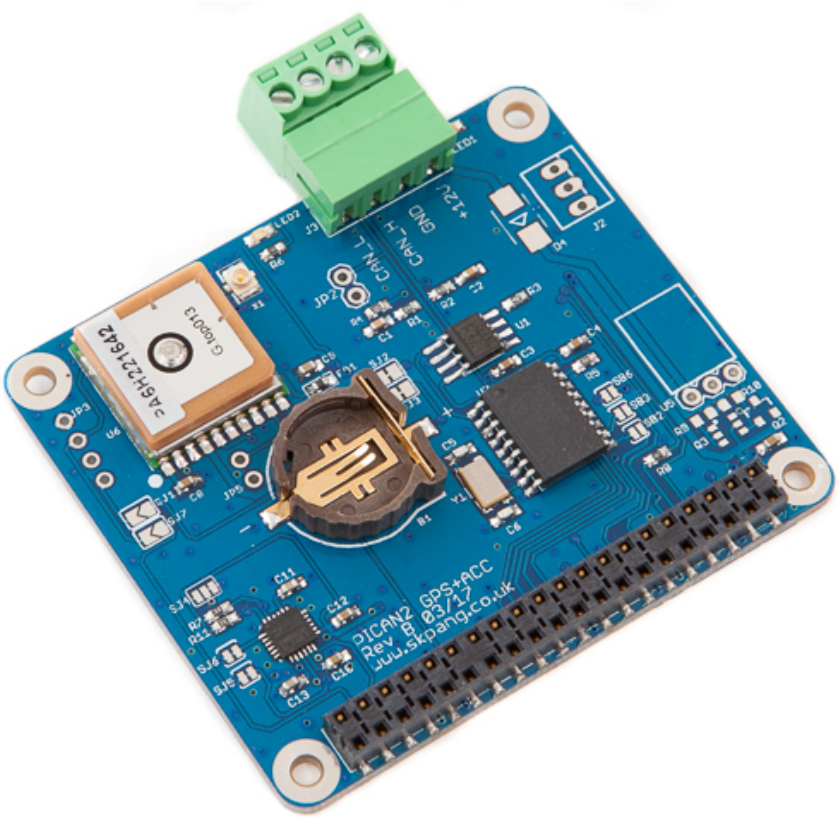


PiCAN GPS + Gyro +Accelerometer

USER GUIDE

V1.0

Product name	PiCAN GPS + Gyro + Accelerometer CAN-Bus Board for Raspberry Pi
Model number	RSP-PICANGPSACC
Manufacturer	SK Pang Electronics Ltd

Contents

Table of Contents

1. Introduction

This PiCAN GPS board provides CAN-Bus capability for the Raspberry Pi 2 and 3. It uses the Microchip MCP2515 CAN controller. CAN connections are made via 4 way screw terminal plug. GPS is provided by a 66 channels MTK3339 chipset module. An onboard battery holder for a CR1220 cell. The backup power is for the real time clock and help to reduce fix time. The GPS module has built in patch antenna but an external active antenna can also be use via the uFL connector.

The MPU-6050 provides six axis gyro and accelerometer with an onboard Digital Motion Processor™ (DMP™) capable of processing complex 9-axis MotionFusion algorithms, the MPU-6050 does away with the cross-axis alignment problems that can creep up on discrete parts.

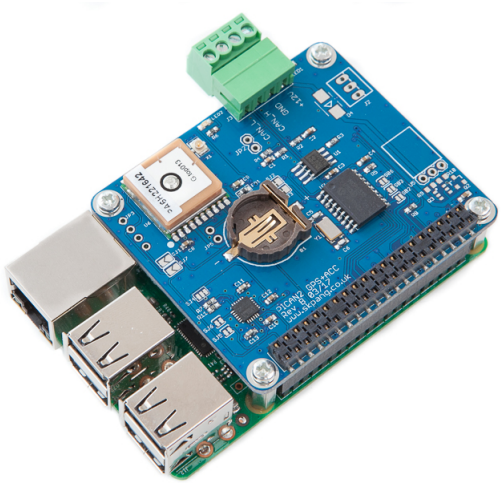
Easy to install SocketCAN driver. Programming can be done in C or Python.

1.1. Features

- CAN v2.0B at 1 Mb/s
- High speed SPI Interface (10 MHz)
- Standard and extended data and remote frames
- CAN connection via screw terminal
- 120Ω terminator ready
- Serial LCD ready
- LED indicator
- Four fixing holes, comply with Pi Hat standard
- SocketCAN driver, appears as can0 to application
- Interrupt RX on GPIO25
- MTK3339 chipset
- -165 dBm sensitivity, 10 Hz updates, 66 channels
- RTC battery holder
- Fix status LED
- On board patch antenna
- uFL connector for external active antenna
- I2C Digital-output of 6 or 9-axis MotionFusion data in rotation matrix, quaternion, Euler Angle, or raw data format
- Input Voltage: 2.3 - 3.4V
- Selectable Solder Jumpers on CLK, FSYNC and AD0
- Tri-Axis angular rate sensor (gyro) with a sensitivity up to 131 LSBs/dps and a full-scale range of ±250, ±500, ±1000, and ±2000dps
- Tri-Axis accelerometer with a programmable full scale range of ±2g, ±4g, ±8g and ±16g
- Digital Motion Processing™ (DMP™) engine offloads complex MotionFusion, sensor timing synchronization and gesture detection
- Embedded algorithms for run-time bias and compass calibration. No user intervention required

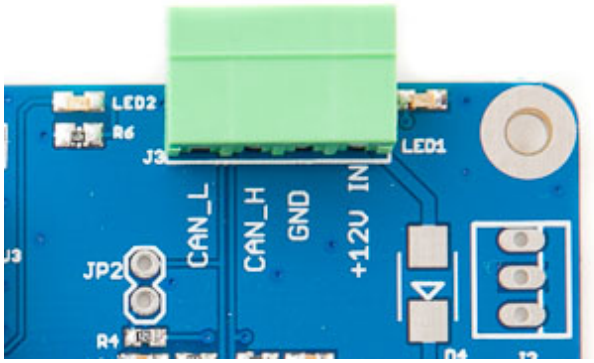
2. Hardware Installation

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.



1.2. Screw Terminal

The CAN connection can also be made via the 4 way screw terminal.



CAN0 (J3)	
Pin number	**Function**
1	n/c
2	GND
3	CAN_H
4	CAN_L

1.3. 120Ω Terminator

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

1.4. LED1

There is a red LEDs fitted to the board. This is connected to GPIO04. It can use as a general purpose indicator.

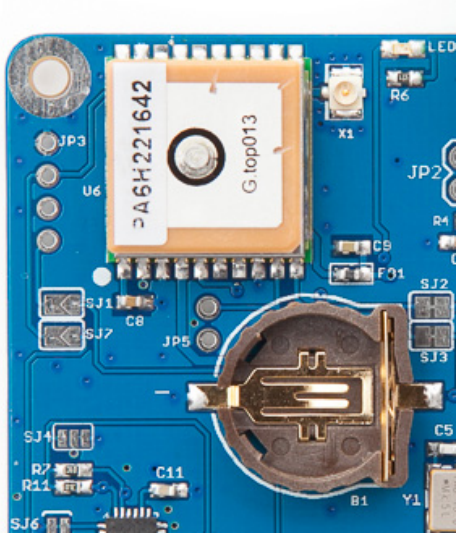
1.5. LED2

The LED blinks at about 1Hz while it's searching for satellites and blinks once every 15 seconds when a fix is found.

1.6. Not Fitted Items

J2 can be use to power a serial LCD with data on TXD line from the Pi. There is also 5v supply on J2.

1.7. External GPS Antenna and Backup Battery



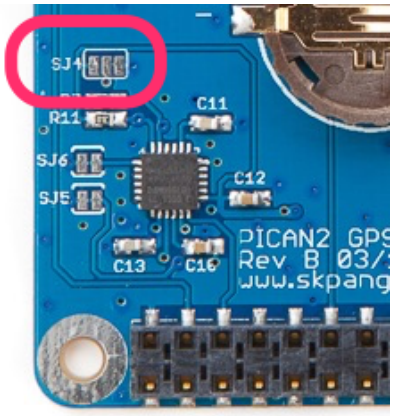
An external GPS antenna can be fitted via connector X1. This is an uFL connector. A uFL to SMA cable adapter is normally required.

A backup battery can be fitted for the RTC and help to reduce fix time.

1.8. GPS 1PPS (Pulse Per Second)

The GPS module has a 1PPS output that can be connected to the Pi's GPIO. To use the 1PPS close solder bridge SJ7. The pulses will now appear on GPIO26.

1.9. MPU-6050 I2C Address



The default I2C address for the MPU-6050 is 0x68.

An alternative address is 0x69. To change the address to 0x69, cut the track linking the default solder bridge on the right two pads on SJ4. Apply a solder bridge linking the left two pads.

3. Software Installation

The following procedure is for the Raspberry Pi 3.

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

https://www.raspberrypi.org/downloads/raspbian/

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

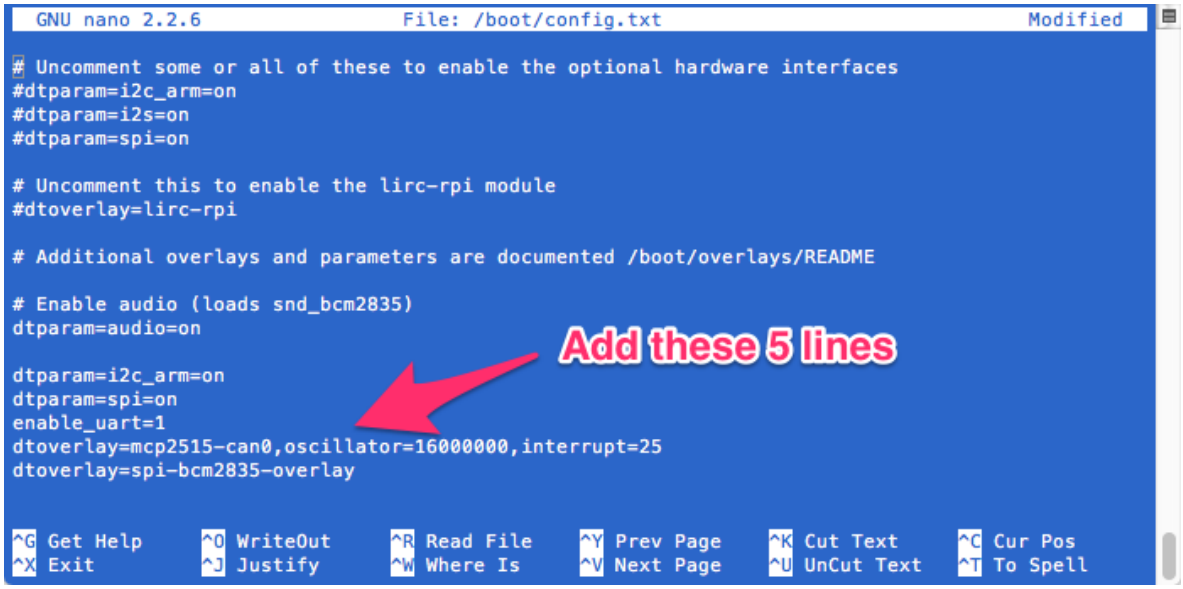
```
sudo apt-get update
sudo apt-get upgrade
sudo reboot
```

Add the overlays by:

```
sudo nano /boot/config.txt
```

Add these 5 lines to the end of file:

```
dtparam=i2c_arm=on
dtparam=spi=on
enable_uart=1
dtoverlay=mcp2515-can0,oscillator=16000000,interrupt=25
dtoverlay=spi-bcm2835-overlay
```



Edit the cmdline.txt by:

```
sudo nano /boot/cmdline.txt
```

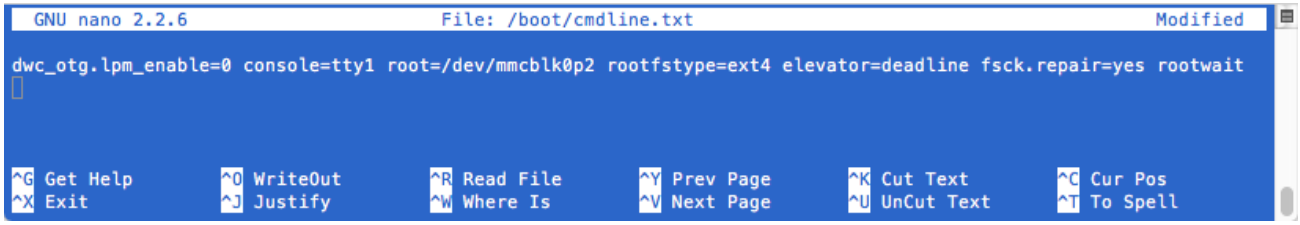
Change

```
dwc_otg.lpm_enable=0 console=serial0,115200 console=tty1
root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline
fsck.repair=yes rootwait
```

to

```
dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2
rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait
```

That is remove `console=serial0,115200`



Reboot Pi:

```
sudo reboot
```

1.10. MPU-6050 Test

First install the i2c-tools package by:

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

Now check if the MPU-6050 can be detected.

```
sudo i2cdetect -y 1
```

The screen should look like this:

Next install the python-smbus python module:

```
sudo apt-get install python-smbus
```

Now install example python files

```
git clone https://github.com/skpang/MPU-6050-examples.git
```

```
cd MPU-6050-examples
```

Run the test program

```
python3 mpu6060_test.py
```

Tilt and shake the board, you should see the value changes.

1.11. Basic GPS Test

The raw data can be read by:

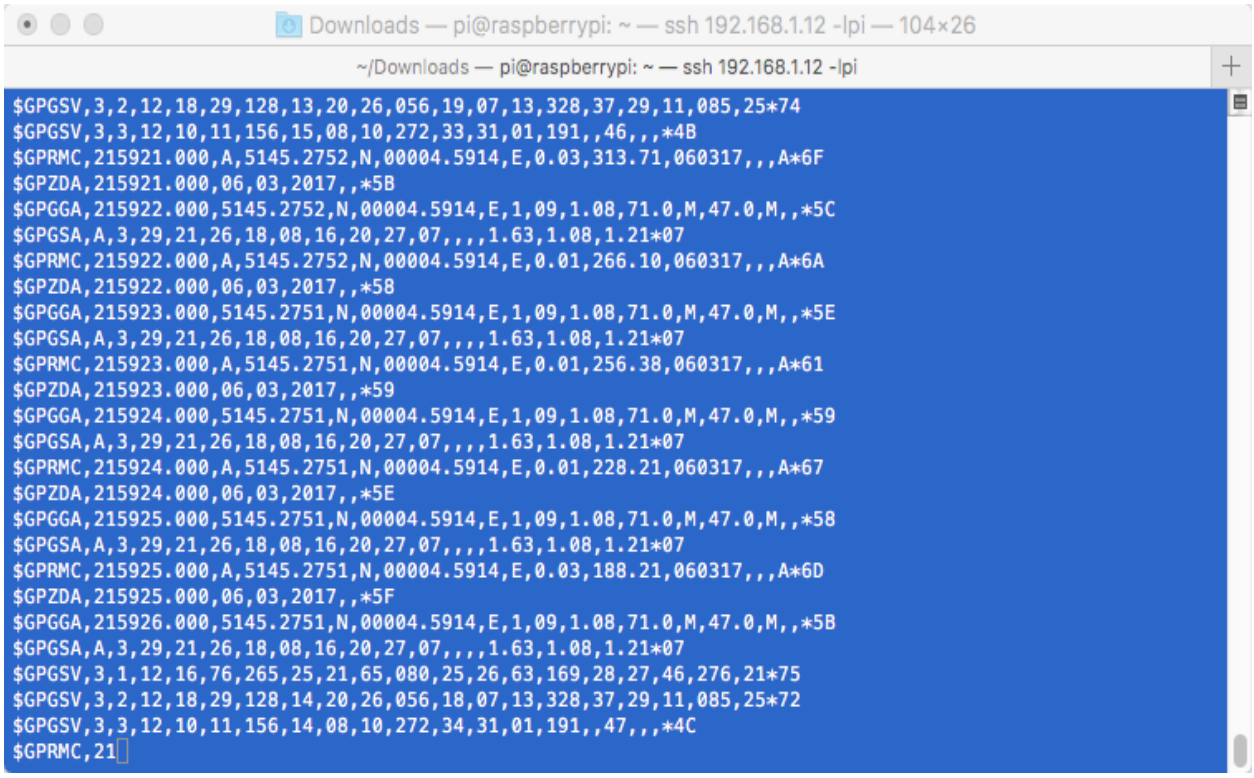
```
stty -F /dev/ttyS0 raw 9600 cs8 clocal –cstopb
```

Now read the data

```
cat /dev/ttyS0
```

If you see the data with lots of commas then it means the module has not got a fix yet. Make sure the GPS module has a clear view of the sky. Alternatively connect an external GPS antenna to the board and place the antenna outdoor.

You should see something like this:



1.12. Using GPSD

A program called GPSD can be use to view the data in a formatted way.

To install GPSD, type:

```
sudo apt-get install gpsd gpsd-clients python-gps
```

You need first to disable systemd service that GPSD installs:

```
sudo systemctl stop gpsd.socket
```

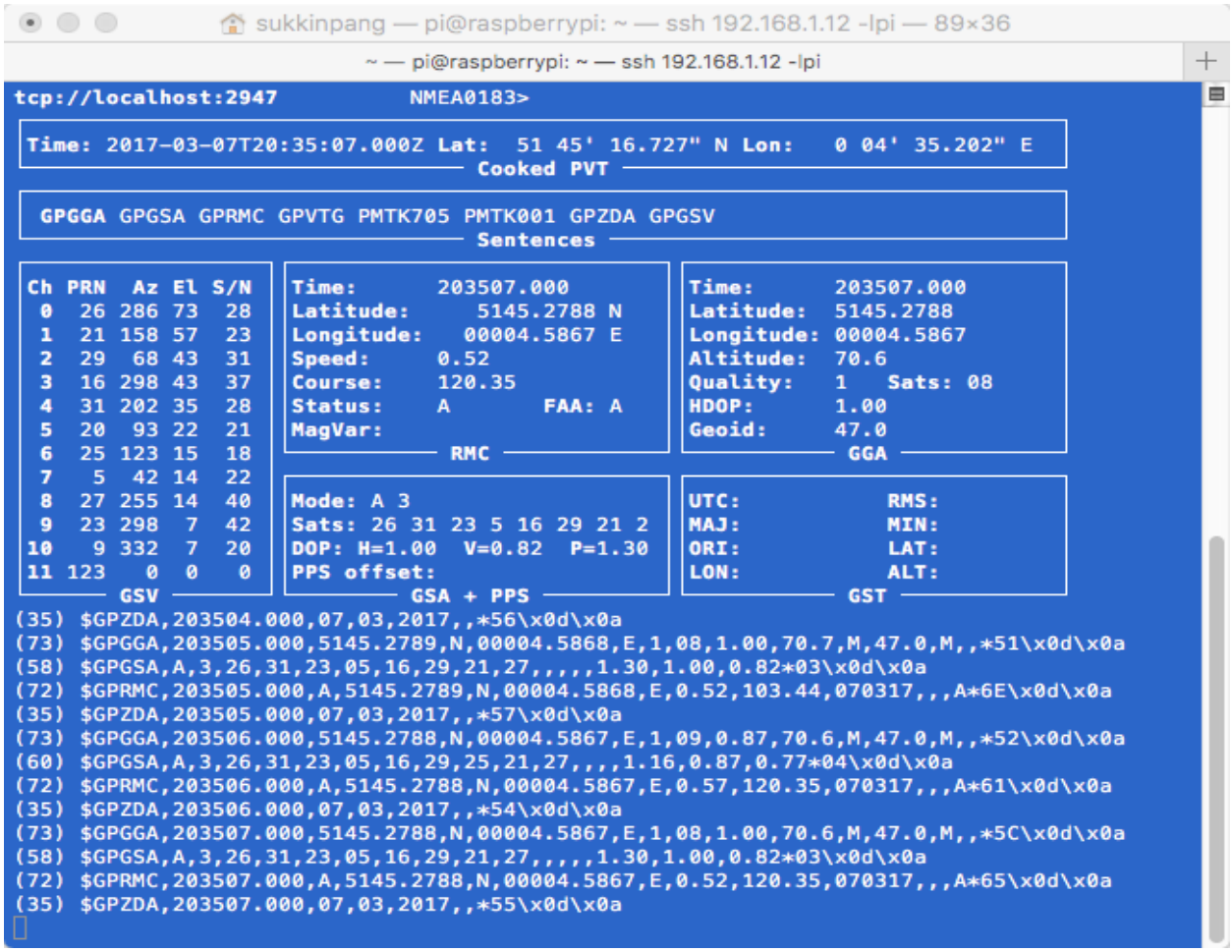
```
sudo systemctl disable gpsd.socket
```

To start GPSD, type:

```
sudo gpsd /dev/ttyS0 -F /var/run/gpsd.sock
```

```
gpsmon
```

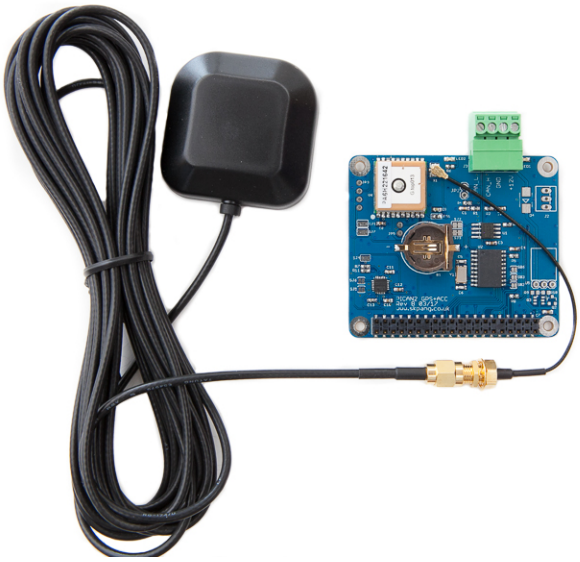
You should see this screen:



1.13. External GPS Antenna

The PiCAN-GPS board has built in patch antenna which provides a -165dBm sensitivity. However if your project is in a box and does not have a clear view of the sky then an external antenna is more suitable.

The optional external antenna has a SMA connector and an uFL to SMA cable adapter is required.



1.14. Bring Up the CAN Interface

You can now bring the CAN interfaces up:

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

Download and copy the CAN test programs to the Pi.

http://www.skpang.co.uk/dl/can-test_pi2.zip

Connect the PiCAN2 to your CAN network via screw terminal .

To send a CAN message on can0 (CAN B J4) use :

```
./cansend can0 7DF#0201050000000000
```

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

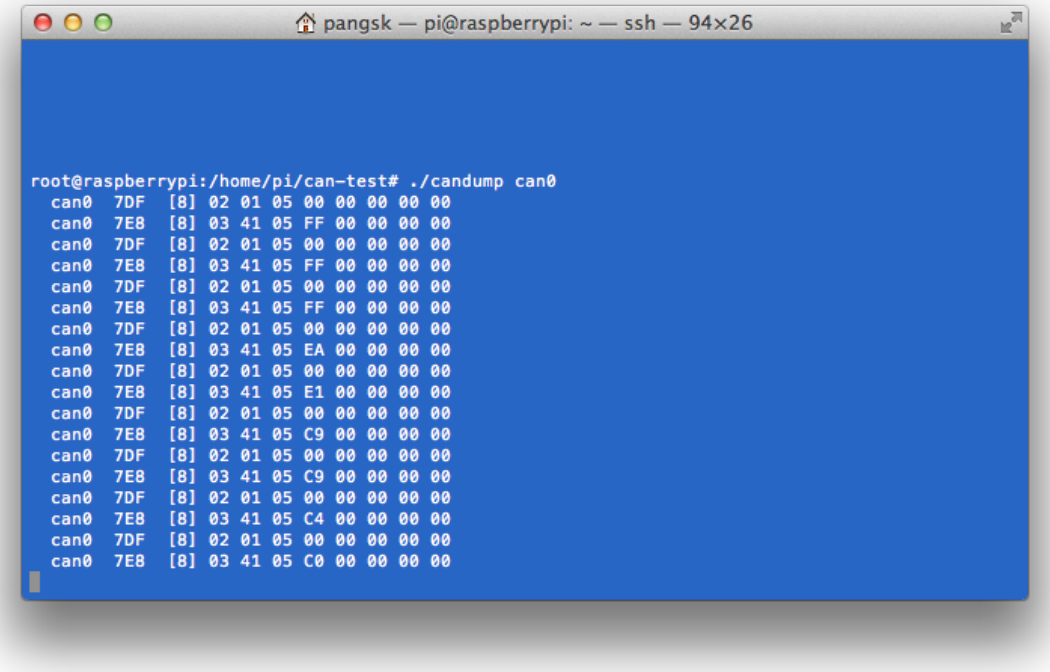
To send a CAN message on can1 (CAN A J3) use :

```
./cansend can1 7DF#0201050000000000
```

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

```
./candump can0
```

You should see something like this:



4. Writing Your Own Software

You can write your own application software in either C or Python.

1.15. Application in Python

Download the Python-CAN files from:

```
https://bitbucket.org/hardbyte/python-can/get/4085cffd2519.zip
```

Unzip and install by

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

Bring the CAN interface up if it is not already done:

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

Now start python3

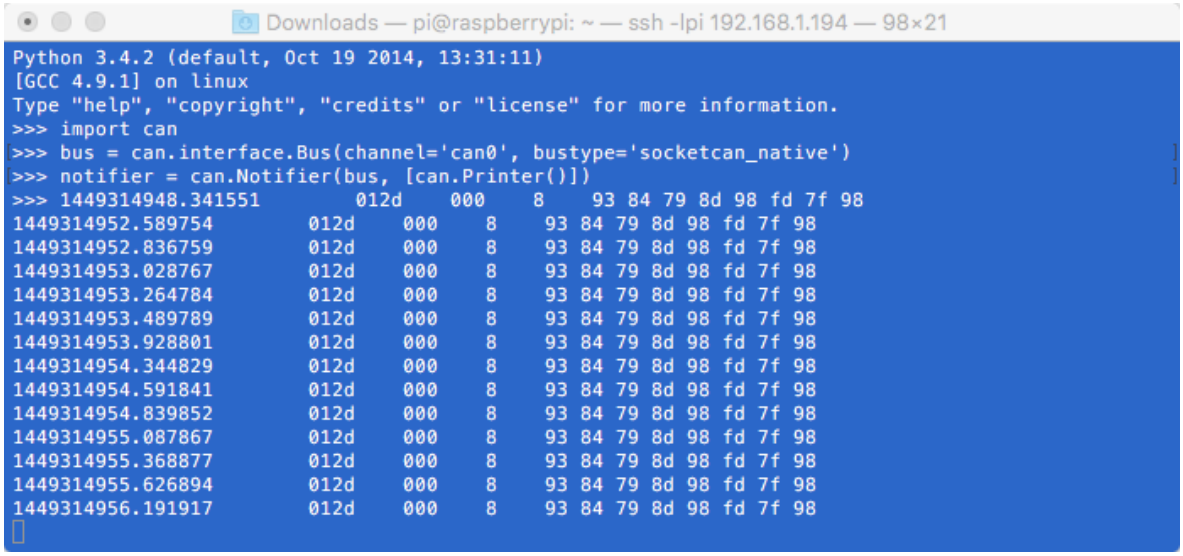
```
python3
```

To sent a message out type the following lines:

```
import can
bus = can.interface.Bus(channel='can0', bustype='socketcan_native')
msg = can.Message(arbitration_id=0x7de,
                  data=[0, 25, 0, 1, 3, 1, 4, 1],
                  extended_id=False)
bus.send(msg)
```

To received messages and display on screen type:

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



1.16. Application in C

Bring the CAN interface up if it is not already done:

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

Download the source code and example files by typing the following in the command prompt:

```
wget http://skpang.co.uk/dl/cantest.tar
```

Unpack the tar file and change into directory by:

```
tar xf cantest.tar
cd linux-can-utils
```

The example file is called cantest.c to edit this file, type the following in the command prompt:

```
nano cantest.c
```

Line 77 is the CAN message to be sent out.

```
unsigned char buff[] = "7DF#0201050000000000";
```

7DF is the message ID and 0201050000000000 is the data. Change the data to suit.
Press CTRL-X to exit.
To compile the program type:

```
make
```

Check there are no errors. To run the program type:

```
./cantest
```