can0 interface:

```
sudo /sbin/ip link set can0 up type can bitrate 500000 dbitrate 2000000 fd on sample-point .8 dsample-point .8
```

Now start python3 and try the transmit with CAN FD and BRS set.

```
python3

import can

bus = can.interface.Bus(channel='can0', bustype='socketcan',fd = True)

msg = can.Message(arbitration_id=0x7de,is_fd = True, bitrate_switch =
True,data=[0,0,0,0,0,0x1e,0x21,0xfe, 0x80, 0, 0,1,0])

bus.send(msg)
```

```
pi@raspberrypi:~ $ python3
Python 3.9.2 (default, Mar 12 2021, 04:06:34)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import can
>>> bus = can.interface.Bus(channel='can0', bustype='socketcan',fd = True)
>>> msg = can.Message(arbitration_id=0x7de,is_fd = True, bitrate_switch = True,data=[0,0,0,0,0,0x1e,0x21,0xfe, 0x80, 0, 0,1,0])
>>> bus.send(msg)
>>> bus.send(msg)
>>>
```

To received messages and display on screen type in:

```
notifier = can.Notifier(bus, [can.Printer()])
```

```
>>> msg = can.Message(arbitration_id=0x7de,is_fd = True, bitrate_switch = True,data=[0,0,0,0,0,0x1e,0x21,0xfe,
0x80, 0, 0,1,0])
>>> bus.send(msg)
>>> bus.send(msg)
>>> notifier = can.Notifier(bus, [can.Printer()])
>>> Timestamp: 1653164390.561713        ID: 0400    S Rx     F BS       DL: 64    44 64 56 35 74 56 74 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 46 74 67 56 74 56 74 56 74 56 74 00 00 00 00 04 56 74 65 70
 00 00 00 00 00 00 00 00 00 00 00 00 78 98 79 69    Channel: can0
Timestamp: 1653164393.817720        ID: 0400    S Rx     F BS       DL: 64    44 64 56 35 74 56 74 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 46 74 67 56 74 56 74 56 74 56 74 00 00 00 00 04 56 74 65 70 00
00 00 00 00 00 00 00 00 00 00 00 00 78 98 79 69    Channel: can0
Timestamp: 1653164398.897447        ID: 0400    S Rx               DL:  4    02 00 56 34                Chann
el: can0
Timestamp: 1653164403.721645        ID: 0400    S Rx               DL:  8    00 10 00 00 00 00 00 00    Chann
el: can0
Timestamp: 1653164406.113598        ID: 0100    S Rx     F BS       DL: 64    44 64 56 35 74 56 74 00 00 00 00
00 00 00 00 00 00 0d fa 23 53 25 ad ad 00 00 00 46 74 67 56 74 56 74 56 74 56 74 00 00 00 00 04 5a df a5 70 00
00 00 00 00 00 00 00 00 00 00 00 00 78 98 79 69    Channel: can0
Timestamp: 1653164406.825656        ID: 0100    S Rx     F BS       DL: 64    44 64 56 35 74 56 74 00 00 00 00
00 00 00 00 00 00 0d fa 23 53 25 ad ad 00 00 00 46 74 67 56 74 56 74 56 74 56 74 00 00 00 00 04 5a df a5 70 00
00 00 00 00 00 00 00 00 00 00 00 00 78 98 79 69    Channel: can0
>>>
```

Documentation for python-can can be found at :

https://python-can.readthedocs.io/en/stable/index.html

More expamles in github:

https://github.com/skpang/PiCAN-FD-Python-examples

6 GPS GNSS Usage

Ensure a GPS GNSS antenna is connected to board. Place the antenna outdoor with a good view to the sky.

LED3 flashes to indicate a lock.

6.1 Communicate over UART

To use the UART port ensure SB6 and SB7 are made.

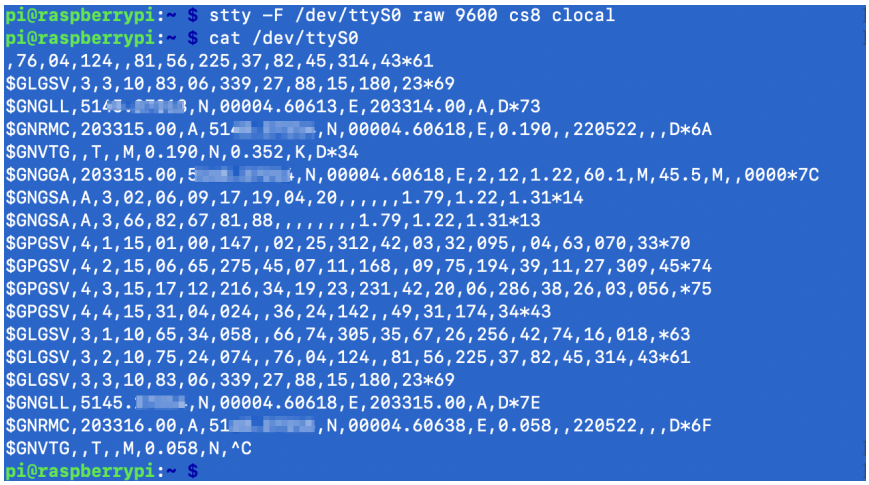
The default baudrate is 9600 8-N-1.

To do basic UART test on the Raspberry Pi, type in:

```
stty -F /dev/ttyS0 raw 9600 cs8 clocal
```

```
cat /dev/ttyS0
```

You should now see the GNSS/GPS sentences like this:



6.2 Communicate over I2C

To use the I2C port ensure SB4 and SB5 are made.

First install I2C tool:

```
sudo apt-get install i2c-tools
```

Now check I2C port:

```
sudo i2cdetect -y 1
```

Check address 0x42 has shown up.

```
pi@raspberrypi:~ $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- 42 -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@raspberrypi:~ $
```

Now download the i2c python script from Github:

https://github.com/skpang/Raspberrypi_i2c_gps

Start the script by:

```
python3 i2c-gps.py
```

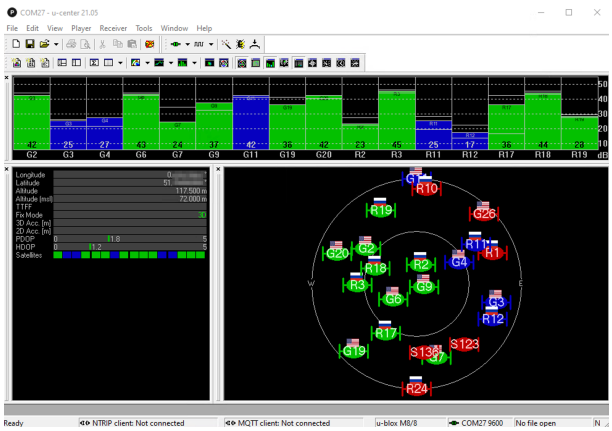
You should now see the GNSS/GPS sentences over I2C like this:

```
pi@raspberrypi:~ $ python3 i2c-gps.py
$GNVTG,,,,,,,,,N*2E
$GNGGA,205230.00,,,,,0,00,99.99,,,,,,*7E
$GNGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99*2E
$GPGSV,3,1,11,02,32,309,,03,25,100,,04,55,064,,06,67,253,*72
$GPGSV,3,3,11,26,06,049,,36,24,142,,49,31,174,*4F
$GLGSV,3,2,10,75,28,064,,76,12,119,,81,47,216,,82,50,301,*66
$GLGSV,3,3,10,83,15,337,,88,06,180,*63
$GNGLL,,,,,205230.00,V,N*52
$GNRMC,205231.00,V,,,,,,,220522,,,N*61
$GNGGA,205231.00,,,,,0,00,99.99,,,,,,*7F
$GNGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99*2E
$GNGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99*2E
$GPGSV,3,1,11,02,32,309,,03,25,100,,04,55,064,,06,67,253,*72
$GPGSV,3,3,11,26,06,049,,36,24,142,,49,31,174,*4F
$GLGSV,3,1,10,65,25,064,,66,75,348,,67,34,264,,74,12,010,*6A
$GLGSV,3,2,10,75,28,064,,76,12,119,,81,47,216,,82,50,301,*66
$GLGSV,3,3,10,83,15,337,,88,06,180,*63
$GNGLL,,,,,205231.00,V,N*53
```

6.3 Communicate over USB

The PiCAN FD GPS board can function as a standalone and communicate over USB. This requires an USB-C cable plug into the board. The USB-C also provides power to the board. This can be plugged into a PC.

On a Windows PC, u-center can be used:

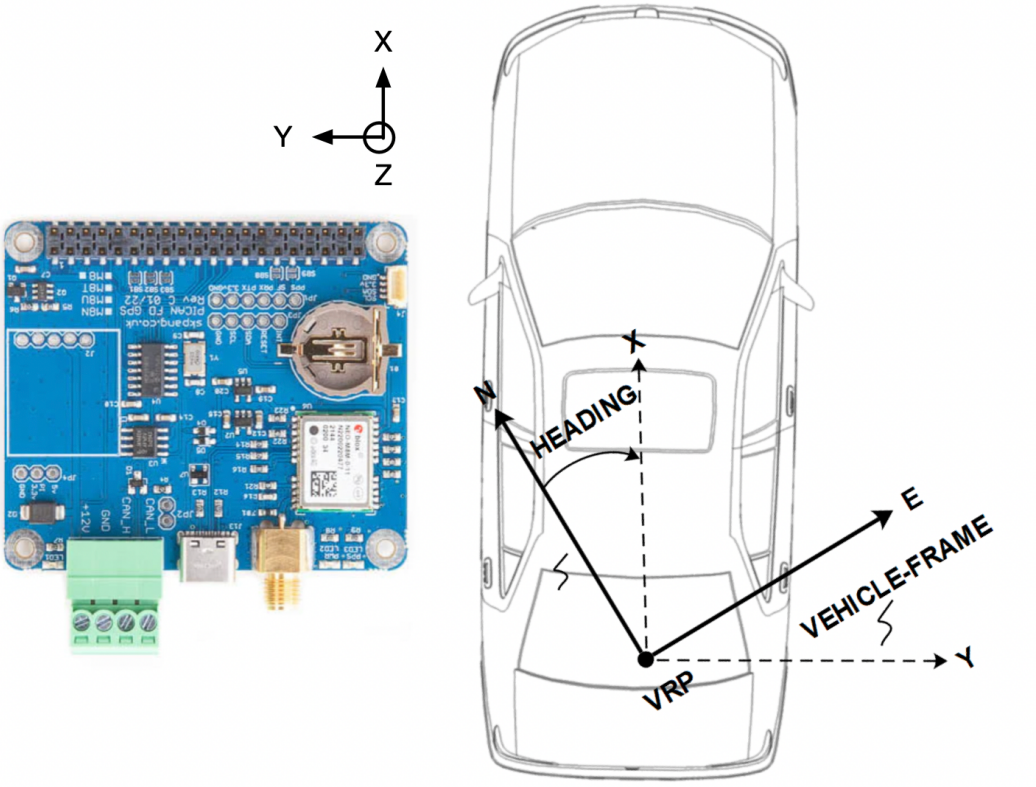


Firmware update for the GPS/GNSS can also be done over USB.

6.4 Using UDR module with 3D sensors

For detail of the Untethered Dead Reckoning (UDR) please read chapter 29 of the u-blox Protocol Specification.

The PiCAN FD NEO-M8U board must be mounted so the X-axis points towards the front of the vehicle as shown below.

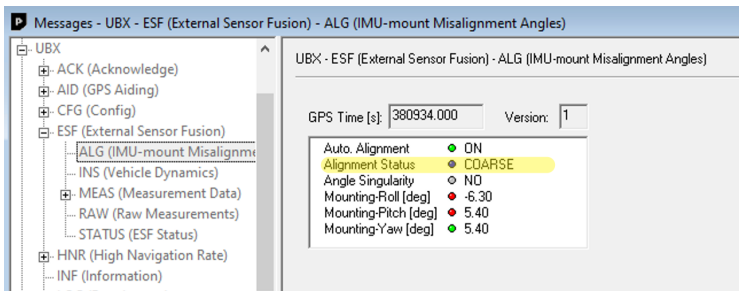


6.5 Sensor Calibration

After the board is mounted with the X-axis pointing to the front, the sensor must be calibrated. This is done automatically with the following procedure.

- Vehicle stationary first. LED3 (blue LED) blanking to indicator a GNSS lock
- Drive with left and right turns
- Reach a speed of over 30 km/h (19mph)

The status can be check with the u-center software, with Alignment Status:



This can be also check in your own software by Interrogating message UBX-ESF-ALG (0x10 0x14). See page 278 of the u-blox protocol specification.

Python3 example of Dead Reckoning by Sparkfun.

https://github.com/sparkfun/Qwiic_Ublox_Gps_Py/blob/master/examples/dead_reckoning_ex3.py