OpenRTK Documentation

Aceinna Engineering

Apr 07, 2022

Table of contents

OpenRTK is an integrated GNSS (Global Navigation Satellite System) high precision chip and precisely calibrated Inertial Measurement Unit open-source platform for the development of navigation and localization algorithms. Users are able to quickly develop and deploy custom navigation/localization algorithms and custom sensor integrations on top of the OpenRTK platform. OpenRTK also has pre-built drivers in Python as well as a developer website - Aceinna Navigation Studio (ANS). These tools make logging and plotting data, including custom data structures and packets very simple.

Social: Twitter | Medium

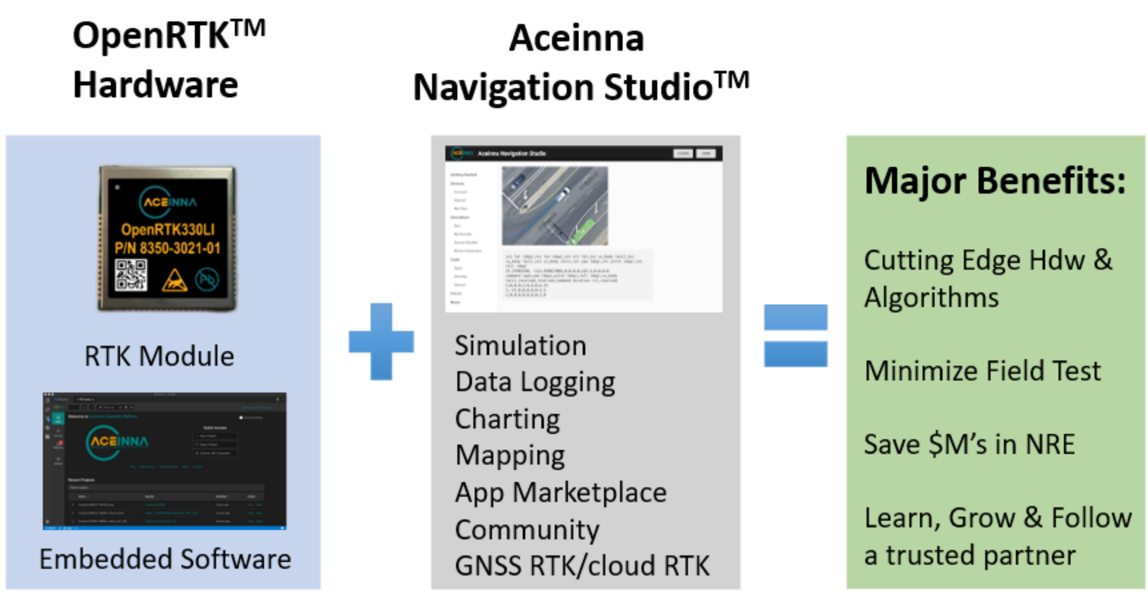
Part I

About OpenRTK

CHAPTER 1

Overview

OpenRTK is an integrated GNSS (Global Navigation Satellite System) high precision chip and precisely calibrated Inertial Measurement Unit open-source platform for the development of navigation and localization algorithms. A free Visual Studio Code (VSCode) extension is installed which contains all the software and tools necessary to create and deploy custom embedded sensor apps using OpenRTK. Visual Studio Code is the recommended IDE and the extension configures VS Code to include easy access to compilation, code download, JTAG debug, IMU and GNSS data logging as well as OpenRTK platform updates and news. A developer website called Aceinna Navigation Studio (ANS) includes additional support tools including a GUI for controlling, plotting and managing data files logged by your Custom RTK/IMU module. About OpenIMU, you can refer to Aceinna OpenIMU Developer Manual.



The OpenRTK and ANS platform and tool-chain are supported on all three Major OS cross-development platform:

- Windows 7 or 10

- MAC OS 10
- Ubuntu 14.0 or later

Note: Contributions to the public repositories related to this project are welcomed. Please submit a pull request.

The following pages cover:

- What is OpenRTK
- What is the Acienna Navigation Studio
- Who is using OpenRTK and the Acienna Navigation Studio

1.1 What is OpenRTK?

OpenRTK is an open source hardware and software platform for development of high-performance navigation and localization applications on top of multi-constellation, multi-frequency Global Navigation Satellite System (GNSS) chips, a family of low-drift pre-calibrated Inertial Measurement Units (IMU) and cloud based server supports.

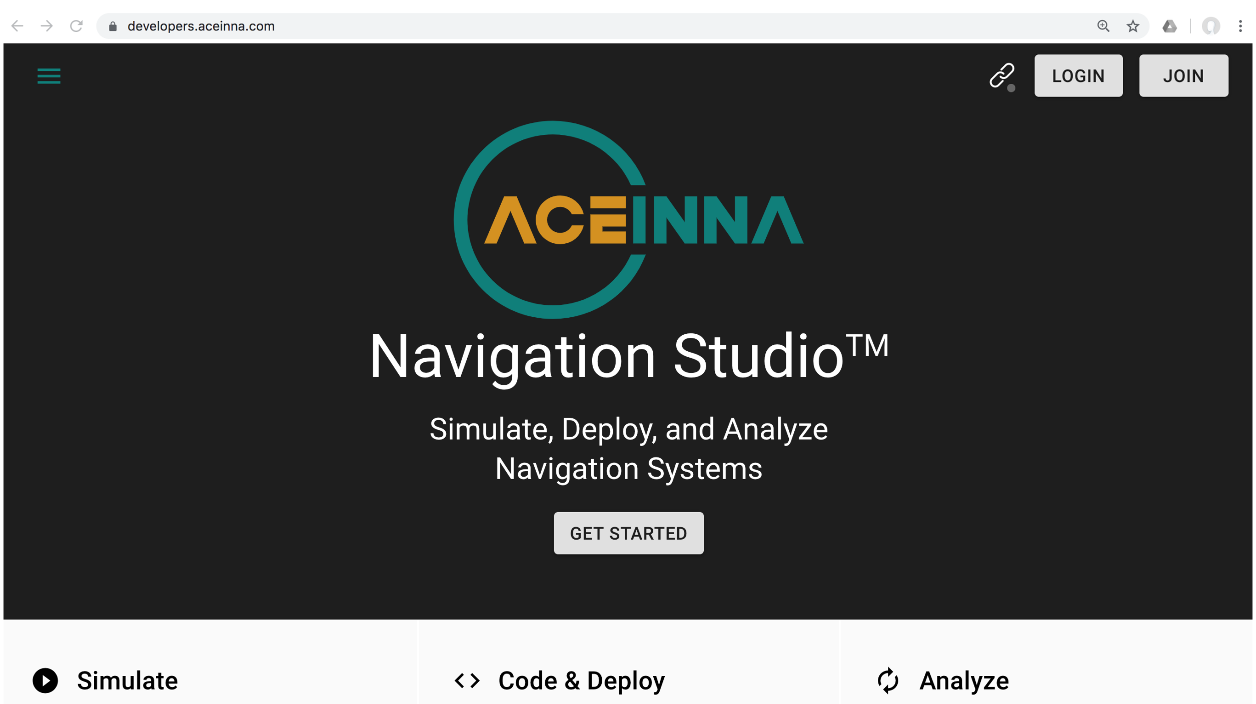
- **Hardware**
  - OpenRTK hardware features of a multi-frequency, multi-constellation GNSS chipset from STMicroelectronics (aka ST), a triple-redudant 6-axis IMU sensor module from Aceinna, and an embedded STM32 ARM Cortex-M4 MCU with floating-point computation support for complex positioning engine
  - Spare I/O and ports for external sensors such as Odometer, camera for enhanced sensor fusion navigation
  - There comes two form-factors as follows:

Model	Description
OpenRTK330LI	Inertial Navigation System Module – Industrial Grade
RTK330LA	Inertial Navigation System Module – Automotive Grade (Contact Aceinna)

- **Software**
  - OpenRTK embedded software (i.e. the module firmware) is developped on top of the standard STM32 Cortex MCU library
  - Utilizes the FreeRTOS as the real time operating system for MCU
  - Provides a cost-free embedded environment and toolchain using VS Code and the associated Aceinna extension (based on PlatformIO)
  - Features with open-sourced firmware in the drivers and user interfaces, user can use or modify the provided firmware code to utilize or customize:
    * raw IMU data generation in sensor data extraction, pre-filtering and output data rate/format/interface and so on
    * UART input/output baudrate/mode/messages
    * CAN input/output mode/messages
    * Ethernet driver and input/output mode/messages
    * SPI driver
    * Bluetooth driver

- Features with proprietary positioning engine library (NOT open-sourced):
  - GNSS Real Time Kinematic (RTK) positioning engine
  - GNSS/IMU integrated Inertial Navigation System (INS) positioning engine
- **Cloud Service**
  - OpenRTK cloud service provides Networked Transport of RTCM via Internet Protocol (NTRIP) server and caster service for GNSS correction data
  - Provides online developer site for user interface
    - Web GUI
    - Data and algorithm simulation
    - Database for storage
    - Live support forum

1.2 What is Aceinna Navigation Studio?



- The Aceinna Navigation Studio (https://developers.aceinna.com) is a navigation system developer's website and web-platform.
- It consists of a graphical user interfSace to control and configure OpenRTK units.
- Using a JSON configuration file ("openrtk.json"), the graphical user interface can be customized for user specific messaging and settings without any additional coding. This aligns the embedded code with both the Python device server and the GUI pages available on ANS (https://developers.aceinna.com).
- Online tools include graphing, mapping, logging, simulation, GNSS RTK, and GNSS cloud RTK.
- User Forum is available at (https://forum.aceinna.com).

Python & the Acienna Navigation Studio

The Acienna Navigation Studio (ANS) requires Python to operate. If the user has not installed Python, it can be installed from https://www.python.org/downloads/. Download and install the latest version.

An open-source Python driver for openrtk is available and required. The Python driver can be used directly from the terminal to load, log, and test your application. The driver leverages the PySerial library to connect to an OpenRTK of a serial connection. The python script supports configuring units, firmware updates (JTAG is faster for debugging), and local data logging.

In addition, the open-source Python driver can acts as a server connecting the OpenRTK hardware with our ANS developer platform for a GUI experience, cloud data storage and retrieval, as well as stored file charting/plotting tools.

The Aceinna VS Code extension ensures a python environment automatically. The OpenRTK python code can be installed independently by cloning the repository https://github.com/Aceinna/python-openimu or using pip as shown below.

```
pip install openimu
```

1.3 Who is using it?

OpenRTK is to be used for commercial applications in agriculture, transportation, unmanned vehicles, machine control, marine navigation, and other industries where efficiencies can be gained from the application of precise, continually available position and time information.

1.3.1 Applications

Unmanned Vehicles

Initially, unmanned vehicles were used primarily by the defense industry. However, as the unmanned vehicle market has grown and diversified, the commercial use of unmanned vehicles has also grown and diversified. Some of the current civilian uses for unmanned vehicles are: search and rescue, crop monitoring, wildlife conservation, aerial photography, environmental research, infrastructure inspection, bathymetry, landmine detection and disposal, HAZMAT inspection and disaster management. As the civilian unmanned vehicle market expands, so will the civilian use of unmanned vehicles.

Machine Control

GNSS technology is being integrated into equipment such as bulldozers, excavators, graders, pavers and farm machinery to enhance productivity in the realtime operation of this equipment, and to provide situational awareness information to the equipment operator. The adoption of GNSS-based machine control is similar in its impact to the earlier adoption of hydraulics technology in machinery, which has had a profound effect on productivity and reliability.

Precise Agriculture

In precision agriculture, GNSS-based applications are used to support farm planning, field mapping, soil sampling, tractor guidance, and crop assessment. More precise application of fertilizers, pesticides and herbicides reduces cost and environmental impact. GNSS applications can automatically guide farm implements along the contours of the earth in a manner that controls erosion and maximizes the effectiveness of irrigation systems. Farm machinery can be operated at higher speeds, day and night, with increased accuracy. This increased accuracy saves time and fuel, and maximizes the efficiency of the operation. Operator safety is also increased by greatly reducing fatigue.

Note: This product has been developed exclusively for commercial applications. It has not been tested for, and makes no representation or warranty as to conformance with, any military specifications or its suitability for any military application or end-use. Additionally, any use of this product for nuclear, chemical or biological weapons, or weapons research, or for any use in missiles, rockets, and/or UAV's of 300km or greater range, or any other activity prohibited by the Export Administration Regulations, is expressly prohibited without the written consent and without obtaining appropriate US export license(s) when required by US law. Diversion contrary to U.S. law is prohibited. Specifications are subject to change without notice.

Part II

Tutorial

CHAPTER 2

Quick Start

Contents

Note: if the figures are blur, click on the figure to see the clearer version

2.1 OpenRTK330LI EVK Introduction

The OpenRTK330LI Evalution Kit (EVK) is designed to evaluate the OpenRTK330LI module with the online Aceinna Navigation Studio (ANS) and related software stack. A full set of OpenRTK330 EVK is shown below after you unpack the product box.

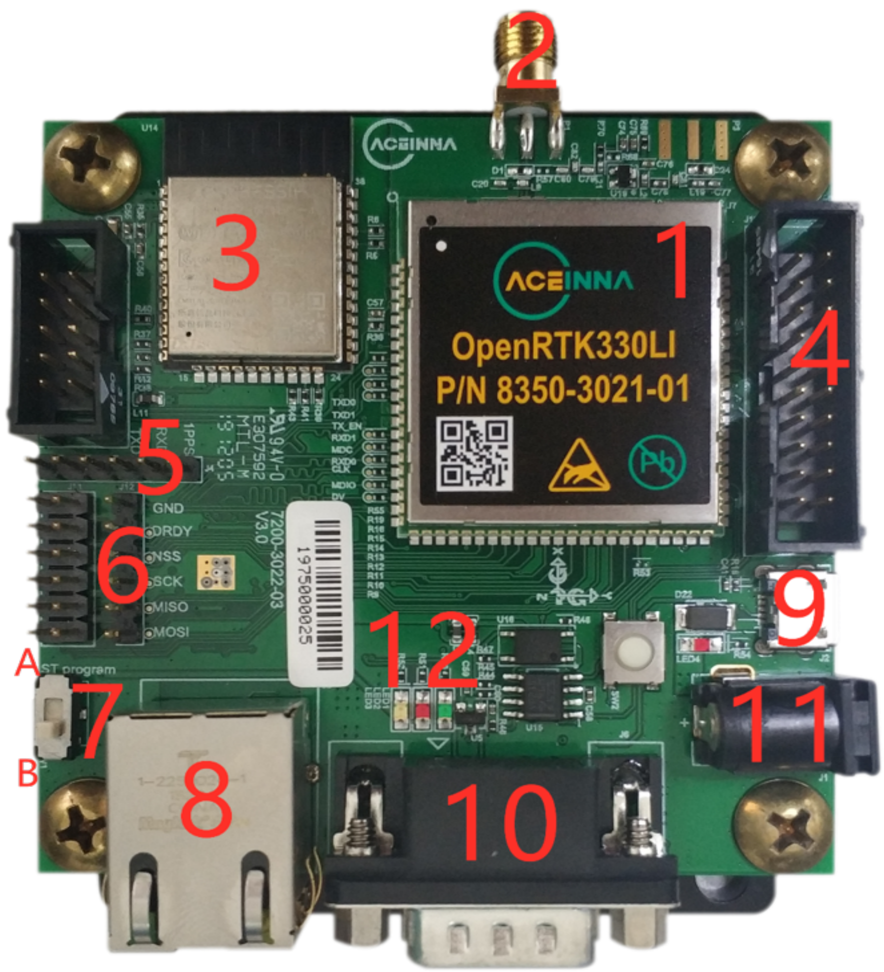
where

- 1: ST-Link debugger
- 2: Multi-Constellation and Multi-frequency GNSS antenna, supports
  - GPS L1/L2/L5
  - GLONASS L1/L2
  - GALILEO E1/E5/E6
  - BEIDOU B1/B2
- 3: Micro-USB cable


1
2
3
4
5
OpenRTK330LI
P/N 8350-3021-01

- 4: OpenRTK330 Evaluation Board (EVB) with metal flat mounting board
- 5: 12-V DC adapter with 5.5 x 2.1 mm power jack

The picture below shows the detailed overview of OpenRTK330 EVB



where some of the parts are listed here

- 1: OpenRTK330 GNSS/IMU integrated module
- 2: GNSS antenna SMA interface
- 3: Espressif ESP32 bluetooth module
- 4: SWD/JTAG connector, 20-pin
- 7: Boot mode swtich
  - Position A: booting from bootloader

    - Position B: normal working mode

- 8: RJ45 jack for Ethernet connection
- 9: Micro-USB port
- 10: 9-pin CAN interface
    - Pin-7: CAN_H signal
    - Pin-2: CAN_L signal
- 12: EVB working status LEDs, yellow, red, and green LED from left to right

2.2 Quick Setup and Usage

2.2.1 Prerequisites

Hardware

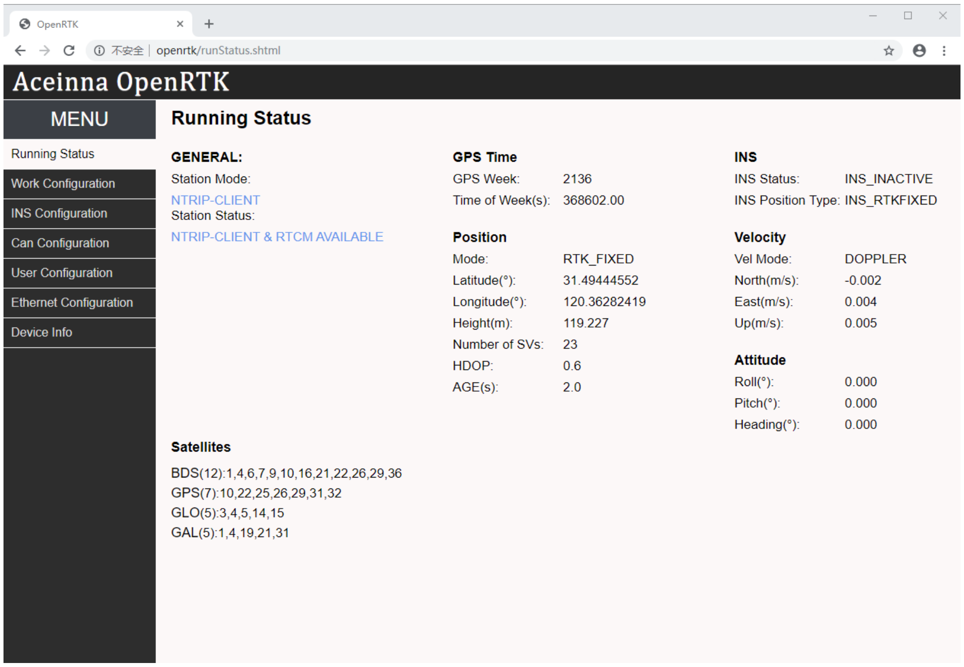
- OpenRTK330LI EVK
- Ethernet cable (must have, not included in the EVK)
- Ethernet router/network switch (optional, not included in the EVK)

Software

- The online Aceinna Navigation Studio (ANS) deverloper website, manily for
    - OpenRTK devices management and technical forum and support
    - Web-based Graphical User Interface (GUI)
    - App center for online firmware upgrade
- The OpenRTK Python driver: Python based program runs on a PC, click here to download the latest version of executables
    - Send/Receive data from ANS to enable Web GUI and online firmware upgrade for OpenRTK330LI device
    - Log and parse OpenRTK330LI output data, positioning solution and other debug information to binary and text files

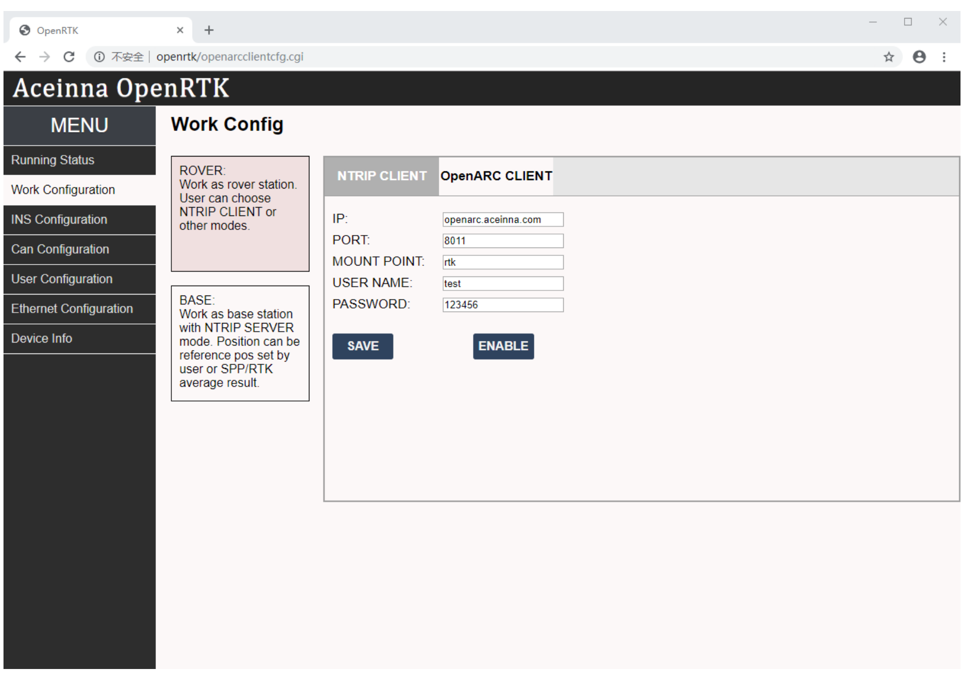
2.2.2 Usage Steps

1. **Power and data link**: connect the EVB with a PC using a Micro-USB cable, and the **YELLOW** LED (#12 on the EVB figure above) flashes. The EVB is powered on, and four serial com ports are established on the PC.
2. **Antenna**: connect a GNSS multi-frequency antenna to the SMA interface (#2 on the EVB figure), the **GREEN** LED (#12 on the EVB figure above) flashes if the incoming GNSS signal is valid
3. **Network**: Use an Ethernet calbe to connect the EVB with a network router or switch, and then connect a PC to the same router/switch using an Ethernet cable. The OpenRTK330LI EVB gets internet access and assigned an IP address in the local network via DHCP.
4. **GNSS RTK and INS Configuration**: open a browser (Google Chrome is recommended), visit http://openrtk,
    - You will firstly see the following device running status page

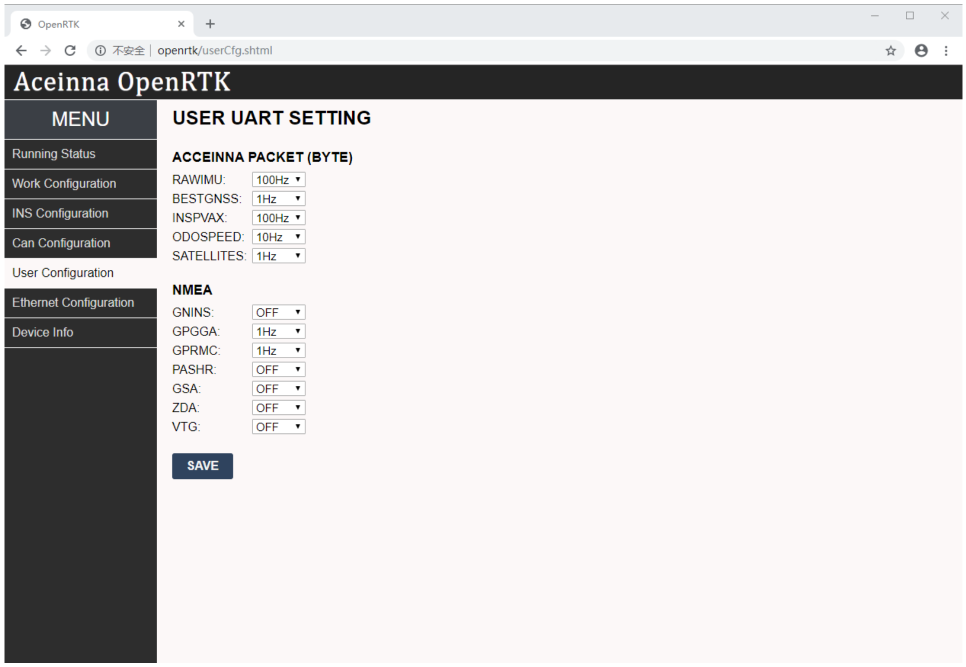


- On the left side menu bar, click "Work Configuration" tab to choose the following working mode of the device and configure it accordingly:
  - Rover: works as a nomarl GNSS positioning unit that is also referring to "NTRIP client" receiving GNSS data correction
  - Base: works as a GNSS reference station with known position and sending GNSS data to "NTRIP server" to be used as GNSS data correction

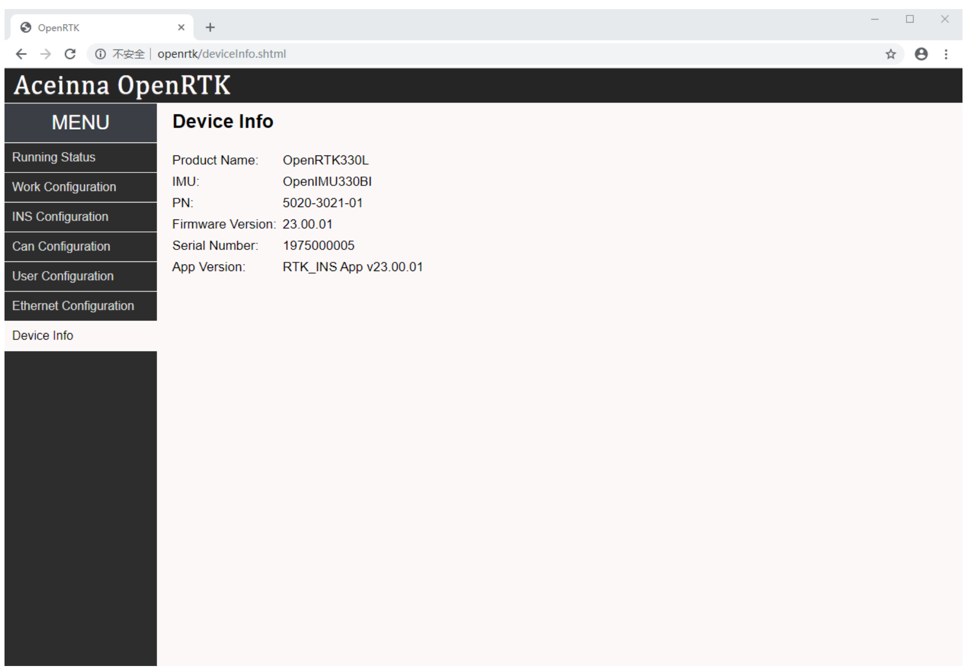
  Please refer to the "How-to-use" chapter for the detailed configurations.



- On the left side menu bar, click "User Configuration" tab to select the user output data and rate among the options provided, including NMEA0183 messages and Aceinna format binaries



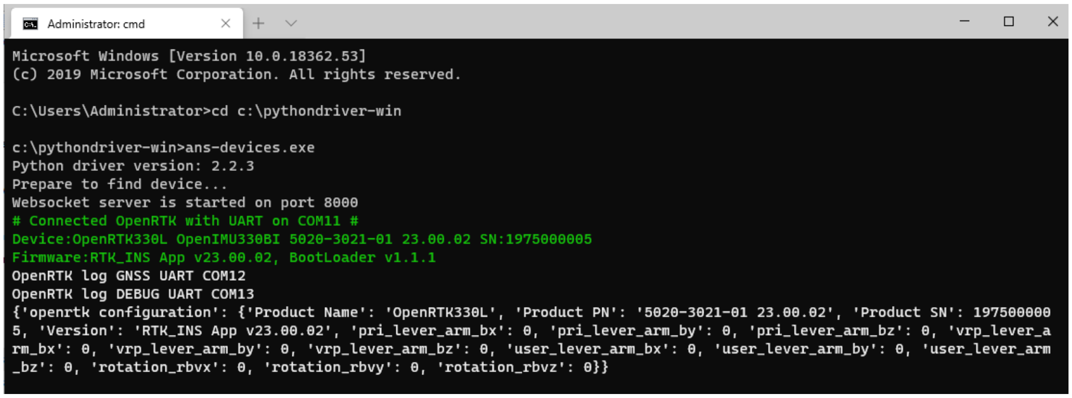
- On the left side menu bar, click "Device Info" tab to have the detailed device information displayed, including firmware version, product number and serial number etc..



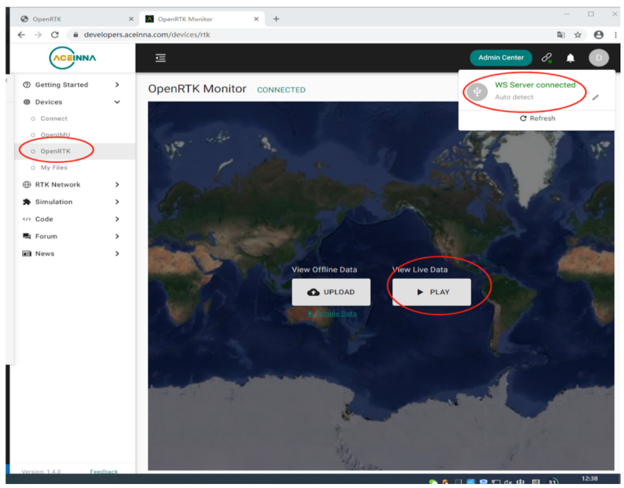
5. **Live Web GUI**: download the latest Python driver executable (v2.2.4 and later), and run it in a command line, for example:

```
cd c:\pythondriver-win
.\ans-devices.exe
```

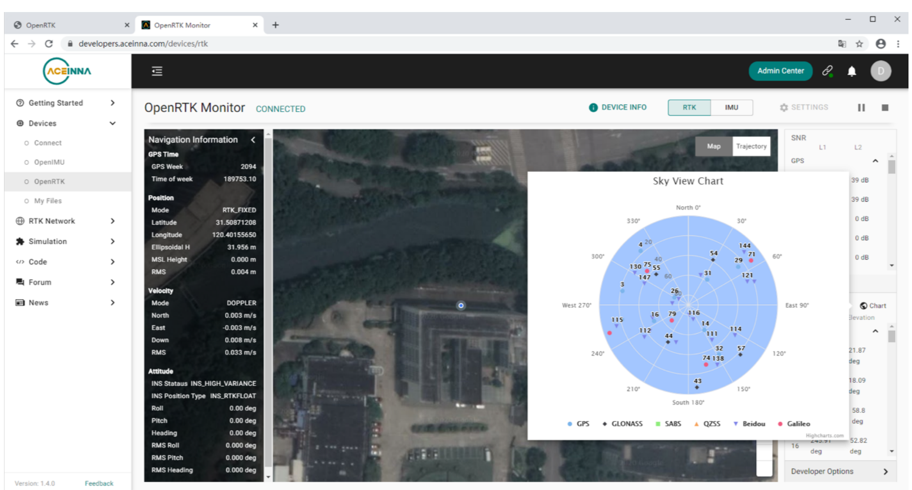
- Check the console output, the Python driver connects the device and the online ANS website, if successfully, the following connection information is displayed



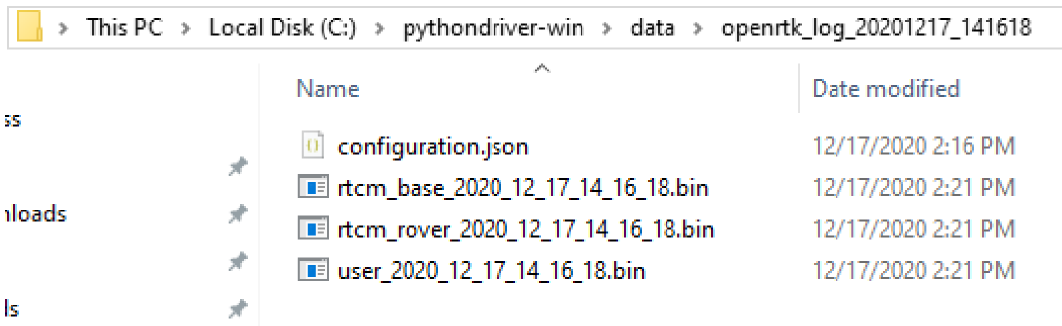
- Go to the online ANS, on the left side menu bar, click "Devices"->"OpenRTK", then we will have the "OpenRTK Monitor" webpage as shown below, and the center "Play" button is highlighted indicating correct device connection with the Web GUI,



- Click "Play", you will have a live web GUI showing positioning information, map presentation and associated satellites information



6. **Data Logging and Parsing**: when the device is connected with the PC via the micro-USB cable, the running Python driver is logging all serial port output into files, including raw GNSS/IMU data, positioning solution and the device configuration. These files are located in a subfolder labelled ".pythondriver-windataopenrtk_log_xxxxxxxx_xxxxxx", e.g.



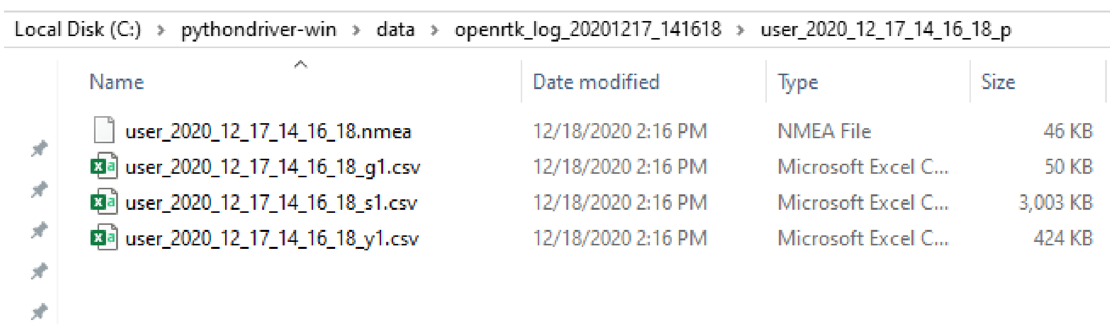
which,

- configuration.json: is the device configuration information
- rtcm_base_xxxx_xx_xx_xx_xx_xx.bin: is the received GNSS RTK correction data through internet, in RTCM format
- rtcm_rover_xxxx_xx_xx_xx_xx_xx.bin: is the GNSS raw data from the device, in RTCM format
- user_xxxx_xx_xx_xx_xx_xx.bin: is the output from the USER UART, including NMEA0183 messages in ASCII format, raw IMU data and GNSS RTK/INS solution in binary format

Go to the "openrtk_data_parse" subfolder, run the parser executable as below

```
cd c:\pythondriver-win\
.\ans-devices.exe parse -t openrtk -p ..\data\openrtk_log_20201217_141618
```

A subfolder with the name "user_xxxx_xx_xx_xx_xx_xx_p" is created and contains the decoded files all in ASCII format, e.g.



which:

- user_xxxx_xx_xx_xx_xx_xx.nmea: contains the GGA and RMC NMEA0183 messages
- user_xxxx_xx_xx_xx_xx_xx_g1.csv: is the GNSS RTK solution
- user_xxxx_xx_xx_xx_xx_xx_s1.csv: is the raw IMU data
- user_xxxx_xx_xx_xx_xx_xx_y1.csv: is the GNSS satellites information that are used in the solution

2.3 Note

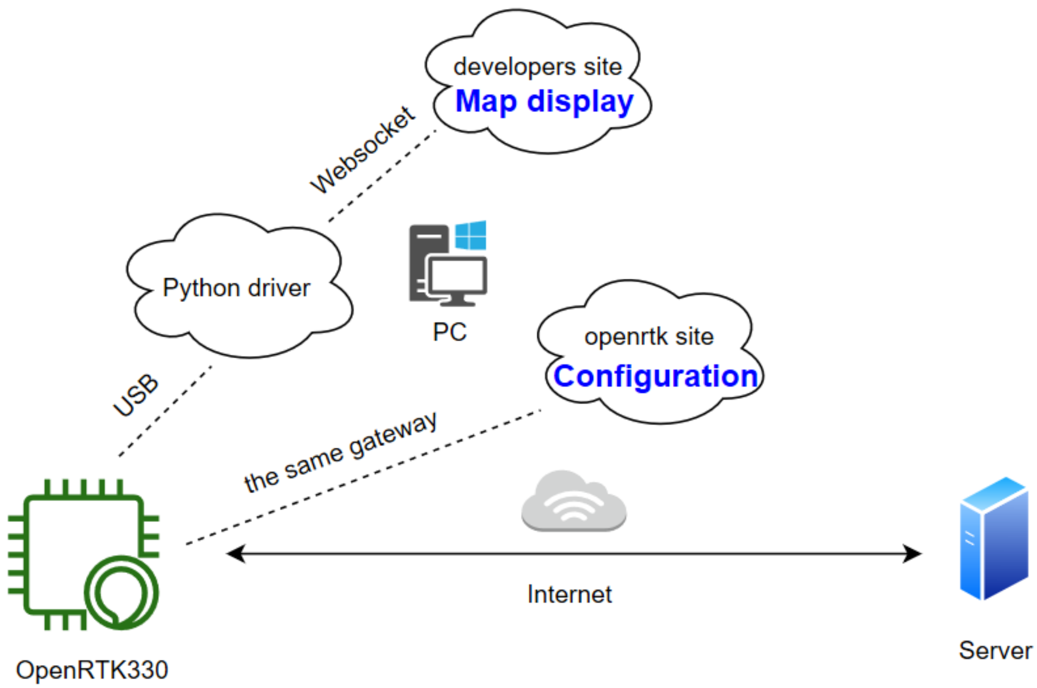
This section presents a brief introduction and quick start on using OpenRTK330LI EVK for RTK and INS positioning. Please refer to the remaining sections of this tutorial chapter to explore more on OpenRTK330LI's features and its usage.

CHAPTER 3

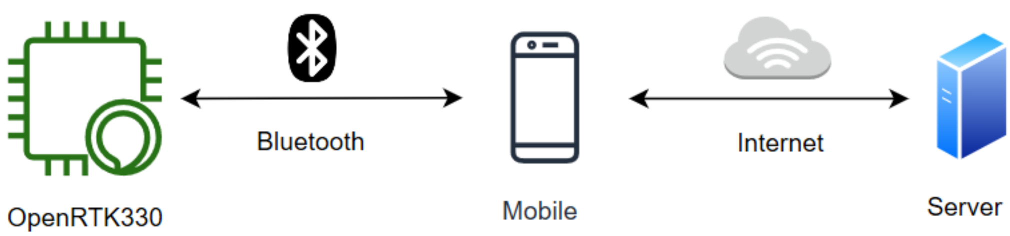
How to Use OpenRTK330 EVK?

Note the usage of OpenRTK330 is described with the OpenRTK330 EVK. There are two types of user APP provided to interact with both the module and a NTRIP server over the internet providing GNSS correction data for RTK positioning:

- PC: using the Ethernet interface
  - Ethernet connectivity between module and NTRIP server with a lightweight TCP/IP stack embedded in firmware
  - Module settings on positioning parameters and user configuration with a web GUI embedded in firmware
  - Map and positioning information display on the online web GUI ("OpenRTK Monitor") of Aceinna developer website



- Android: the "OpenRTK" Android App with the following contents
  - Bluetooth connectivity to module
  - 4G connectivity to NTRIP server
  - Map display with user trajectory and positioning infromation
  - Module settings on positioning parameters and user configuration



The following two subsections cover the detailed steps of using the two types of user App.

3.1 With a PC

Using the OpenRTK330L EVK to evaluate the OpenRTK330L product with a PC requires

- access to the online web based Aceinna Navigation Studio (ANS) via the Micro-USB connection between the EVB and the PC
- access to NTRIP server over the internet via a Ethernet connection between the EVB and the PC

3.1.1 Usage

1. **Power and data link**

   Connect the EVB with a PC using a Micro-USB cable, and the **YELLOW** LED (#12 on the EVB figure above) flashes. The EVB is powered on, and four serial ports are established on the PC.

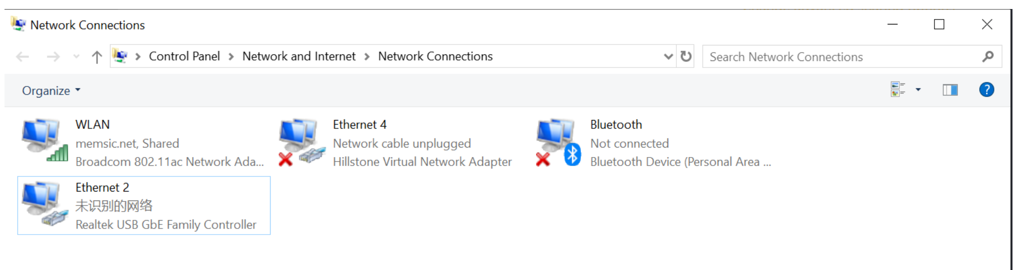
2. **Antenna**

   Connect a GNSS multi-frequency antenna to the SMA interface (#2 on the EVB figure), the **GREEN** LED (#12 on the EVB figure above) flashes if the incoming GNSS signal is valid

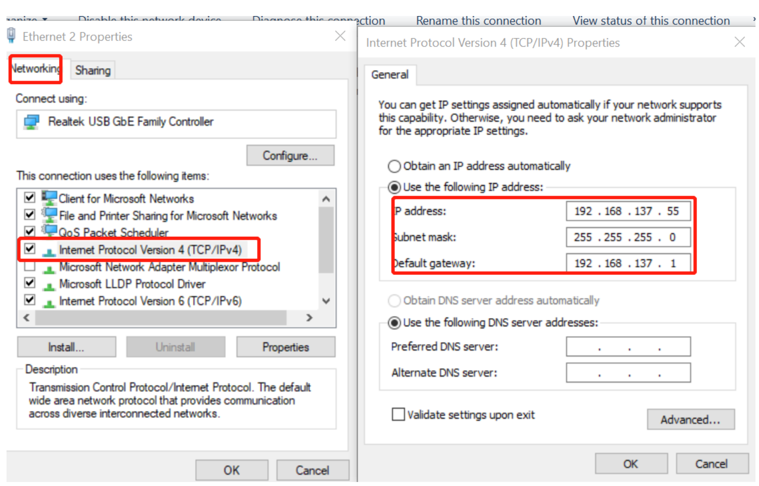
3. **Network**

   There are two ways of for the OpenRTK330LI EVB gets access to internet:

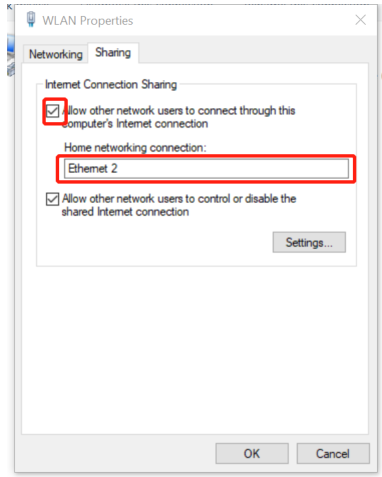
   - Use an Ethernet calbe to connect the EVB with a network router or switch, and then connect a PC to the same router/switch using an Ethernet cable. The OpenRTK330LI EVB gets internet access and assigned an IP address in the local network via DHCP.
   - The other way is using an Ethernet cable to connect the EVB and the PC directly, which requires internet sharing between the PC and the EVB. For example, with a Windows 10 PC,
     - Go to Control PanelNetwork and InternetNetwork Connections, an Ethernet subnetwork is established for the Ethernet connection between the EVB and the PC, e.g. "Ethernet 2" as shown below.



     - Right-click "Ethernet 2", and then click "Properties", on the "Networking" tab, click "Internet Protocol Version 4 (TCP/IPv4)", configure the IP settings as follows: the gateway has to be 192.168.137.1, and the subnet mask has to be 255.255.255.0, while the IP address can be assigned to one that has not been taken in the network 192.168.137.xx.



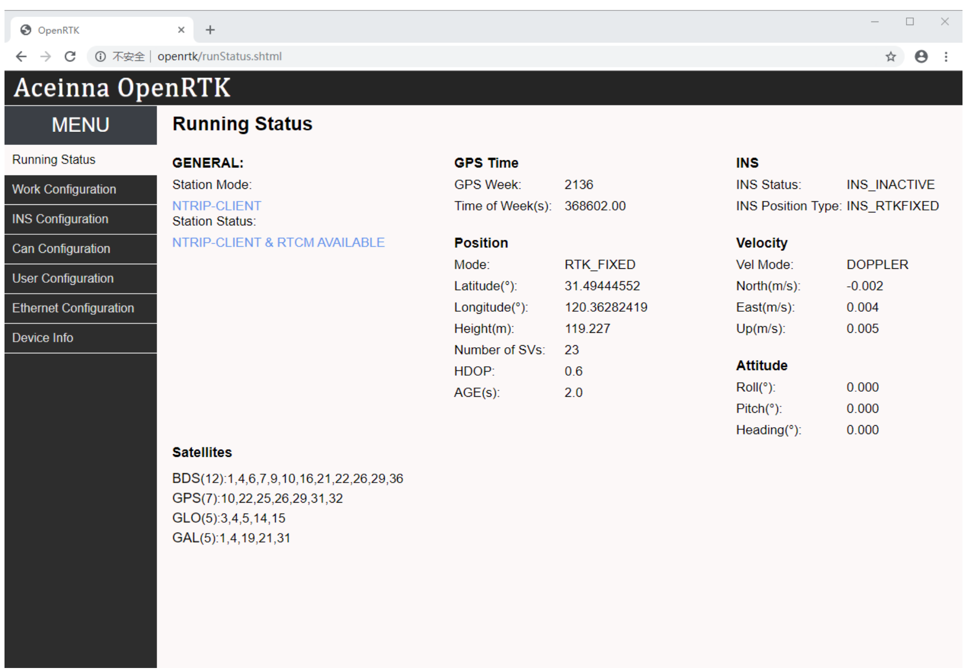
     - Then, right-click WLAN (assuming the PC uses WiFi for internet access), go to Properties->Sharing, check the "Allow other network users to connect through this computer's internet connection", and select "Ethernet 2" on the drop down menu below, click "OK" to enable the EVB to have access to internet shared by the PC.



4. **Device Configuration on the Embedded Web Pages**

   A lightweight TCP/IP web service is embedded inside the OpenRTK330 firmware, user can access the device configurations on the embedded web pages when the device and the PC are in the same local Ethernet network (connected to the same router or direclty connected). Open a browser (Google Chrome is recommended), visit http://openrtk,

   - You will firstly see the following device running status page



   Besides the positioning information, this web page displays the working mode and status of the device on the most left-upper conner,

     – *Station Mode*, has the following two values:

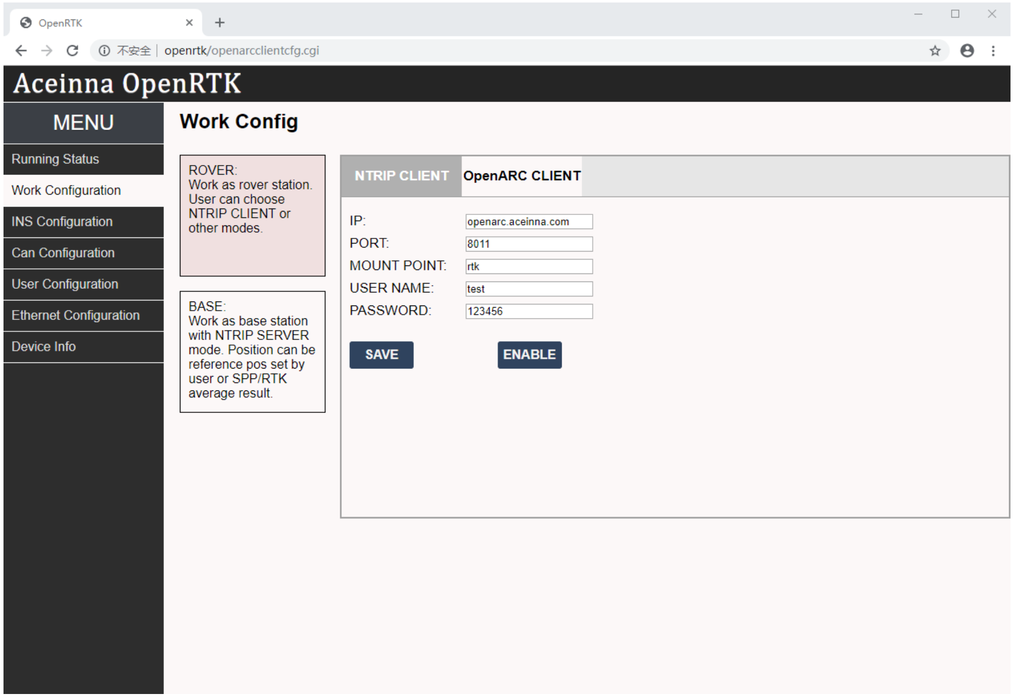
       * NTRIP-CLIENT: the device works as a NTRIP client (Rover), and is used as a positioning/navigation equipment

    * NTRIP-SERVER: the device works as a NTRIP server (Base), and broadcasts its GNSS data to NTRIP clients for differential GNSS operation

  and the mode changes when the user configures the device differently in the "Work Configuration" tab.

  - *Station Status*, has the following thirteen values:
    * "Waiting…": waiting for changes to take effective
    * "NTRIP-CLIENT & CONNECT…": trying to connect with a NTRIP server
    * "NTRIP-CLIENT & CONNECT FAIL": failed to connect with a NTRIP server
    * "NTRIP-CLIENT & CONNECTED": connect with a NTRIP server successfully, waiting for GNSS correction data
    * "NTRIP-CLIENT & RTCM AVAILABLE": received GNSS correction data successfully from the NTRIP server
    * "NTRIP-SERVER & CONNECT…": trying to connect with a NTRIP caster
    * "NTRIP-SERVER & CONNECT FAIL": failed to connect with a NTRIP caster
    * "NTRIP-SERVER & CONNECTED": connect with a NTRIP caster successfully, waiting to output GNSS correction data
    * "NTRIP-SERVER & RTCM OUTPUT": outputting GNSS correction data
    * "OpenARC CONNECT…": trying to connect with Aceinna's OpenARC cloud service (e.g. NTRIP server and data service)
    * "OpenARC CONNECT FAIL": failed to connect with Aceinna's OpenARC cloud service
    * "OpenARC CONNECTED": connect with a Aceinna's OpenARC cloud service successfully, waiting for GNSS correction data
    * "OpenARC RTCM AVAILABLE": received GNSS correction data successfully from Aceinna's OpenARC cloud service

- On the left side menu bar, click "Work Configuration" tab to choose the following working mode of the device and configure it accordingly:
  - Rover: works as a nomarl GNSS positioning device that is also referring to "NTRIP client", and receives GNSS data correction from a NTRIP server that has to be configured with the following information, as shown by the "OpenARC Client" tab below
    * *IP: openarc.aceinna.com*
    * *PORT: 8011*
    * *Mount Point: RTK*
    * *User Name: username*
    * *Password: password*



OpenARC is a cloud service provided by Aceinna for users in the United States to receive nation-wide GNSS correction data for RTK operation, without the need to set up a local GNSS base station. More details refer to the section "OpenARC Service" (click here) in this tutorial.

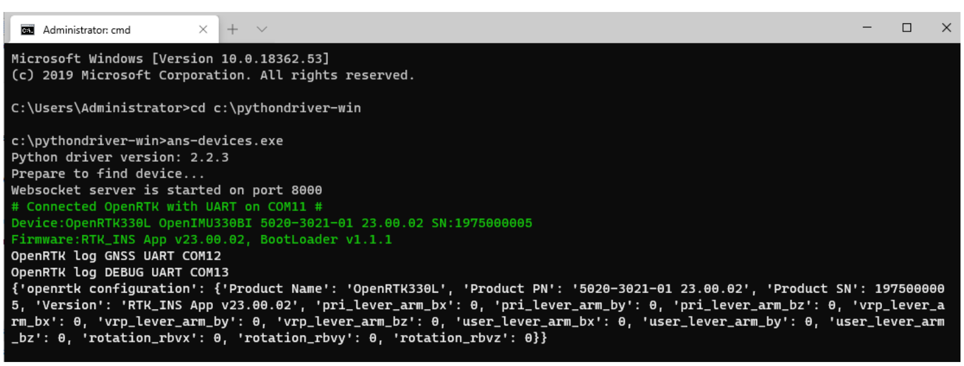
– Base: works as a GNSS reference station with known position and sending GNSS data to "NTRIP server" to be used as GNSS data correction

5. **Live Web GUI on the Online ANS Website**

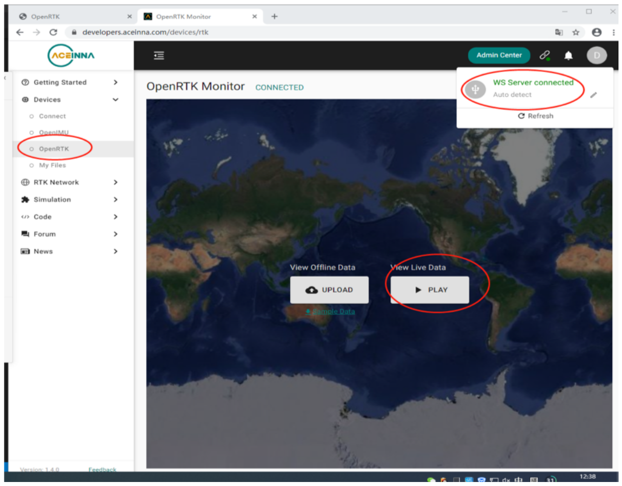
   Download (click here) the latest Python driver executable (v2.2.4 and later), and run it in a command line, for example:

```
cd c:\pythondriver-win
.\ans-devices.exe
```

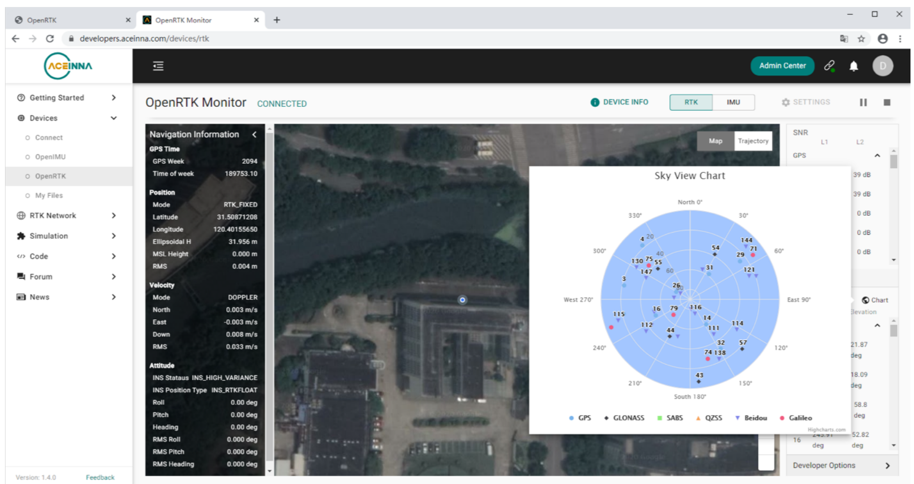
- Check the console output, the Python driver connects the device and the online ANS website, if successfully, the following connection information is displayed



- Go to the online ANS, on the left side menu bar, click "Devices"->"OpenRTK", then we will have the "OpenRTK Monitor" webpage as shown below, and the center "Play" button is highlighted indicating correct device connection with the Web GUI,



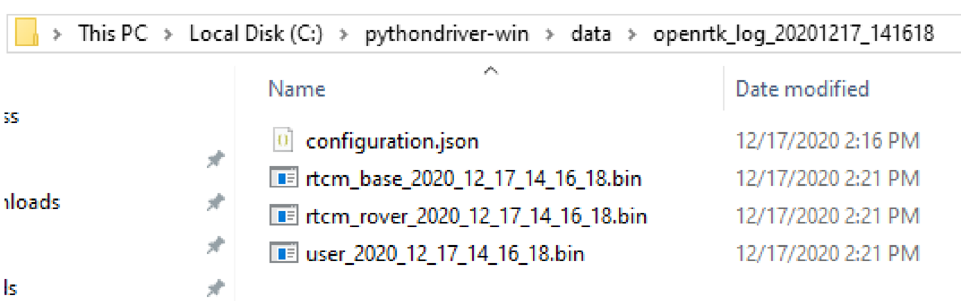
- Click “Play”, you will have a live web GUI showing positioning information, map presentation and associated satellites information



6. **Data Logging and Parsing on a PC**

- With the UART/Serial port

When the device is connected with the PC via the micro-USB cable, the running Python driver is logging all serial port output into files, including raw GNSS/IMU data, positioning solution and the device configuration. These files are located in a subfolder labelled “.pythondriver-windataopenrtk_log_xxxxxxxx_xxxxxx”, e.g.



which,

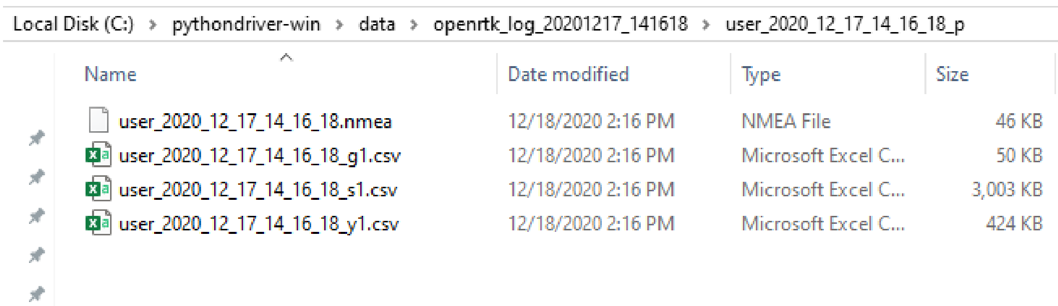
- configuration.json: is the device configuration information

- rtcm_base_xxxx_xx_xx_xx_xx_xx.bin: is the received GNSS RTK correction data through internet, in RTCM format
- rtcm_rover_xxxx_xx_xx_xx_xx_xx.bin: is the GNSS raw data from the device, in RTCM format
- user_xxxx_xx_xx_xx_xx_xx.bin: is the output from the USER UART, including NMEA0183 messages in ASCII format, raw IMU data and GNSS RTK/INS solution in binary format

Go to the "openrtk_data_parse" subfolder, run the parser executable as below

```
cd c:\pythondriver-win\
.\ans-devices.exe parse -t openrtk -p ..\data\openrtk_log_20201217_141618
```

A subfolder with the name "user_xxxx_xx_xx_xx_xx_xx_p" is created and contains the decoded files all in ASCII format, e.g.



which:

- user_xxxx_xx_xx_xx_xx_xx.nmea: contains the GGA and RMC NMEA0183 messages
- user_xxxx_xx_xx_xx_xx_xx_g1.csv: is the GNSS RTK solution
- user_xxxx_xx_xx_xx_xx_xx_s1.csv: is the raw IMU data
- user_xxxx_xx_xx_xx_xx_xx_y1.csv: is the GNSS satellites information that are used in the solution

• With the CAN Interface

User could use a CAN-USB (e.g. https://canable.io/) or CAN-TTL adapter to connect with the DB-9 male interface on the EVB to log and parse the CAN messages (click here for definitions). Note that user has to write their own CAN message parsing code using the provided lib or open-source code from the adapter provider.

3.2 With an Android Smartphone

Using the OpenRTK330L EVK to evaluate the module requires

- the installation the "OpenRTK" Android App: provides 4G access to NTRIP server over the internet
- Micro-USB connection to a PC for power and data logging connection

3.2.1 "OpenRTK" App Installation

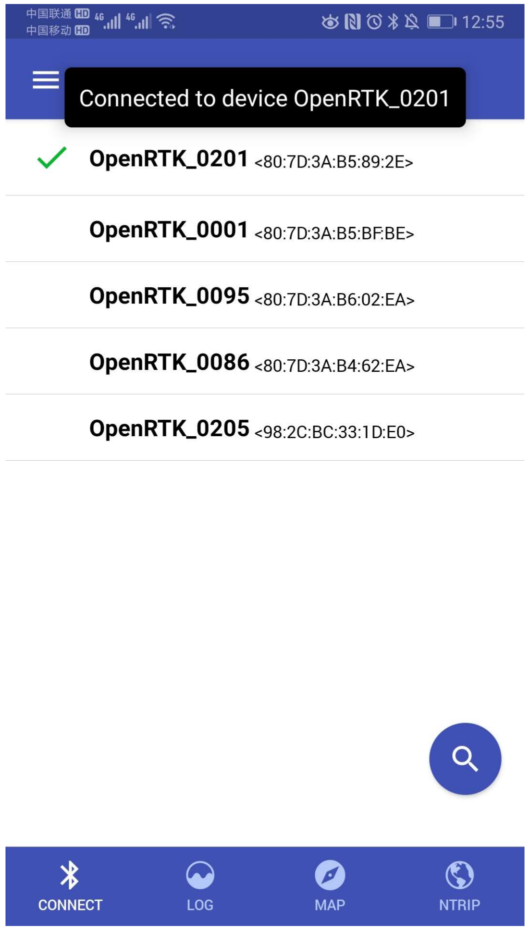
1. Scan the QR code below or click here to download the Android apk installation file. Make sure your Android version is 8.0 or above.

2. Open the downloaded APK file to install the App

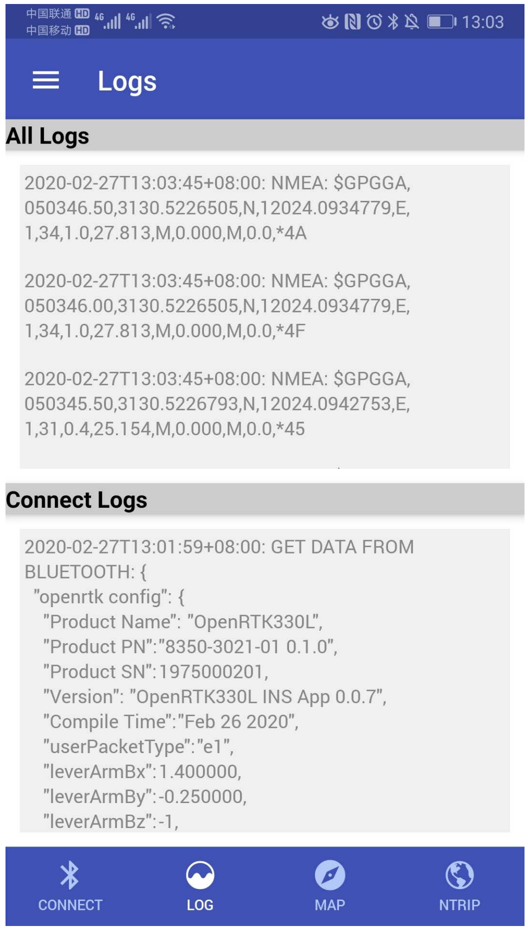
Note: Please grant the OpenRTK App access to run in Android system's backend.

3.2.2 Usage Steps

1. **Connection**
    - Connect the OpenRTK330 EVB to a PC via a Micro-USB cable, then connect the EVB with a GNSS antenna, checking the LED lights for working status
    - YELLOW: flashing light indicating GNSS chipsets is powered on with valid 1PPS signal output
    - GREEN: flashing light indicating OpenRTK330L INS App is running correctly with valid GNSS signal receiving
    - Enable "Bluetooth" function and "Location" access right for "OpenRTK" App on your Anroid device
    - Open the "OpenRTK" Andorid App, as shown by the picture below, go to the "Connect" tab and click the "search" icon (right bottom) to search for your device. If your OpenRTK330 device is found, a Bluetooth device ID appears on the "Connect" list. By factory setting, the Bluetooth device ID is "OpenRTK_<four digits>" and the four digits are the last four digits of your OpenRTK330 module S/N. Click your Bluetooth device ID and if connected successfully, a notification appears

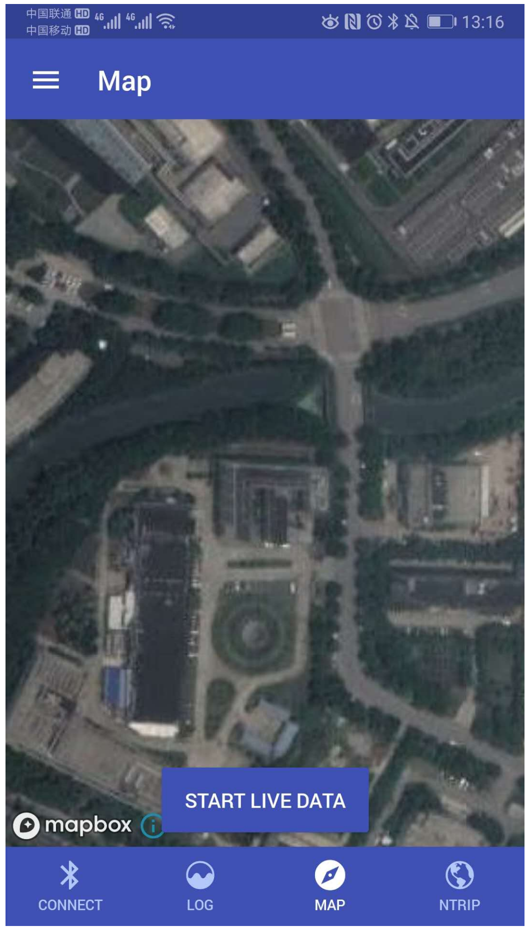


- Besides, detailed Bluetooth connection and user configuration information of the device can be found on the lower window of the "Log" tab, and NMEA GGA messages are reporting the OpenRTK330 device position on the upper window of the "Log" tab.

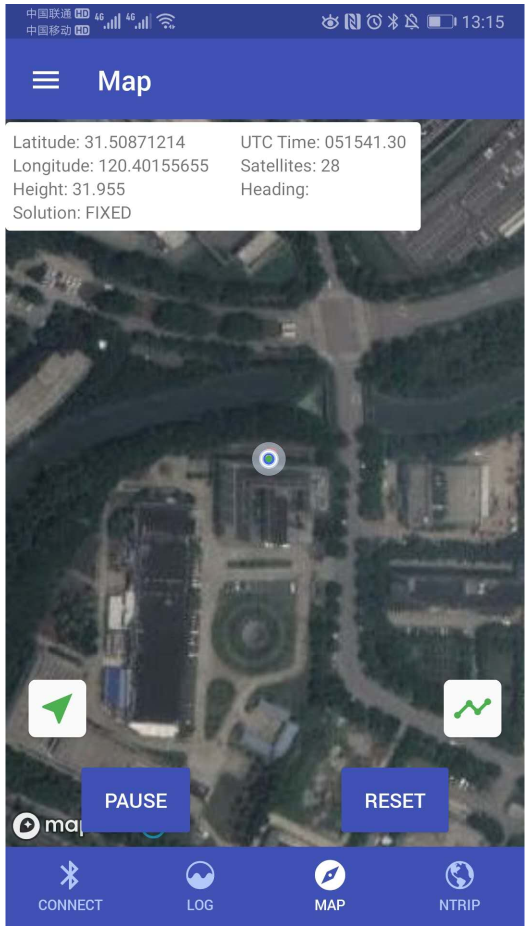


2. **Map Presentation**

   - Once the Bluetooth connection made successfully, and OpenRTK330 is reporting positioning information to Android App, go to "Map" tab and click "Start Live Data" to start a live map presentation

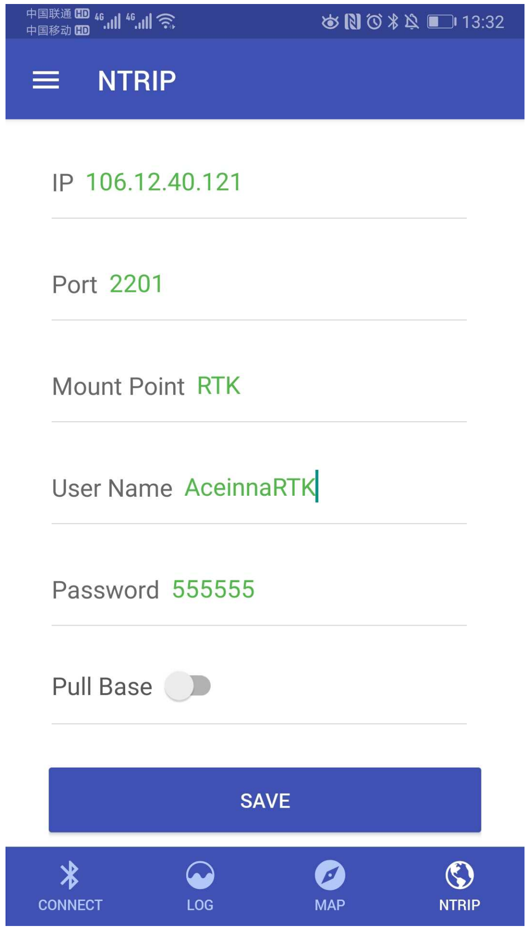


- Real time positioning information and trajectory is shown

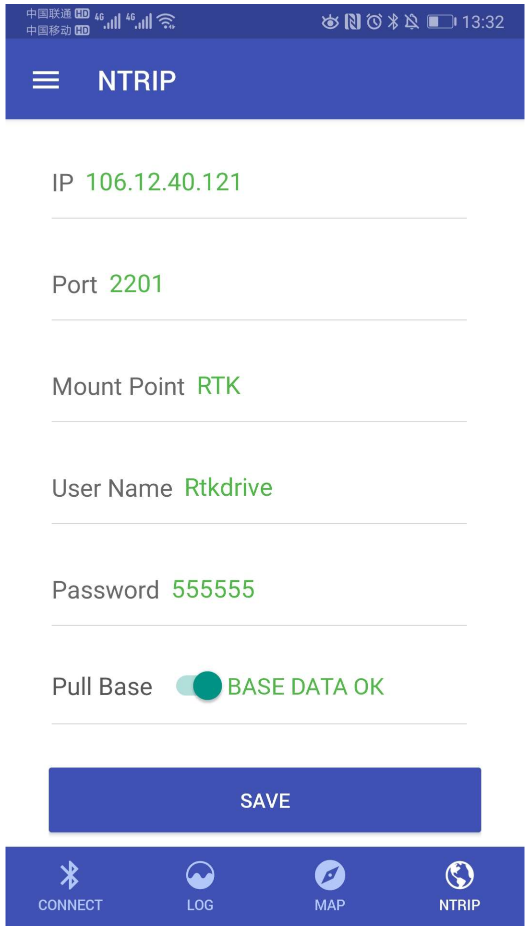


3. **NTRIP Configuration**

- In order to get GNSS RTK positioning, go to "NTRIP" tab and configure the NTRIP server settings of your GNSS correction data provider

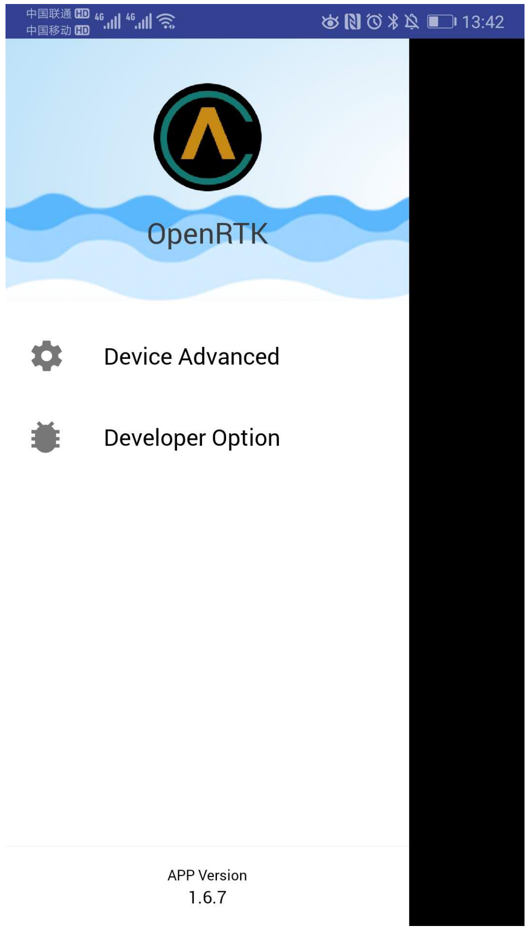


- Click "SAVE" to save your NTRIP server settings to your OpenRTK330 module, and then switch on "Pull Base" to get GNSS correction data for RTK

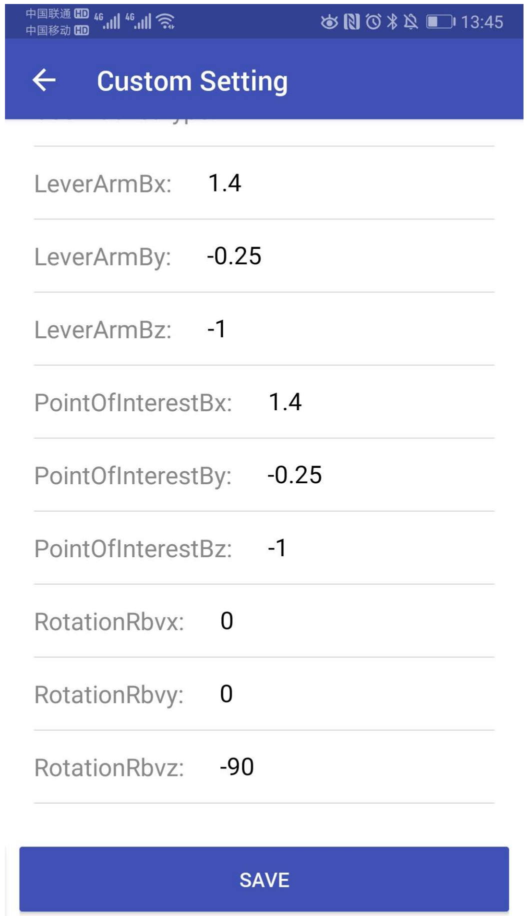


4. **User Configuration**.

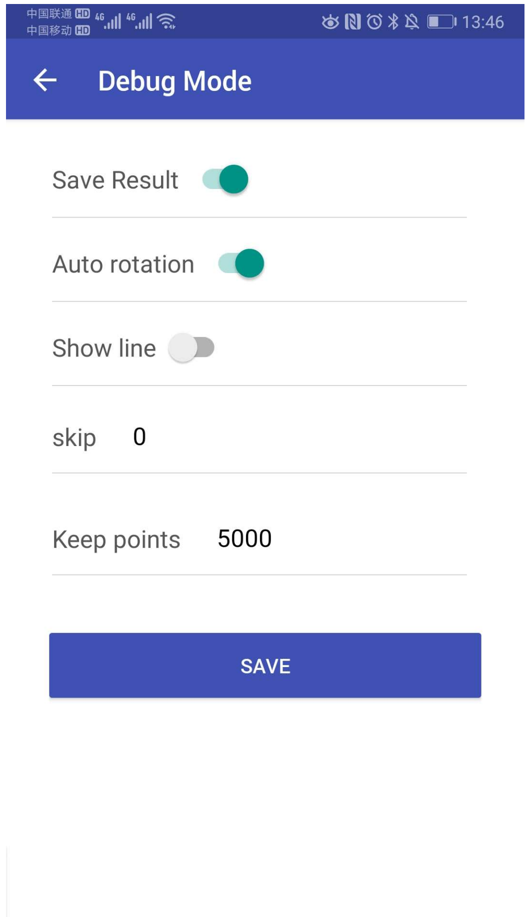
   From anyone of the four tabs, you can access the menu for user configuration by clicking the icon "" at the upper left corner



- Click “Device Advanced”: user can change and save OpenRTK330 device settings, like Bluetooth ID, lever arm and so on.



- Click "Developer Option": user can configure the Android App on map presentation and switch on/off of saving positioning results (NMEA GGA messages only) to Android phone storage. The defualt storage path is "Android/data/com.aceinna.rtk/files/log"



5. **Data Logging**

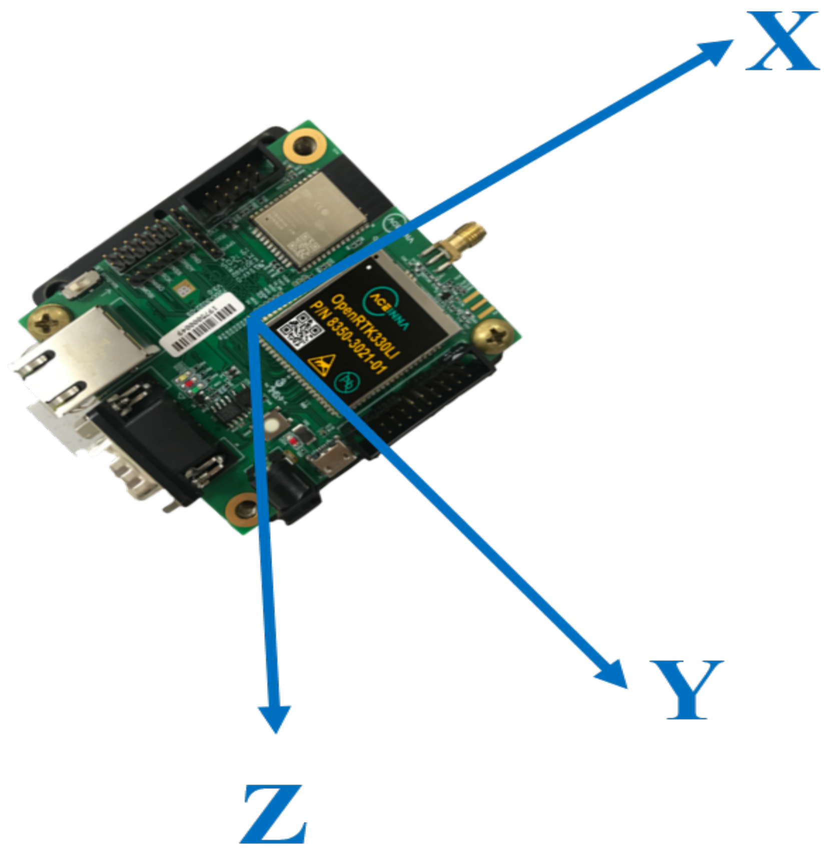
   In the mobile use case, user still needs a PC to log the device output data into files, the OpenRTK Android app doesnot log data on the phone. Refer to the previous section "With a PC" for data logging details.

3.3 EVK Vehicle Installation

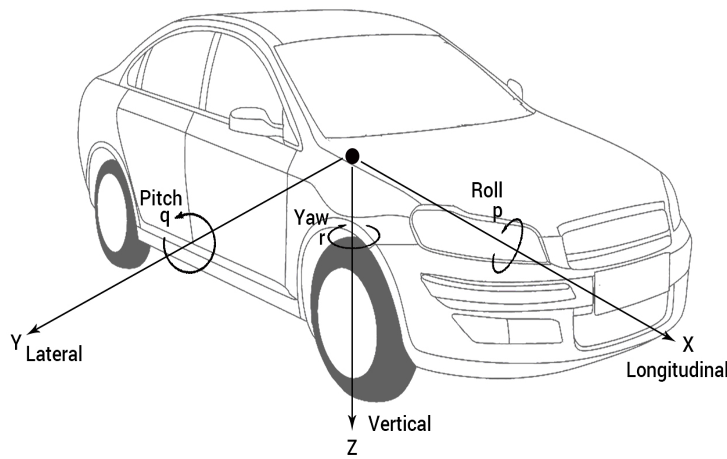
3.3.1 Reference coordinate frames

In order to install the OpenRTK330 EVB on vehicle for driving test, a few reference frames listed below has to be defined

- **The IMU body frame** is defined as below and shown in the figure. By default the INS solution of OpenRTK330 is provided at the center of navigation of the IMU (refer to the mechanical drawing for accurate IMU navigation center position on the EVB).
- x-axis: points to the same direction as the SMA antenna interface
- z-axis: perpendicular to x-axis and points downward
- y-axis: points to the side of the EVK and completes a right-handed coordinate system
- **The vehicle frame** is defined as


X
Y
Z
OpenRTK330LI
P/N 8350-3021-01

  - x-axis: points out the front of the vehicle in the driving direction
  - z-axis: points down to the ground
  - y-axis: completes the right-handed system

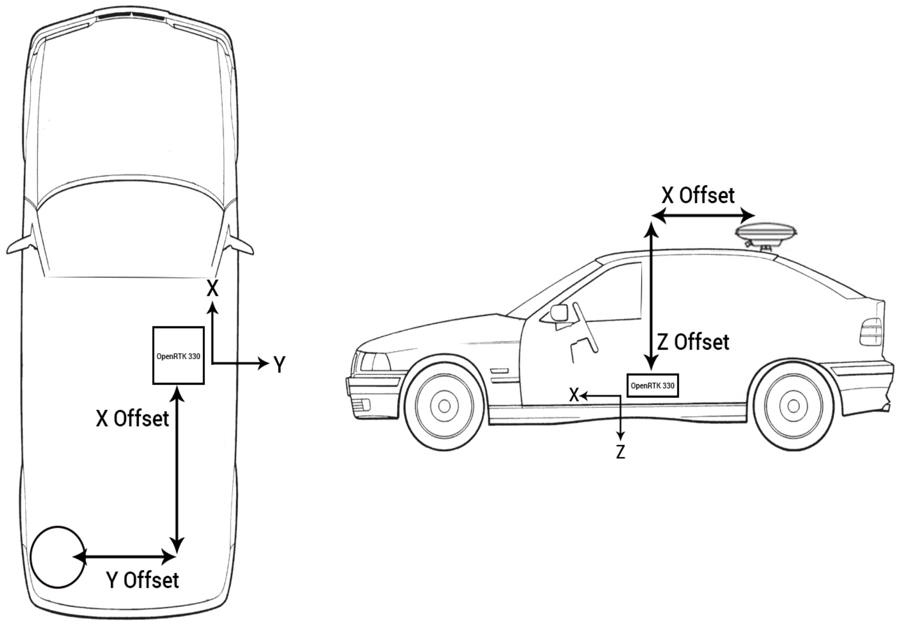


- **The local level navigation frame** is defined as
  - x-axis: points north
  - z-axis: points down parallel with local gravity
  - y-axis: points east
- **The user output frame** is used to transfer the INS solution to a user designated position.

3.3.2 Installation Parameters

Depends on the vehicle installation of the OpenRTK330 system, user has to configure two types of offsets to make the GNSS integrated INS solution work

- Translation offset
  - *GNSS antenna lever-arm*: GNSS position is estimated to the phase center of the GNSS antenna, and INS position is estimated to the center of the navigation of the IMU. The translation from the IMU center to the phase center of the GNSS antenna has to be known and applied to the integrated system via user configuration of the antenna lever-arm. The GNSS/INS integrated solution outputs position at the IMU center. For example, the lever arm in the figure below is [x, y, z] = [-1.0, -1.0, -1.0] meter.



- *User output lever-arm*: If user wants the above GNSS/INS integrated solution output at a more useful position, the translation between the IMU center and the designated point of interest has to be known and applied via user configuration of point of interest lever-arm.

- Rotation offset: If the axes of the IMU body frame of the installed OpenRTK330 unit is not aligned with the vehicle frame, the orientation of the IMU relative to the vehicle also has to be known and applied via user configuration of rotation angles between the IMU body frame and vehicle frame. For example, given a installation setup as shown by the following figure

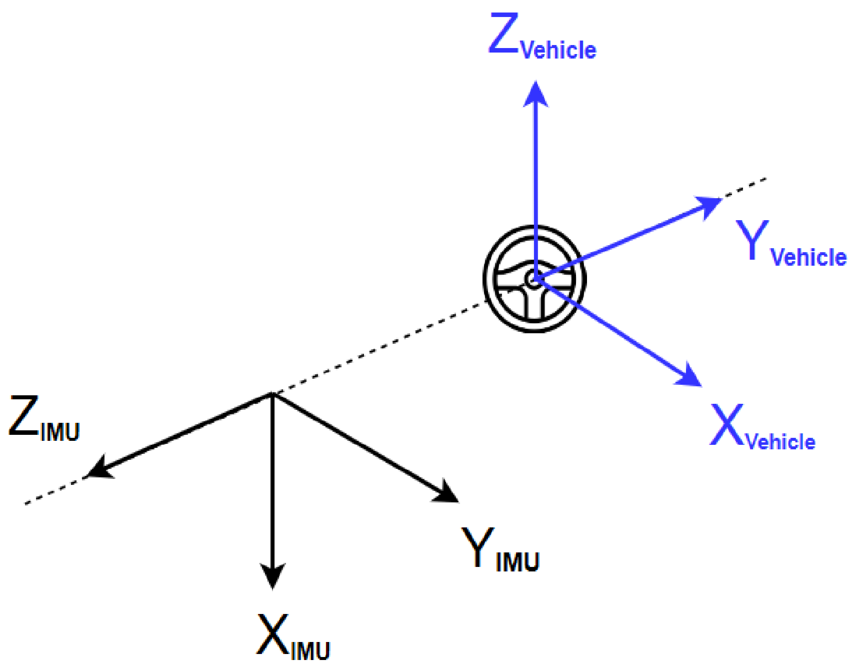
  We have to mathematically rotate the IMU body frame to align with the vehicle frame, in the following order:

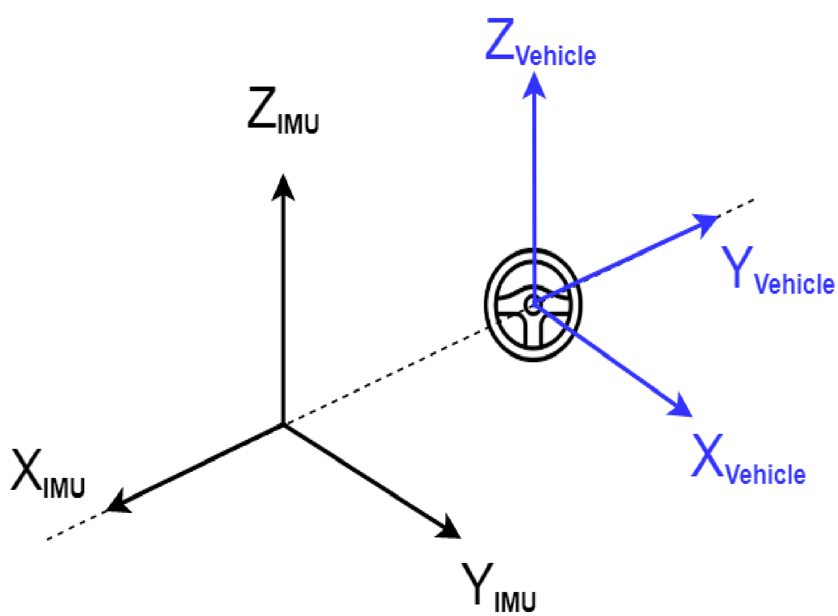
  1. Rotate IMU cooridnate frame to get z-axis aligned
  2. Rotate IMU cooridnate frame to get x-axis aligned
  3. Rotate IMU cooridnate frame to get y-axis aligned

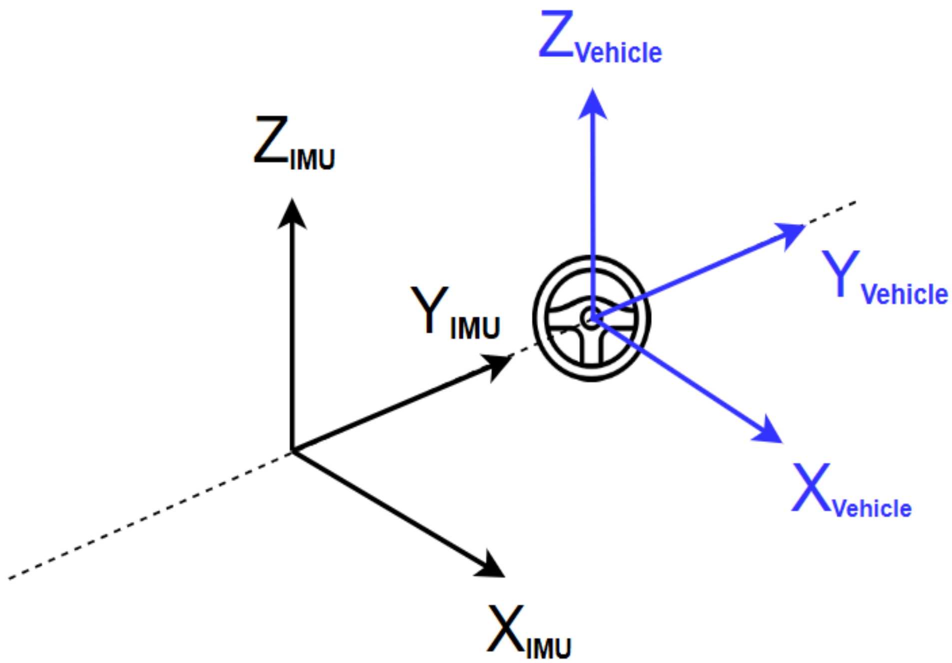
  For the example above, firstly rotate 90 degrees clockwise along IMU y-axis to align z-axis of two frames,

  Then rotate 90 degrees counter-clockwise along IMU z-axis to align x-axis of two frames.

  The final rotation matrix angles that user has to configure are [x, y, z] = [0, -90, 90] degrees.


Z Vehicle
Y Vehicle
X Vehicle
Z IMU
Y IMU
X IMU


Z Vehicle
Z IMU
Y Vehicle
X Vehicle
X IMU
Y IMU



3.3.3 Odometer Input from Vehicle

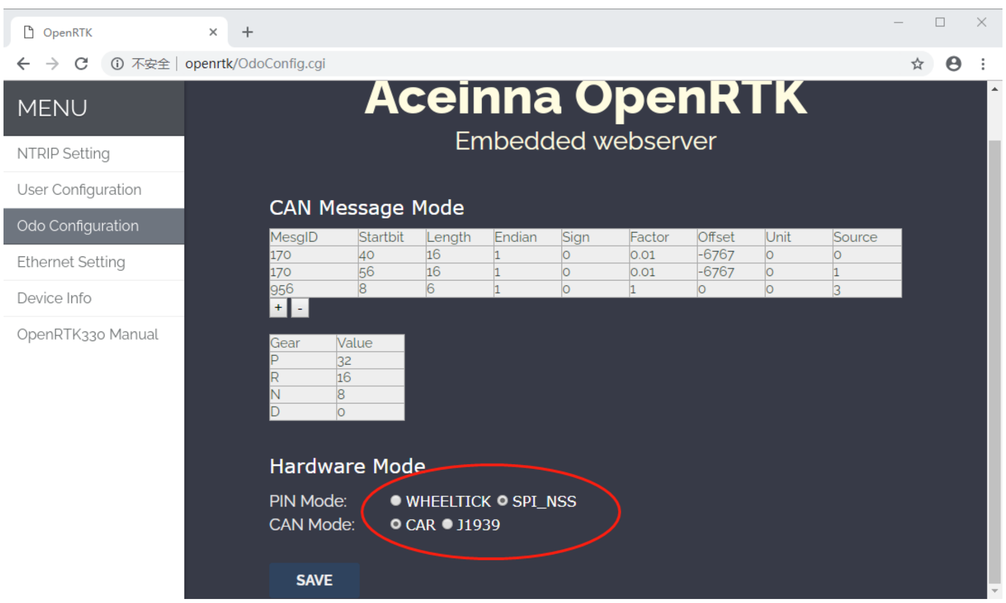
To fully explore the dead reckoning (DR) for vehicular positioning, OpenRTK330LI EVK has the following three options to get the Odometer data input from the vehicle:

- CAN interface
- wheel-tick signal and FWD (i.e. forward) signal
- USER UART input message

CAN interface

User is recommended to use a OBDII-CAN cable to connect the EVB DB-9 interface with one OBDII interface on the vehicle, the following photos show an example

The CAN message contains vehicle Odometer speed data is different among manufacturers, OpenRTK330LI EVK provides user configuration on the internal Web interface (https://openrtk) to accommondate the different input CAN messages, as shown below



User has to check the "CAR" option for the CAN mode to enable the data input working mode of the CAN interface, as shown in the red circle. In the table above, user input the following fields to configure how the OpenRTK330LI module should parse the incoming Odometer message from CAN bus:

- MesgID: CAN message ID, decimal value
- Startbit: the number of starting bit of the Odometer data
- Length: the Odometer data Length in number of bits
- Endian: 0 - little endian; 1 - big endian
- Sign: 0 - unsigned; 1 - signed

- Factor and Offset: actual Odometer value = (original value + Offset) * Factor
- Unit: 0 - km/h; 1 - mph; 2 - m/s
- Source:
  - 0 - right-rear wheel speed (RR)
  - 1 - left-rear wheel speed (LR)
  - 2 - vehicle speed (combined)
  - 3 - gears: fill-in the gear (P, R, N, and D) value in the table below

There are two options to input the vehicle speed depending on the Odometer CAN messages,

- Configure the source to have RR and LR enalbed to obtain aveaged real wheel speed
- Configure the source to have a single combined vehicle speed

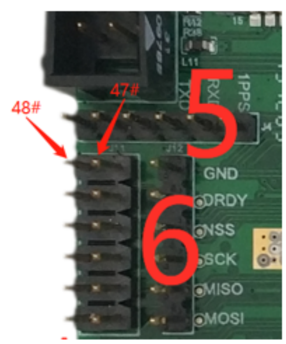
and the first option above is recommendded.

USER UART interface

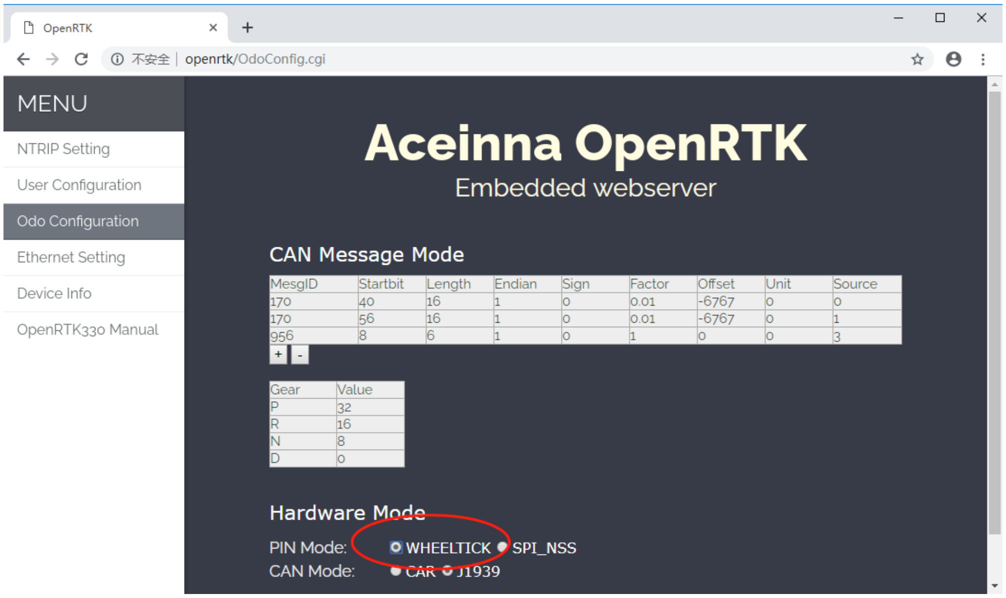
With this approach, user need to extract vehicle speed information from the CAN bus or the wheel speed encoder and send in the real vehicle speed value through the USER UART, using the "cA" packet described in the USER UART data protocol section.

Wheel-tick encoder interface

Another approach to integrate vehicle speed for DR is shown below. A typical aftermarket wheel-tick encoder is shown on the left. Note that OpenRTK330LI EVB currently only supports one wheel-tick encoder input. As shown by the right side photo below, the phase-A and phase-B should connect with the #47 and #48 jumper on the EVB, respectivelly. The input voltage for the pins of OpenRTK330LI EVB is 3.3 v, if the wheel-tick encoder output voltage does not fit, user has to bring in additional voltage conversion circuits or module.



In the current design, the wheel-tick input processing takes over the interrupter of the MCU from the SPI communication ports, thus user needs to choose one of two working mode on the internal web interface page, as shown by the red circle in the figure below


OpenRTK
不安全 | openrtk/OdoConfig.cgi
MENU
NTRIP Setting
User Configuration
Odo Configuration
Ethernet Setting
Device Info
OpenRTK330 Manual
Aceinna OpenRTK
Embedded webserver
CAN Message Mode
MesgID Startbit Length Endian Sign Factor Offset Unit Source
170 40 16 1 0 0.01 -6767 0 0
170 56 16 1 0 0.01 -6767 0 1
956 8 6 1 0 1 0 0 3
Gear Value
P 32
R 16
N 8
D 0
Hardware Mode
PIN Mode: WHEELTICK SPI_NSS
CAN Mode: CAR J1939

CHAPTER 4

Firmware Online Upgrade

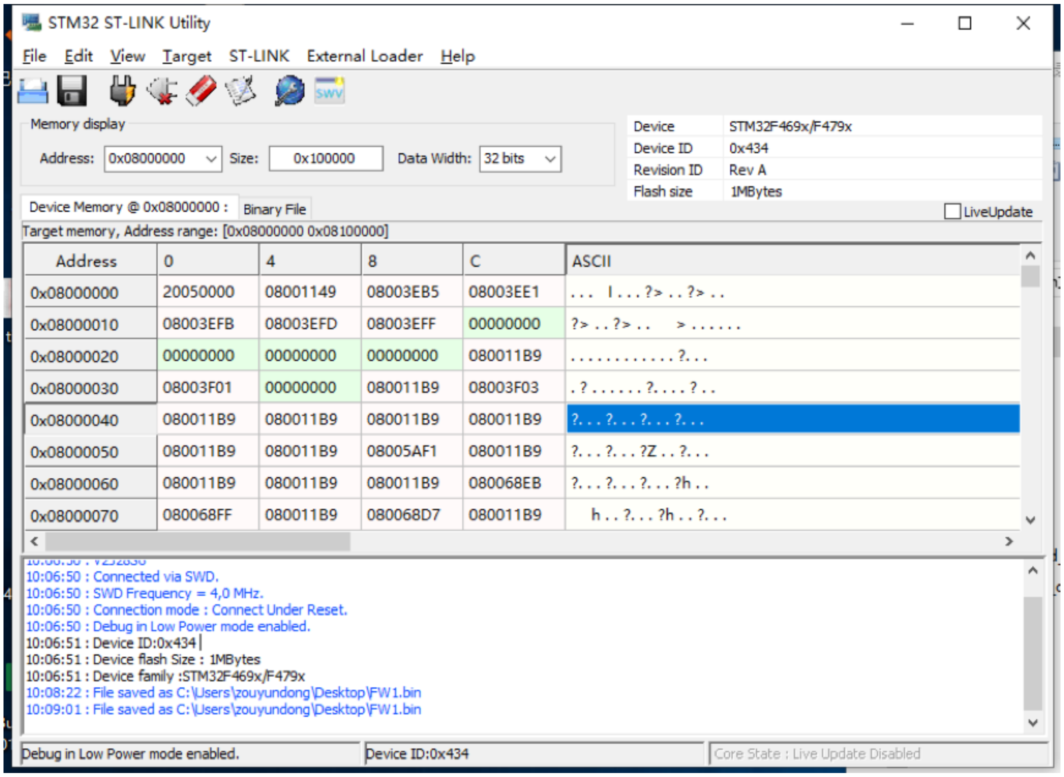
Contents

4.1 WARNING!!!

I. **SAVE BEFORE DEVELOPMENT START**: it's strongly recommended to save your factory OpenRTK330 module system image file to a binary file to be able to recover the whole system if something unexpected happened! Especially, if the system bootloader and IMU calibration tables are damaged, OpenRTK330 will not work properly.

- Save system image
  1. Download and install ST-Link Utility from here
  2. Connect ST-Link debugger between OpenRTK330 EVB and PC and power on the EVB
  3. Open ST-Link Utility software on the PC and go to Target->Connect
  4. Enter value **0x08000000** in *Address* box and **0x100000** in *Size* box as shown by the figure below, then hit enter
  5. Click File->Save As to save the system image file

- Recover system image
  1. Connect ST-Link debugger between OpenRTK330 EVB and PC and power on the EVB
  2. Open ST-Link Utility software on the PC and go to Target->Connect



3. Click File->Open and open previously saved image file
4. Click Target->Program & Verify and make sure that the start address is 0x08000000 before you click Start button to re-programming the OpenRTK330 module

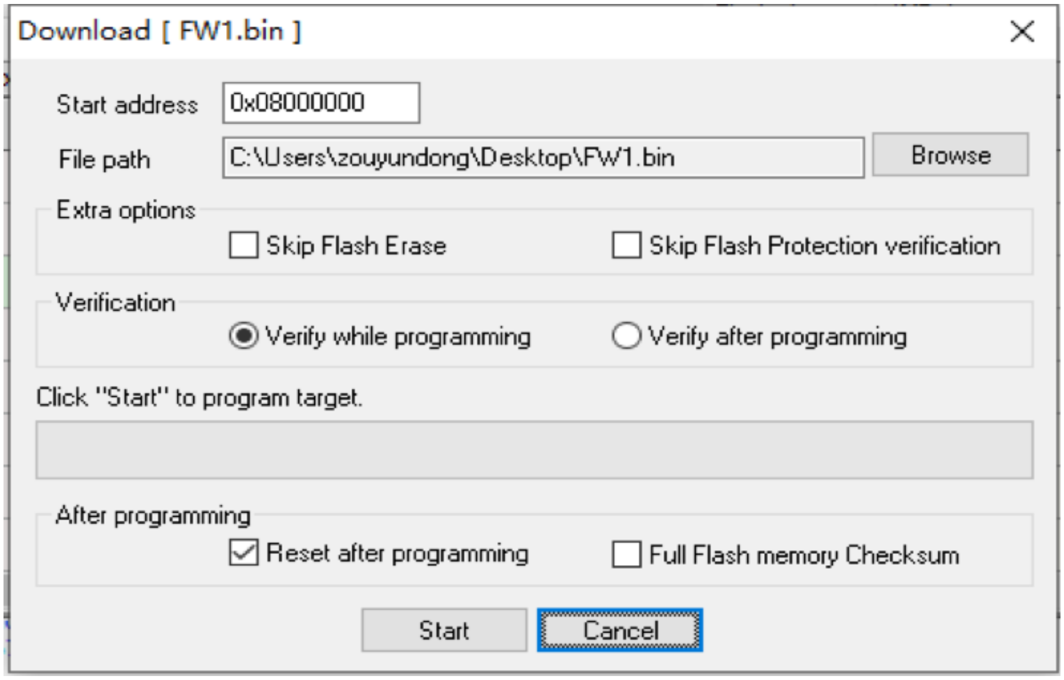
5. Click Target->Option Bytes and select "sector 0", "sector 1", "sector 2", "sector 3" and "sector 11" to perform write protection. Click Apply button for make it effective.

4.2 Firmware Upgrade Online

Work with the online **App Center** of ANS (click here) to **install/update** the OpenRTK330 module **firmware**, as shown by

First, upgrade OpenRTK330LI bootloader (to v1.1.1 and later, Win10 only):

1. Connect ST-LINK debugger between a PC and the EVB
2. Use a Micro-USB cable to connect the PC and the EVB and power on the EVB
3. Download the Bootloader bin file from the App center as shown by the above figure
4. Open ST Utility software, click Target->Connect, then click Target->Program & Verify, on the pop dialog as shown below, load the downloaded bootloader bin file from step 3, check "Verify while programming" and "Reset after programming", click "Start" button



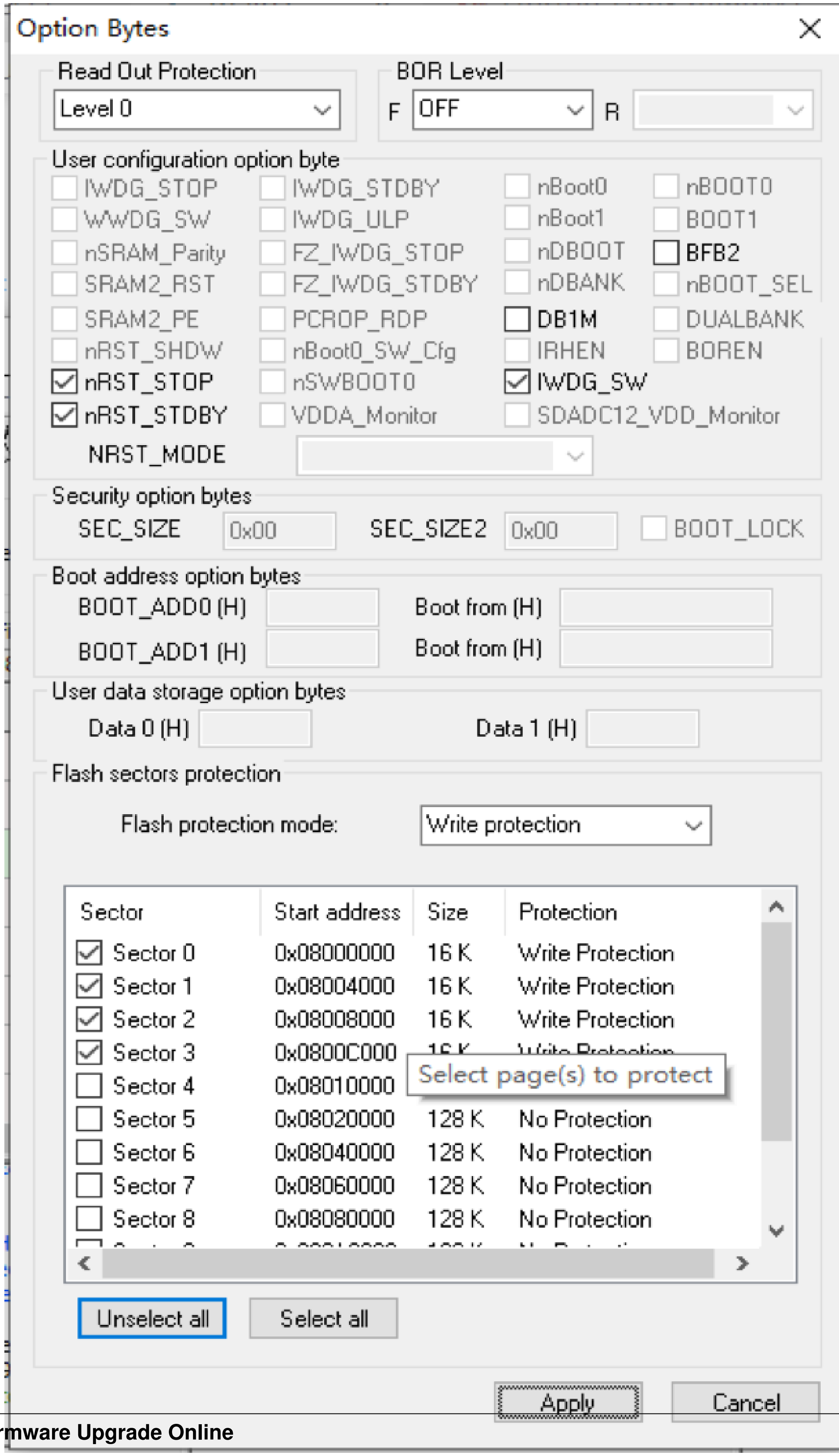
5. Remove ST-LINK debugger from the EVB

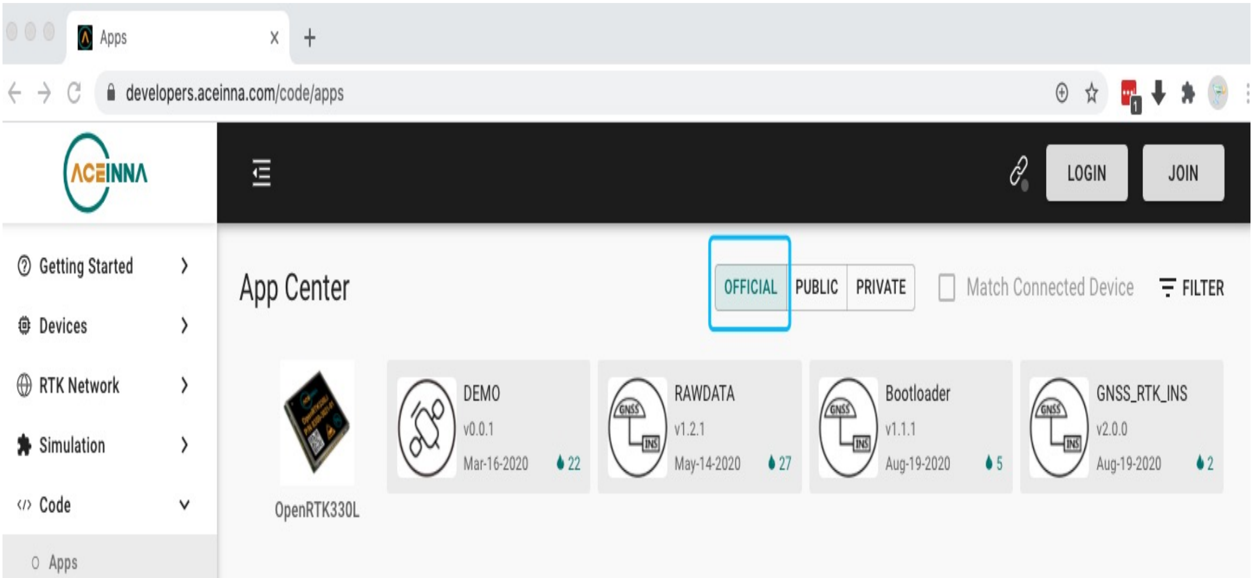
Secondly, follow the steps below to upgrade OpenRTK330 firmware:

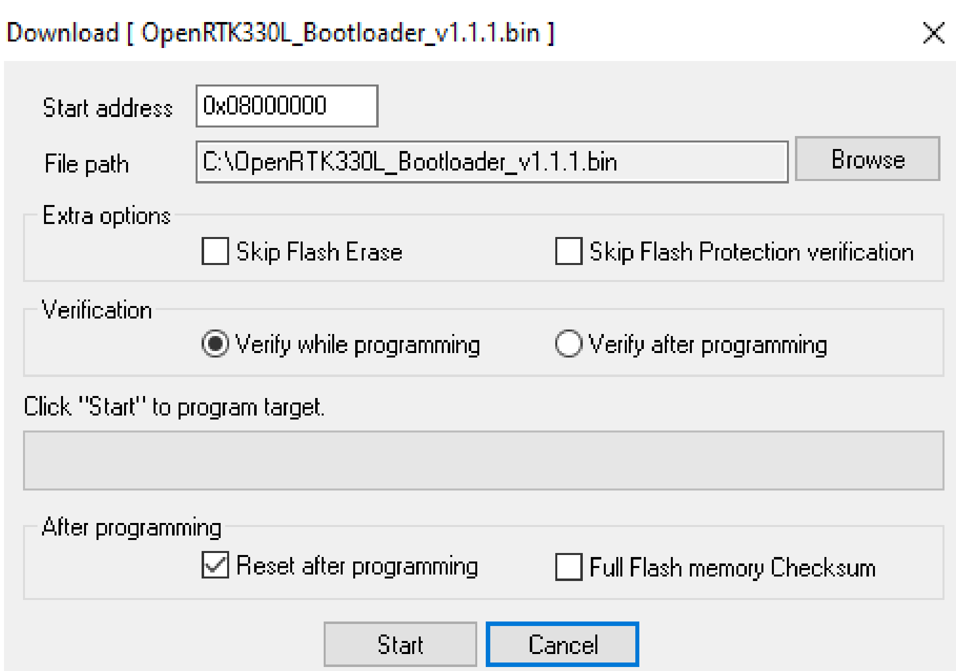
1. Click here to download the latest Python driver (v2.3.0 and later), e.g. "pythondriver-win.zip" for Windows 10
2. Unzip the Python driver on a PC, and run the excutable file "ans-devices.exe" in a command line, e.g.

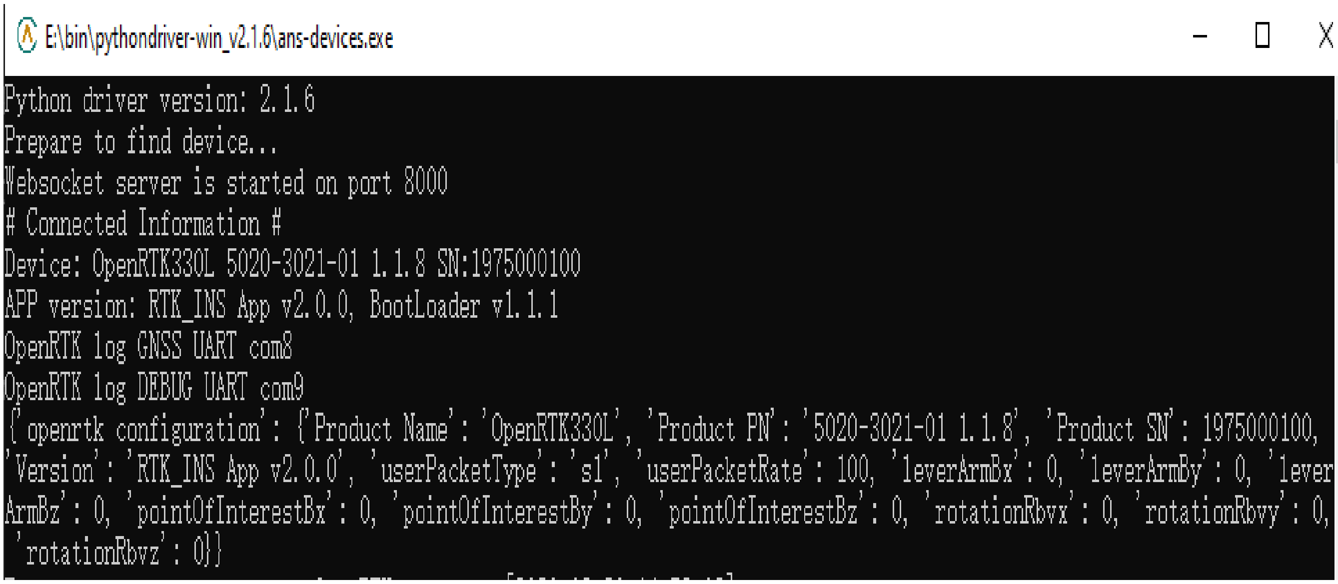
```
c:\pythondriver-win\ans-devices.exe
```

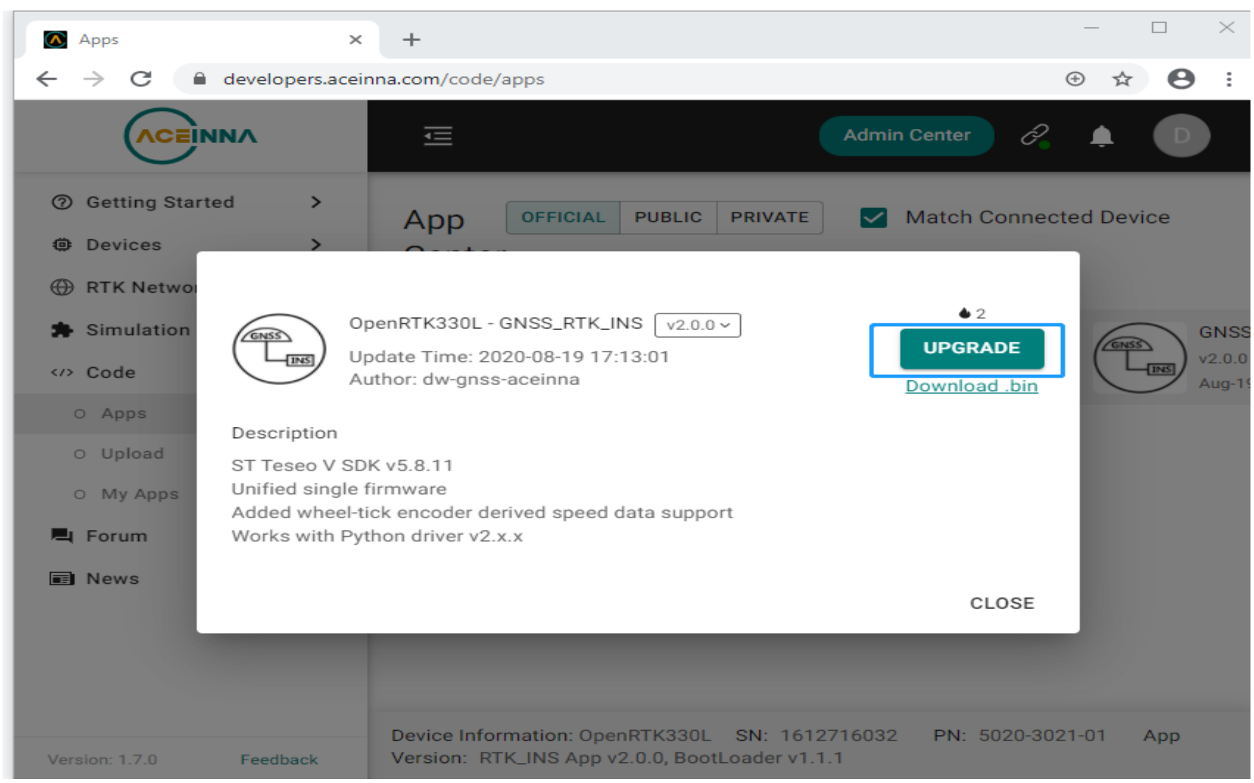
3. Upgrade OpenRTK330 INS App
   - Power on the EVB via connecting a Micro-USB cable between the EVB and a PC, the YELLOW LED starts flashing
   - The python driver keeps scanning available serial ports to connect with OpenRTK330, if connected successfully, you will see the following console output
   - On the above App Center webpage, click "GNSS_RTK_INS" App, and then click the highlighted "UPGRADE" button, the YELLOW LED stops blinking and the GREEN LED starts blinking quickly
   - Upon finishing, you will see the dialog below on the App Center webpage. USER DO NOT have to do any operation, wait for the YELLOW LED to recover blinking. The GREEN LED will start blinking if connected to a GNSS antenna with valid signal receiving

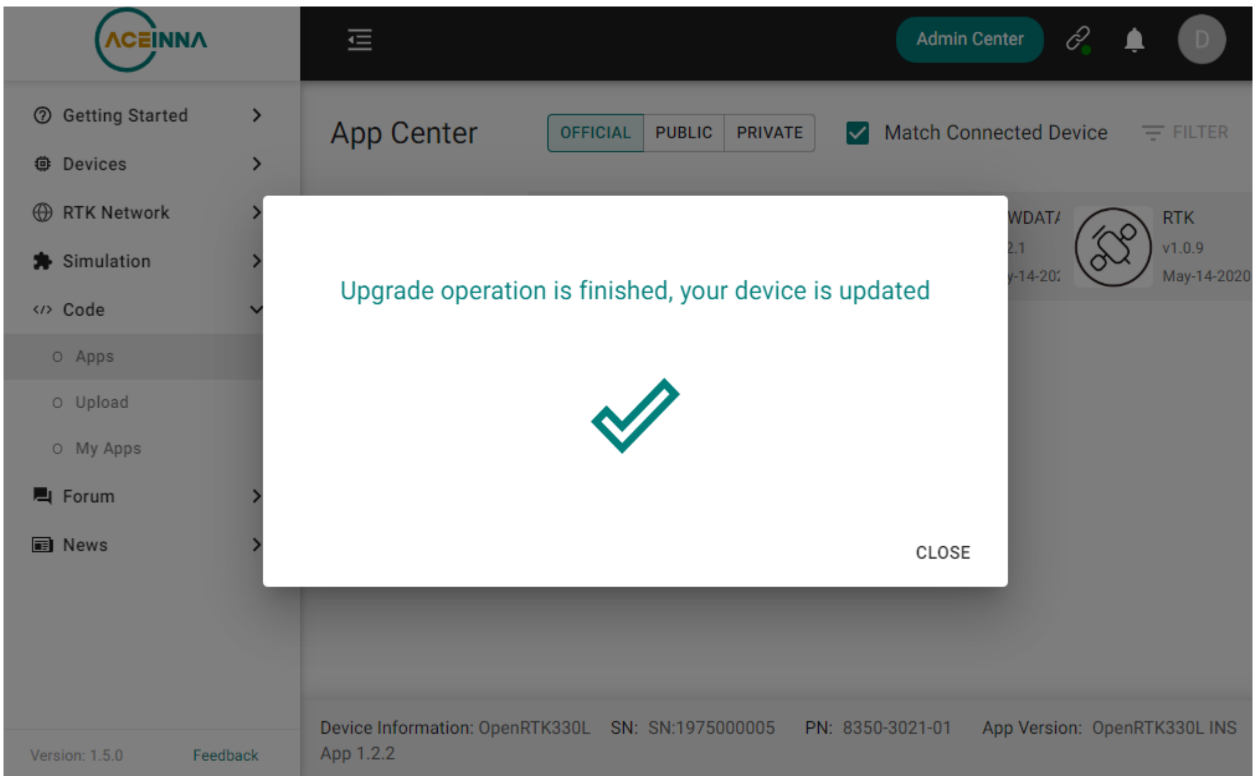

Option Bytes
Read Out Protection
Level 0
BOR Level
F OFF R
User configuration option byte
IWDG_STOP
IWDG_STDBY
nBoot0
nBOOT0
WWDG_SW
IWDG_ULP
nBoot1
BOOT1
nSRAM_Parity
FZ_IWDG_STOP
nDBOOT
BFB2
SRAM2_RST
FZ_IWDG_STDBY
nDBANK
nBOOT_SEL
SRAM2_PE
PCROP_RDP
DB1M
DUALBANK
nRST_SHDW
nBoot0_SW_Cfg
IRHEN
BOREN
nRST_STOP
nSWBOOT0
IWDG_SW
nRST_STDBY
VDDA_Monitor
SDADC12_VDD_Monitor
NRST_MODE
Security option bytes
SEC_SIZE 0x00
SEC_SIZE2 0x00
BOOT_LOCK
Boot address option bytes
BOOT_ADD0 (H)
Boot from (H)
BOOT_ADD1 (H)
Boot from (H)
User data storage option bytes
Data 0 (H)
Data 1 (H)
Flash sectors protection
Flash protection mode:
Write protection
Sector
Start address
Size
Protection
Sector 0 0x08000000 16 K Write Protection
Sector 1 0x08004000 16 K Write Protection
Sector 2 0x08008000 16 K Write Protection
Sector 3 0x0800C000
Select page(s) to protect
Sector 4 0x08010000
Sector 5 0x08020000 128 K No Protection
Sector 6 0x08040000 128 K No Protection
Sector 7 0x08060000 128 K No Protection
Sector 8 0x08080000 128 K No Protection
Unselect all
Select all
Apply
Cancel


Apps
developers.aceinna.com/code/apps
ACEINNA
LOGIN
JOIN
Getting Started
Devices
RTK Network
Simulation
Code
Apps
App Center
OFFICIAL
PUBLIC
PRIVATE
Match Connected Device
FILTER
OpenRTK330L
DEMO
v0.0.1
Mar-16-2020
22
RAWDATA
v1.2.1
May-14-2020
27
Bootloader
v1.1.1
Aug-19-2020
5
GNSS_RTK_INS
v2.0.0
Aug-19-2020
2


Download [OpenRTK330L_Bootloader_v1.1.1.bin]
Start address
0x08000000
File path
C:\OpenRTK330L_Bootloader_v1.1.1.bin
Browse
Extra options
Skip Flash Erase
Skip Flash Protection verification
Verification
Verify while programming
Verify after programming
Click "Start" to program target.
After programming
Reset after programming
Full Flash memory Checksum
Start
Cancel


E:\bin\pythondriver-win_v2.1.6\ans-devices.exe
Python driver version: 2.1.6
Prepare to find device...
Websocket server is started on port 8000
Connected Information
Device: OpenRTK330L 5020-3021-01 1.1.8 SN:1975000100
APP version: RTK_INS App v2.0.0, BootLoader v1.1.1
OpenRTK log GNSS UART com8
OpenRTK log DEBUG UART com9
{'openrtk configuration': {'Product Name': 'OpenRTK330L', 'Product PN': '5020-3021-01 1.1.8', 'Product SN': 1975000100, 'Version': 'RTK_INS App v2.0.0', 'userPacketType': 's1', 'userPacketRate': 100, 'leverArmBx': 0, 'leverArmBy': 0, 'leverArmBz': 0, 'pointOfInterestBx': 0, 'pointOfInterestBy': 0, 'pointOfInterestBz': 0, 'rotationRbvx': 0, 'rotationRbvy': 0, 'rotationRbvz': 0}}


Apps
developers.aceinna.com/code/apps
ACEINNA
Admin Center
Getting Started
Devices
RTK Netwo
Simulation
Code
Apps
Upload
My Apps
Forum
News
Version: 1.7.0
Feedback
App
OFFICIAL
PUBLIC
PRIVATE
Match Connected Device
OpenRTK330L - GNSS_RTK_INS
v2.0.0
Update Time: 2020-08-19 17:13:01
Author: dw-gnss-aceinna
UPGRADE
Download .bin
Description
ST Teseo V SDK v5.8.11
Unified single firmware
Added wheel-tick encoder derived speed data support
Works with Python driver v2.x.x
CLOSE
Device Information: OpenRTK330L SN: 1612716032 PN: 5020-3021-01 App Version: RTK_INS App v2.0.0, BootLoader v1.1.1


ACEINNA
Admin Center
D
Getting Started
Devices
RTK Network
Simulation
Code
Apps
Upload
My Apps
Forum
News
Version: 1.5.0
Feedback
App Center
OFFICIAL
PUBLIC
PRIVATE
Match Connected Device
FILTER
RTK
v1.0.9
May-14-2020
Upgrade operation is finished, your device is updated
CLOSE
Device Information: OpenRTK330L SN: SN:1975000005 PN: 8350-3021-01 App Version: OpenRTK330L INS App 1.2.2

CHAPTER 5

OpenARC GNSS Correction Service

Contents

5.1 Introduction

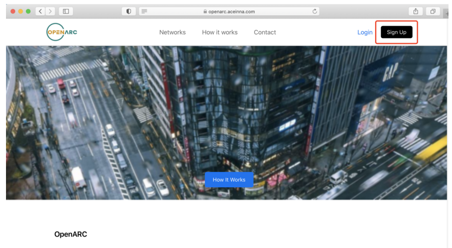
OpenARC is Aceinna's precise positioning platform that offers easy system integration of GNSS corrections with high performance GNSS RTK/INS hardware. OpenARC provides secure GNSS corrections powered by a dense RTK network nation-wide over the United States and a cloud-based architecture.OpenARC offers performance (<10 cm accuracy with no latency), security and integrity (fault tolerance and encryption) and flexibility, while being cost effective.

OpenARC service is inherently supported by the OpenRTK330LI navigation module and its cloud service interface is embedded in the module firmware, and provides a vertically integrated and seamless positioning platform for industrial and autonomous vehicle applications.

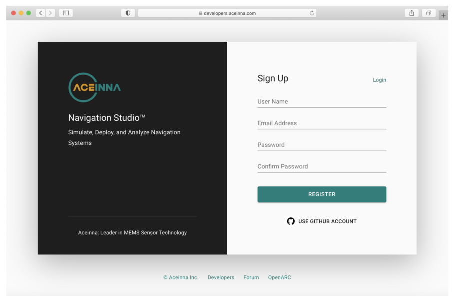
5.2 Usage with the OpenRTK330LI Module

1. **Register an OpenARC user account**

   a. Go to https://openarc.aceinna.com, click "Sign Up" to register an account.

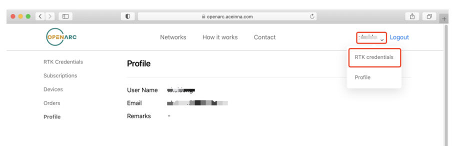


b. On the Sign Up page, enter the user name, email, password and confirm password to register, or directly use your GitHub account to register

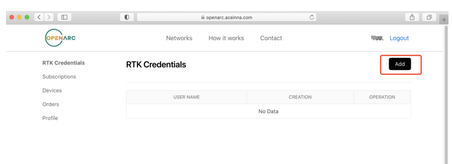


2. **Create GNSS correction service account**

   a. Login your OpenARC user account, click on your username that is located on the right-upper corner of the web page, then click on "RTK credentials"



   b. On the "RTK Credentials" web page, click "Add" button

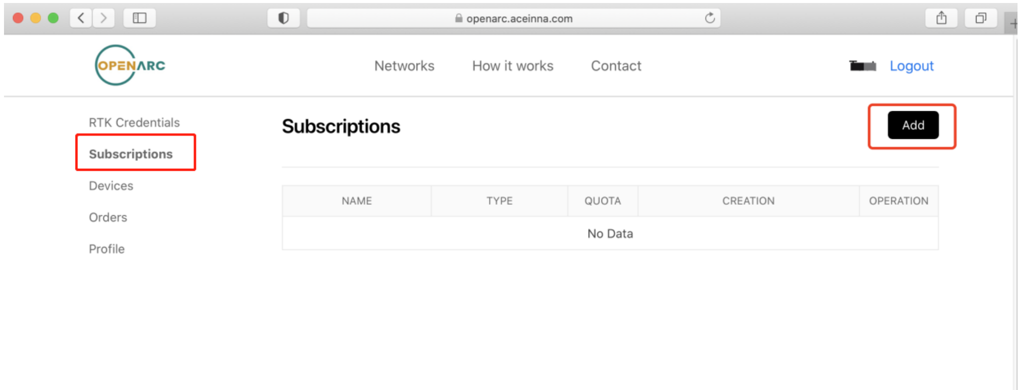


   c. On the "Create RTK Credentials" webpage, create a username and password for your GNSS correction data service, which will be used as the "username" and "password" for a typical NTRIP setting, e.g.

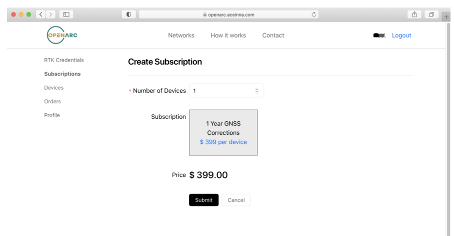
      - *IP Address: openarc.aceinna.com*
      - *PORT: 8011*
      - *Mount Point: RTK*
      - *User Name: username*
      - *Password: password*

3. **Subscribe correction service**

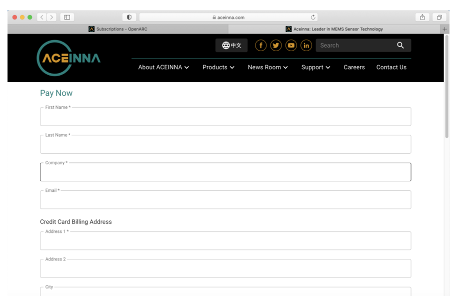
   a. On your OpenARC account webpage, click "Subscriptions" on the left side menu, and then click the "Add" button to create a new data service subscription,



b. On the "Create Subscription" page, select the subscription type and modify the number of devices that will be associated with this subscription, and then click "Submit" button to get to the payment page,

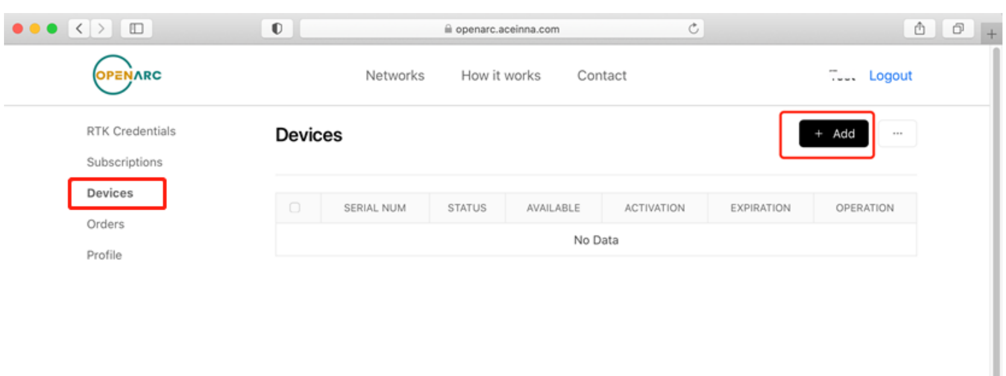


c. Fill in your payment method information, and complete the OpenARC GNSS correction service account creation and subsription.

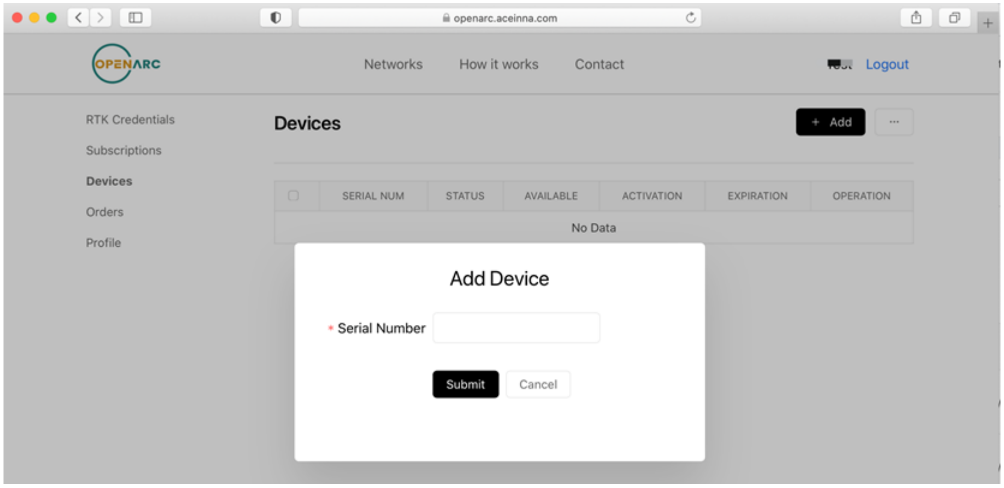


4. **Bundle your OpenRTK330LI device**

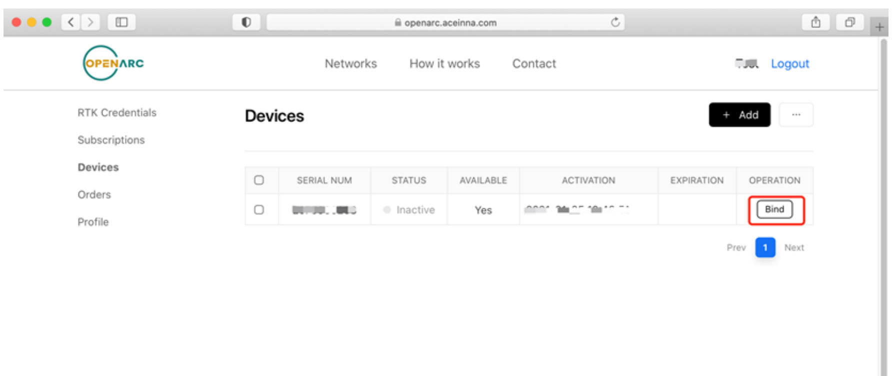
a. On your OpenARC account webpage, click "Devices" on the left side menu, and then click the "Add" button to start adding a device,

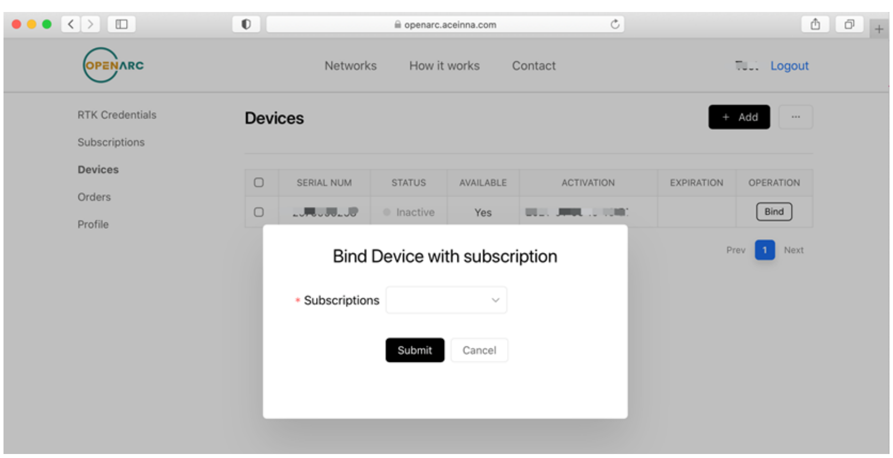


b. On the pop up window, enter your OpenRTK330LI device's serial number manually. This step is optional as OpenARC will associate your device with your subscription automatically when the device is connected with OpenARC for the first time. Each OpenRTK330LI device has a service trial time after you registered with OpenARC by default, which means during this time you can perform RTK positioning with OpenRTK330LI device.



c. Once your OpenRTK330LI device is associated your OpenARC account, for each device on the device list you can click the "bind" button to bundle with your purchased RTK correction service subscription.





Part III

RTK/IMU Modules

CHAPTER 6

The OpenRTK330LI Module

The Aceinna OpenRTK330 module integrates a ST Teseo V automotive grade multi-constellation, multi-frequency Global Navigation Satellite System (**GNSS**) chipset (supports GPS, GALILEO, GLONASS, Beidou, QZSS), a triple-redundant 6-axis (3-axis accelerometer and 3-axis gyro) **MEMS** Inertial Measurement Unit (**IMU**), and a ST M4 MCU as the processor. OpenRTK330 module is targeted for commecial applicaiton for the mass market that requires a reliable, high-precision and yet cost effective **GNSS/INS** integrated positioning solution.

Features with:

- 100 Hz GNSS/INS integrated position, velocity and attitude solution
- Integrated tripple redundant 6-axis IMU sensors
- Integrated multi-frequency GNSS chipset with the following two frequency plans

GNSS	L1/L2 plan	L1/L5 plan
GPS	L1 C/A + L2C	L1 C/A + L5
GLONASS	G1	G1
BeiDou	B1I + B2I	B1I + B2A
Galileo	E1 + E5b	E1 + E5a
QZSS	L1C + L2C	L1C + L5

- RTK algorithms on-board for up to centimetre accuracy
- UART / SPI / CAN / Ethernet Interfaces

6.1 Technical characteristics

Accuracy[1]	
Horizontal Position Accuracy (RMS)	
SPS	1.2 m CEP
RTK[2]	0.02 m

Continued on next page

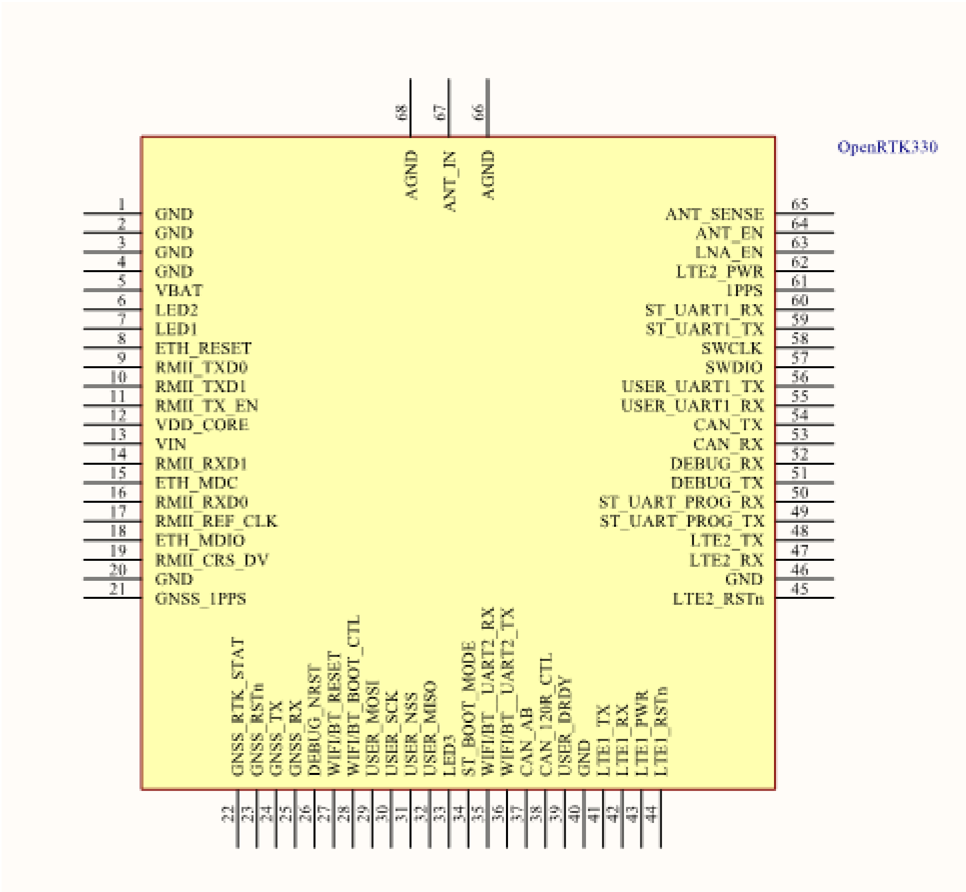
Table 1 – continued from previous page

10s GNSS Outage	0.4 m
Vertical Position Accuracy (RMS)	
SPS	1.8 m CEP
RTK	0.03 m
10s GNSS Outage	0.6 m
Velocity Accuracy (RMS)	
Horizontal	0.02 m/s
Vertical	0.02 m/s
Heading Accuracy (RMS)[3]	0.5°
Attitude Accuracy (Roll/Pitch, RMS)	0.1°
Operating Limits	
Velocity	515 m/s
Acceleration	8 g
Angular Rate	400 °/s
Temperature Calibration Range	-40 °C to +85 °C
Timing	
Time to First Fix[4]	
Cold Start[5]	< 60 s
Warm Start[6]	< 45 s
Hot Start	< 11 s
Signal Re-acquisition	< 2 s
RTK Initialization Time	< 15 s
INS PVA output rate	100 Hz
Sensitivity	
Tracking	-160 dBm
Cold Start	-140 dBm
Environment	
Operating Temperature (°C)	-40 to +85
Non-Operating Temperature (°C)	-55 to +105
Vibration	IEC 60068-2-6 (5g)
Shock survival	MIL-STD-810G (40g)
Electrical	
Input Voltage (VDC)	2.7 to 5.5 V
Power Consumption (W)	1.0 (Typical)
Digital Interface	UART, CAN, SPI, Ethernet
Physical	
Package Type	50-pin LGA
Size (mm)	31 x 34 x 5
Weight (gm)	5

[1] Typical values, subject to ionospheric/tropospheric conditions, satellite geometry, baseline length, multipath and interference effects.
[2] Add 1ppm of baseline length.
[3] After dynamic motion initialization.
[4] Typical values.
[5] No previous satellite or position information.
[6] Using ephemeris and last known position.

Notes

6.2 Pin Definitions



No.	Name	Type	Description
1	GND	P	Ground
2	GND	P	Ground
3	GND	P	Ground
4	GND	P	Ground
5	VBAT	P	Reserved
6	LED2	O	Status2 LED
7	LED1	O	Status1 LED
8	ETH_RESET	O	Reset signal of ETH RMII interface
9	RMII_TXD0	O	Transmit data0 of ETH RMII interface
10	RMII_TXD1	O	Transmit data1 of ETH RMII interface
11	RMII_TX_EN	O	Transmit enable of ETH RMII interface
12	VDD_CORE	P	Reserved
13	VIN	P	Typical DC3.3V, input voltage DC3.0V~3.6V
14	RMII_RXD1	I	Receive data1 of ETH RMII interface
15	ETH_MDC	O	Management interface (MII) clock output
16	RMII_RXD0	I	Receive data0 of ETH RMII interface

Continued on next page

Table 2 – continued from previous page

17	RMII_REF_CLK	I	Clock signal of ETH RMII Interface
18	ETH_MDIO	I/O	Management interface (MII) data I/O
19	RMII_CRS_DV	O	Carrier sense/receive data valid output of ETH RMII interface
20	GND	P	Ground
21	GNSS_1PPS	I	1PPS signal from external GNSS module
22	GNSS_RTK_STAT	I	RTK status signal from external GNSS module
23	GNSS_RSTn	O	Reset signal to external GNSS module
24	GNSS_TX	I	Receive data from external GNSS module
25	GNSS_RX	O	Transmit data to external GNSS module
26	DEBUG_NRST	I	Reset signal of MCU debug interface
27	WIFI/BT_RESET	O	Rest signal for external WIFI/BT module
28	WIFI/BT_BOOT_CTL	O	Boot mode select signal for external WIFI/BT module
29	USER_MOSI	I	SPI interface. Receive data from master
30	USER_SCK	I	SPI interface. Clock signal from master
31	USER_NSS	I	SPI interface. Chip selected signal from master
32	USER_MISO	O	SPI interface. Transmit data to master
33	LED3	O	Status3 LED
34	ST_BOOT_MODE	I	Boot mode control signal for internal ST GNSS chip
35	WIFI/BT_UART2_RX	I	Receive data from external WiFi/BT module
36	WIFI/BT_UART2_TX	O	Transmit data to external WiFi/BT module
37	CAN_AB	O	CAN bus transceiver loopback mode control
38	CAN_120R_CTL	O	CAN termination resistor control (ON/OFF)
39	USER-DRDY	O	Data ready signal
40	GND	P	Ground
41	LTE1_TX	O	Transmit data to external LTE module 1
42	LTE1_RX	I	Receive data from external LTE module 1
43	LTE1_PWR	O	Power control signal for external LTE module 1
44	LTE1_RSTn	O	Reset signal of external LTE module 1
45	LTE2_RSTn	O	Reset signal of external LTE module 2
46	GND	P	Ground
47	LTE2_RX	I	Receive data from external LTE module 2
48	LTE2_TX	O	Transmit data to external LTE module 2
49	ST_UART_PROG_TX	O	Receive data from internal ST GNSS UART2 (GNSS program burning)
50	ST_UART_PROG_RX	I	Transmit data to internal ST GNSS UART2 (GNSS program burning)
51	DEBUG_TX	O	Transmit data. DEBUG serial port
52	DEBUG_RX	I	Receive data. DEBUG serial port
53	CAN_RX	I	Receive data from CAN bus
54	CAN_TX	O	Transmit data to CAN bus
55	USER_UART1_RX	I	Receive data. USER port
56	USER_UART1_TX	O	Transmit data. USER port
57	SWDIO	I/O	Data IO of SWD debug interface
58	SWCLK	I	Clock signal of SWD debug interface
59	ST_UART1_TX	O	Transmit data from internal ST GNSS UART1 port (debug data)
60	ST_UART1_RX	I	Receive data to internal ST GNSS UART1 port (debug data)
61	1PPS	O	1PPS signal
62	LTE2_PWR	O	Power control signal for external LTE module 2
63	LNA_EN	O	Control signal of external LNA power
64	ANT_EN	O	Antenna enable, reserved
65	ANT_SENSE	I	Antenna sensing detection, reserved
66	AGND	P	Internal GNSS RF path ground

Continued on next page

Table 2 – continued from previous page

67	ANT_IN	I	GNSS antenna signal input
68	AGND	P	Internal GNSS RF path ground

6.3 Communication Ports and Operation

USER UART has serial port IAP (program firmware APP) function, ST UART PROG serial port is the serial port for SDK firmware programming, users must connect. BT UART and ETH can pull base rtcm3 for RTK operation, users need to choose at least one connection. Other interface users can connect according to their needs. The hardware design can refer to OpenRTK330 EVK.

6.3.1 User Port

- **Pin**: USER_UART_RX(#55), USER_UART_TX(#56)
- **Default configuration**
- Baud tare: 460800 b/s
- Stop bit: 1
- Data bits: 8
- Check Digit: None
- **Data format**: ACEINNA format, NMEA format
- **The main function**
- Obtain module information: hardware version number, software version number;
- Obtain and configure module user parameters;
- Send data packets: IMU raw data, positioning data, satellite data;
- Send NMEA format data;
- **Function details**

  The following takes configuration parameters as an example to introduce how to use the ACEINNA format:

  1) Send the "gA" command to the module to obtain all current user parameters:

     gA command: [0x55, 0x55, 0x67, 0x41, 0, 0x31, 0x0A]

  2) Use the "uP" command to modify the parameters:

     uP command: [0x55, 0x55, 0x75, 0x50, data length, parameter number, parameter value, CRC_L, CRC_H]

     For example: configure the three parameters of leverArmBx, leverArmBy, leverArmBz to [0.5, -0.5, 1] (unit m), you need to send the "uP" command three times, and the setting result will be returned each time. After the last setting result is returned, send again Set the command next time.

     - **Configure leverArmBx**: [0x55, 0x55, 0x75, 0x50, 0x08, 0x04, 0, 0, 0, 0, 0, 0, 0x3F, 0x1D, 0x32]
     - **Configure leverArmBy**: [0x55, 0x55, 0x75, 0x50, 0x08, 0x05, 0, 0, 0, 0, 0, 0, 0xBF, 0xCB, 0x69]

- **Configure leverArmBz**: [0x55, 0x55, 0x75, 0x50, 0x08, 0x06, 0, 0, 0, 0, 0, 0x80, 0x3F, 0x89, 0x0C]

3) Use the "sC" command to save the parameter value:

   sC command: [0x55, 0x55, 0x73, 0x43, 0, 0xC8, 0xCB]

Special Note

```
"initial":{
   "useDefaultUart": 1,
      "uart":[
       {
           "name": "GNSS",
           "value": "com10",
           "enable": 1
       },
       {
           "name": "DEBUG",
           "value": "com11",
           "enable": 1
       }
    ],
   "userParameters": [
       {
           "paramId": 4,
           "name": "lever arm x",
           "value": 0.0
       },
       {
           "paramId": 5,
           "name": "lever arm y",
           "value": 0.0
       }
    ]
 }
```

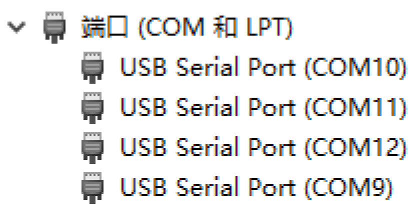
The user serial port is the serial port connected by the python driver. If the user needs to enable the data log function or automatically configure user parameters when the python driver is started, first configure the "initial" field in openrtk.json as shown in Figure above.

Use OpenRTK/OpenIMU python driver operation

1) Set the log serial port

When the python driver is started with the "-r" suffix, the log function will be enabled and the data of the three serial ports of USER, GNSS and DEBUG will be recorded at the same time. The USER serial port number can be automatically identified by the python driver, but GNSS and DEBUG cannot. The user must set these two serial port numbers.

Case 1: The GNSS/DEBUG of OpenRTK330 EVK is the USER serial port number plus 1 and 2 respectively. Just configure the "useDefaultUart" field to 1, and the "uart" field does not work at this time.



Case 2: If the user needs to specify the GNSS/DEBUG serial port number, or does not use the GNSS/DEBUG serial port (the user has not made a hardware connection), the "useDefaultUart" needs to be configured to 0, and the "uart" field is valid at this time, the GNSS/DEBUG When "enable" is 1, it means to use this serial port. When not in use, configure it to 0. "Value" should be the serial port name of the serial port in the system. For example: under windos, open the device manager, as shown in Figure above, find the connected GNSS and DEBUG serial numbers are COM10 and COM11 respectively, the configuration should be as follows:

```
"uart":[
            {
                "name": "GNSS",
                "value": "com10",
                "enable": 1
            },
            {
                "name": "DEBUG",
                "value": "com11",
                "enable": 1
            }
        ],
```

2) Setting paracmeters

When starting the python driver with the "-s" suffix, the "userParameters" parameters can be automatically configured to the OpenRTK device and saved after power off. Find "userParameters" as shown in Figure 2, and configure fields for user parameters. All configurable fields are in "userConfiguration", except for "Data CRC" and "Data Size" whose paramId is 0 or 1 are not configurable, the others can be added to "userParameters". Among them, "paramId" and "value" are mandatory fields, the value of paramId must be consistent with that in "userConfiguration", and the type of value must be consistent with "type".

For example: to configure Ethernet and NTRIP services, the following configuration is required, where the Ethnet mode value is 1 to use static IP mode, and the value is 0 to use DHCP mode.

```
"userParameters": [
    {
        "paramId": 13,
        "name": "Ethnet mode",
        "value": 1
    },
    {
        "paramId": 14,
        "name": "STATIC IP",
        "value": "192.168.137.110"
    },
    {
        "paramId": 15,
        "name": "NETMASK",
        "value": "255.255.255.0"
    },
    {
        "paramId": 16,
```

(continues on next page)

(continued from previous page)


```
        "name": "GATEWAY",
        "value": "192.168.137.1"
    },
    {
        "paramId": 18,
        "name": "IP",
        "value": "203.107.45.154"
    },
    {
        "paramId": 19,
        "name": "PORT",
        "value": 8001
    },
    {
        "paramId": 20,
        "name": "MOUNT POINT",
        "value": " RTCM32_GGB"
    },
    {
        "paramId": 21,
        "name": "USER NAME",
        "value": "username"
    },
    {
        "paramId": 22,
        "name": "PASSWORD",
        "value": "password"
    }
]
```


6.3.2 ST GNSS UART1

- **Pin**: ST_UART1_TX(#59), ST_UART1_RX(#60)
- **Default configuration**
- Baud tare: 460800 b/s
- Stop bit: 1
- Data bits: 8
- Check Digit: None
- **Data formation**: RTCM3 format
- **Main function**: Send raw data of GNSS receiver satellite signal

6.3.3 DEBUG UART1

- **Pin**: DEBUG_TX(#51), DEBUG_RX(#52)
- **Default configuration**
- Baud tare: 460800 b/s
- Stop bit: 1

- Data bits: 8
- Check Digit: None
- **Data formation**: ASSIC format, “P1” packet format
- **Main function**:
- Send “p1” packet data (more detailed than user serial port data), not sending by default
- Get user parameters (only basic parameters are included, user serial port can get all parameters)
- Control “p1” packet data on or off

6.3.4 ST UART PROG

- **Pin**: ST_UART_PROG_TX(#49), ST_UART1_PROG_RX(#50)
- **Default configuration**
- Baud tare: 460800 b/s
- Stop bit: 1
- Data bits: 8
- Check Digit: None
- **Main function**: ST GNSS chip firmware download interface (SDK download port)

6.3.5 BT UART

- **Pin**: BT_UART2_RX(#35), BT_UART2_TX(#36)
- **Default configuration**
- Baud tare: 460800 b/s
- Stop bit: 1
- Data bits: 8
- Check Digit: None
- **Main function**
- Receive RTCM3 data from GNSS base station
- Send module position data in NMEA GPGGA format

6.3.6 SPI Pin Definition

- **Pin**: USER_MOSI(#29), USER_SCK(#30), USER_NSS(#31), USER_MISO(#32)
- **Default configuration**
- Frame format: Motorola
- Data length: 8 bits
- First bit: 1
- CPOL: High

- CPHA: 2Edge
- **Main function**
- Send "p1" data, "p1" packet format (see 4.3 for details)

6.3.7 CAN Pin Definition

- **Pin**: CAN_RX(#53), CAN_TX(#54)
- **Default configuration**
- ECU address: 128 (automatically match, add 1 to this address, maximum 247)
- Baud rate: 250K
- **Data format**: can communication protocol, which can be divided into the following 3 categories according to functions:
- Setting parameters: the user sends a setting parameter command, the module does not return
- Get parameters: the user sends a get parameter command, the content of the command is the PF number and PS number of the data required by the user, and the module returns the corresponding data frame
- Data packet: The module continuously sends data packets according to the data type and frequency configured by the user
- **Main function**
- Support SAE J1939 protocol
- Configure CAN interface parameters
- Send user data packet

6.3.8 RMII Pin Definition

- **Pin**: ETH_RESET(#8), RMII_TXD0(#9), RMII_TXD1(#10), RMII_TX_EN(#11), VDD_CORE(#12), VIN(#13), RMII_RXD1(#14),

ETH_MDC(#15),RMII_RXD0(#16),RMII_REF_CLK(#17),ETH_MDIO(#18),RMII_CRS_DV(#19)

- **Default configuration**
- DHCP mode
- Hostname: openrtk, you can access the Web Interface through http://openrtk in the LAN
- **Main function**
- Support static IP mode and DHCP mode
- Access to Web Interface configuration parameters (including Ethernet, NTRIP, etc.)
- Establish NTRIP CLIENT to pull base rtcm3 data

Part IV

Evaluation Kits

CHAPTER 7

The OpenRTK330LI EVK

Contents

7.1 1. Introduction

The OpenRTK evaluation kit (EVK) is a hardware platform to evaluate the OpenRTK330 GNSS RTK/INS integrated positioning system and develop various applications based on this platform. Supported by the online Aceinna Navigation Studio the kit provides easy access to the features of OpenRTK330 and explains how to integrate the device in a custom design. The OpenRTK EVK is shown below after unpacking.

where

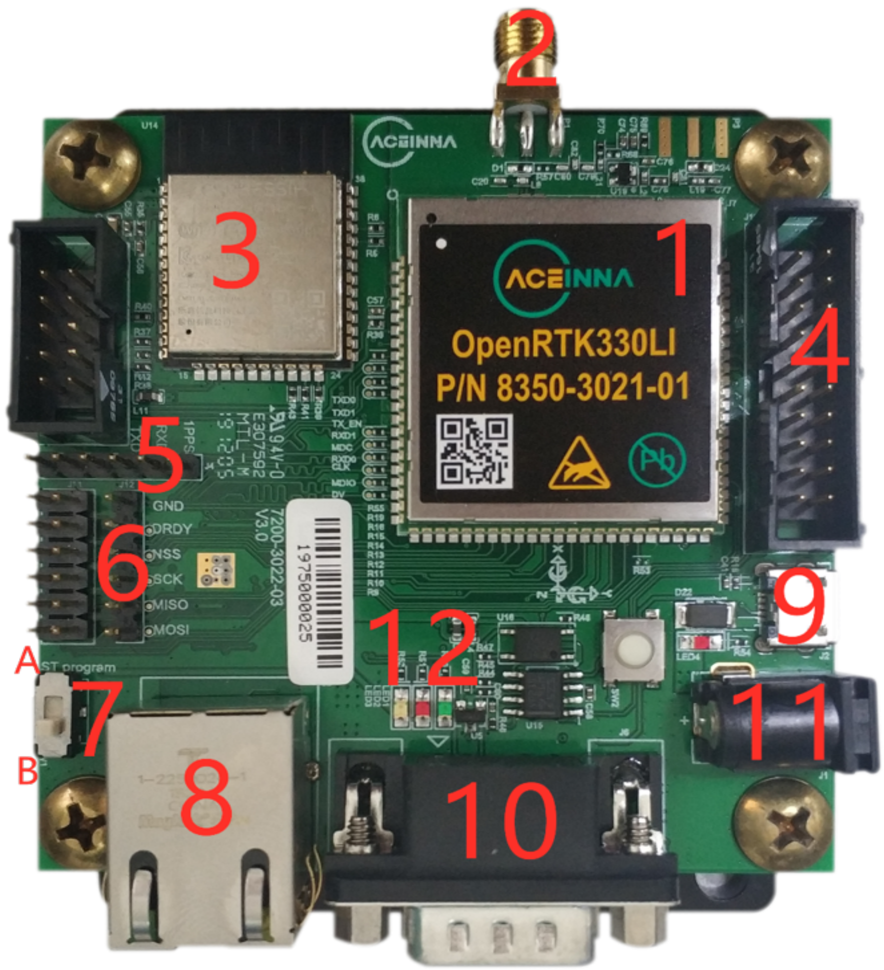
- 1: ST-Link debugger
- 2: Multi-Constellation Multi-frequency GNSS antenna
- 3: Micro-USB cable
- 4: OpenRTK330 Evaluation Board (EVB) with metal flat mounting board
- 5: 12-V DC adapter with 5.5 x 2.1 mm power jack

7.2 2. OpenRTK330 EVB

An OpenRTK330 Evaluation board is shown below in detail

where

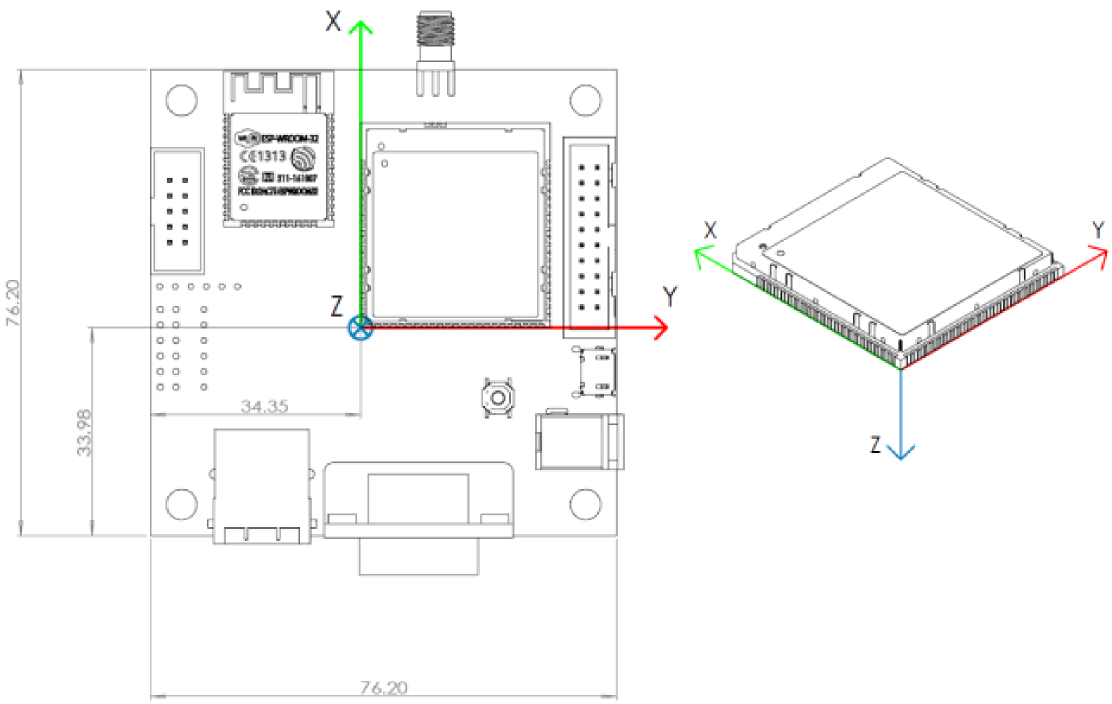
- 1: OpenRTK330 GNSS/IMU integrated module
- 2: GNSS antenna SMA interface
- 3: Espressif ESP32 bluetooth module
- 4: SWD/JTAG connector, 20-pin
- 5: Extension connector with 6-pin interfaces from left to right
  - GND
  - Not Connected


1
2
3
4
5
6
7
8
9
10
11
12
A
B
ACEINNA
OpenRTK330LI
P/N 8350-3021-01
GND
DRDY
NSS
SCK
MISO
MOSI
1PPS
ST program
7200-3022-03
V3.0
1975000025

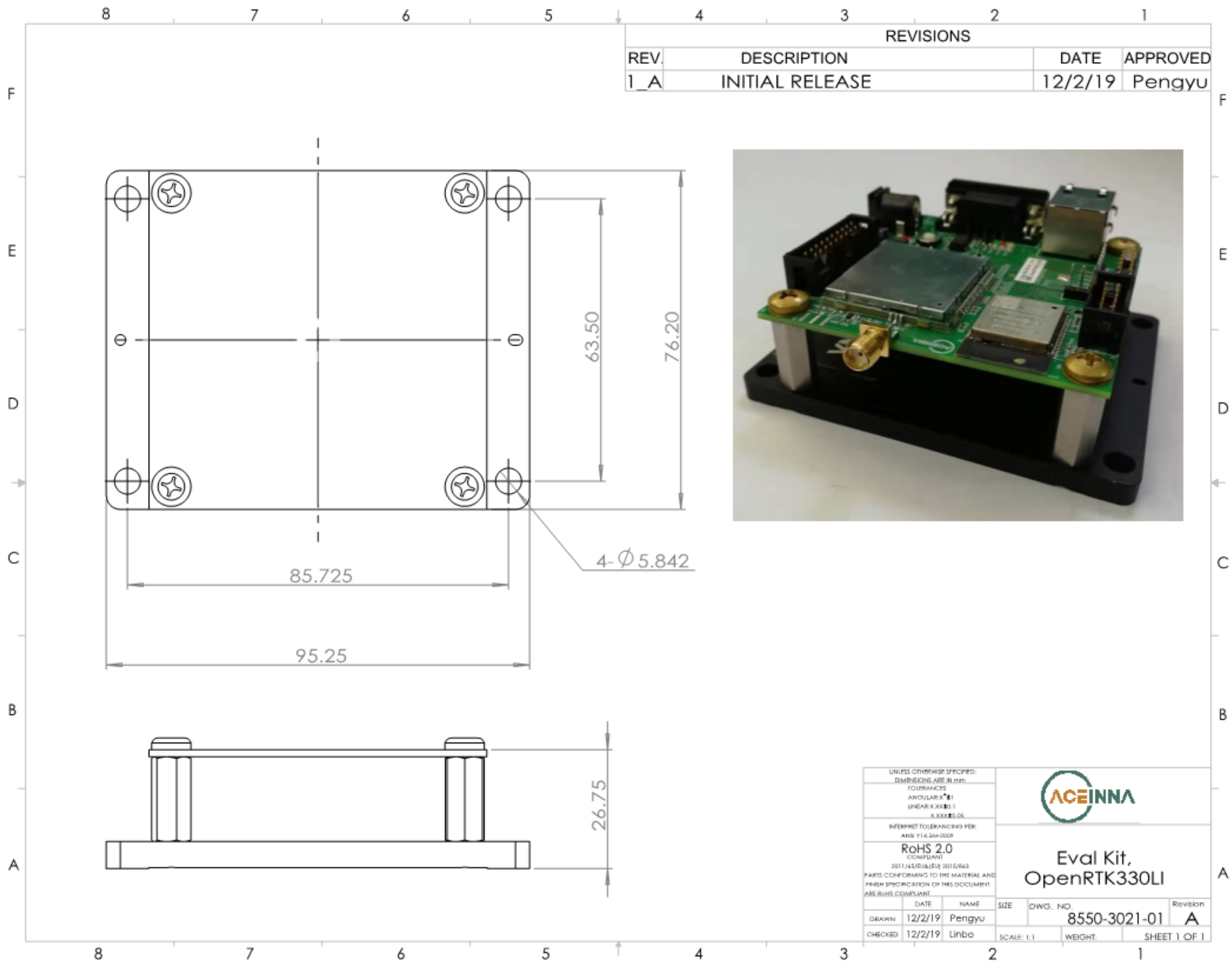
  - Not Connected
  - Connects to pin #56 "USER_UART2_TX" of the OpenRTK330 module
  - Connects to pin #55 "USER_UART2_RX" of the OpenRTK330 module
  - 1PPS outlet
- 6. Extension connector with 6-pin SPI interfaces from left to right
  - Connects to pin #29 "USER_MOSI" of the OpenRTK330 module
  - Connects to pin #30 "USER_SCK" of the OpenRTK330 module
  - Connects to pin #31 "USER_NSS" of the OpenRTK330 module
  - Connects to pin #32 "USER_MISO" of the OpenRTK330 module
  - Connects to pin #39 "USER_DRDY" of the OpenRTK330 module
  - GND
- 7. Boot mode switch with two positions (A and B)
- 8. RJ45 jack for Ethernet interface
- 9. Micro-USB port
- 10. CAN interface
- 11. Power jack for 12-v adapter
- 12. EVB working status LEDs from left to right
  - Yellow: ST GNSS chipset is powered on and working properly
  - Red: valid GNSS base station data receiving
  - Green: valid GNSS signal receiving

7.2.1 EVB Mechanical Drawing

The following mechanical drawing shows the EVB dimension (in mm) and the position of IMU navigation center. The IMU navigation center is fixed to the left-bottom corner of the OpenRTK330LI module on the EVB. User is recommended to measure the level arm from the GNSS antenna phase center to the IMU Navigation center as accurate as possible.

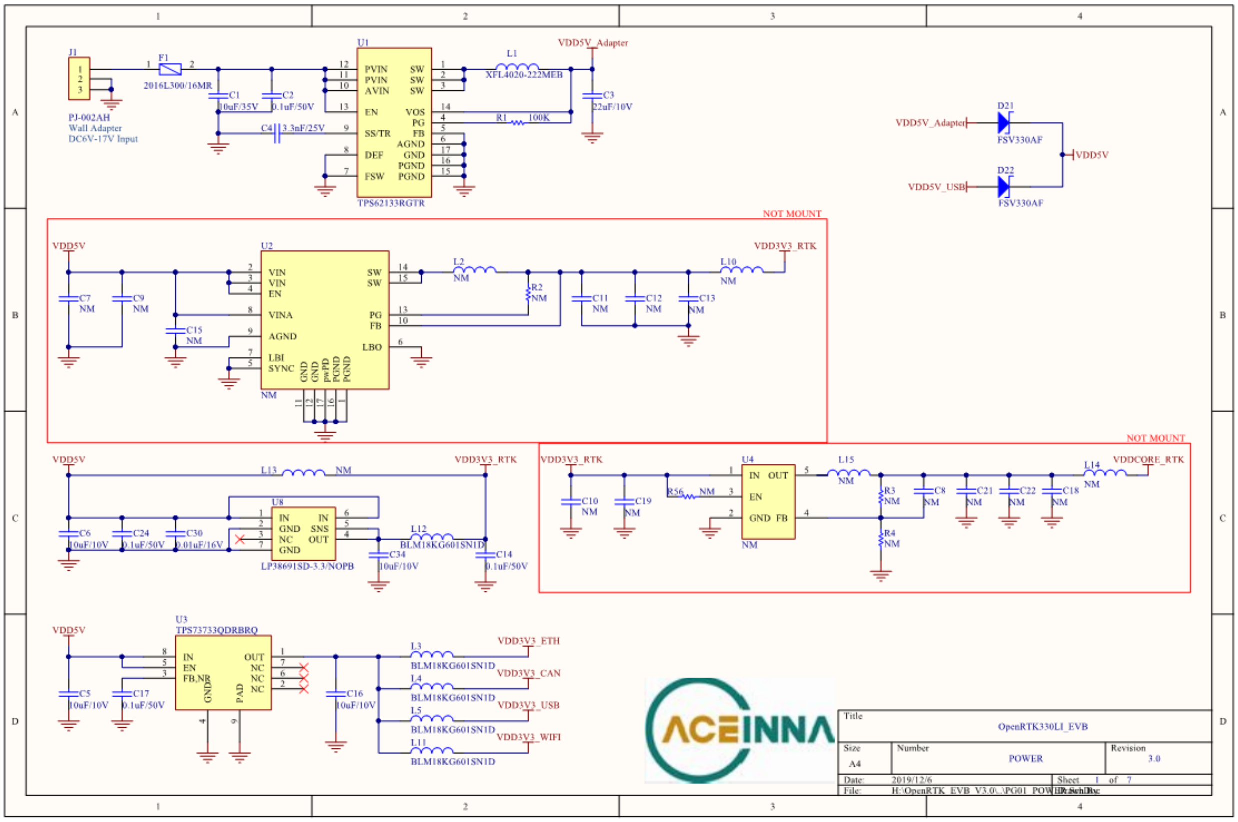


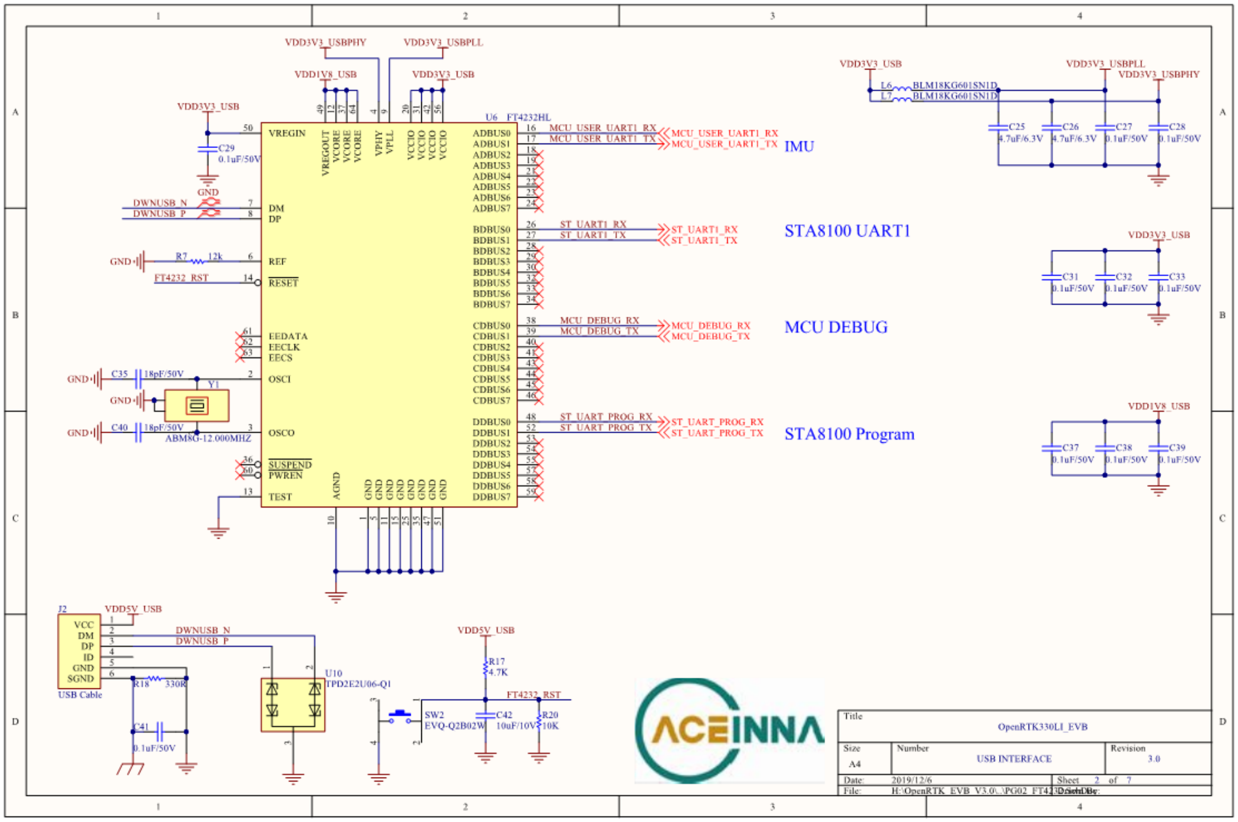
The following mechnical drawing shows the dimension (in mm) of the mounting plate for the OpenRTK330LI EVB:

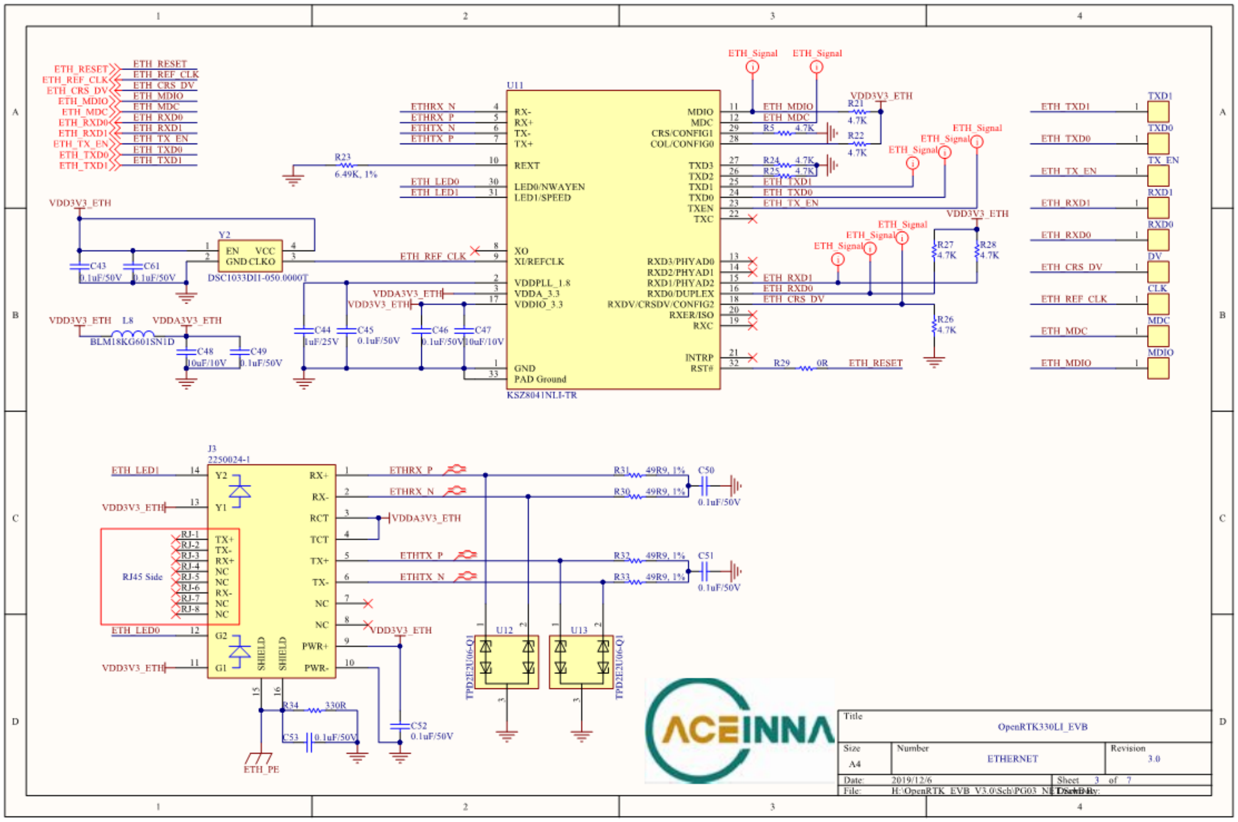


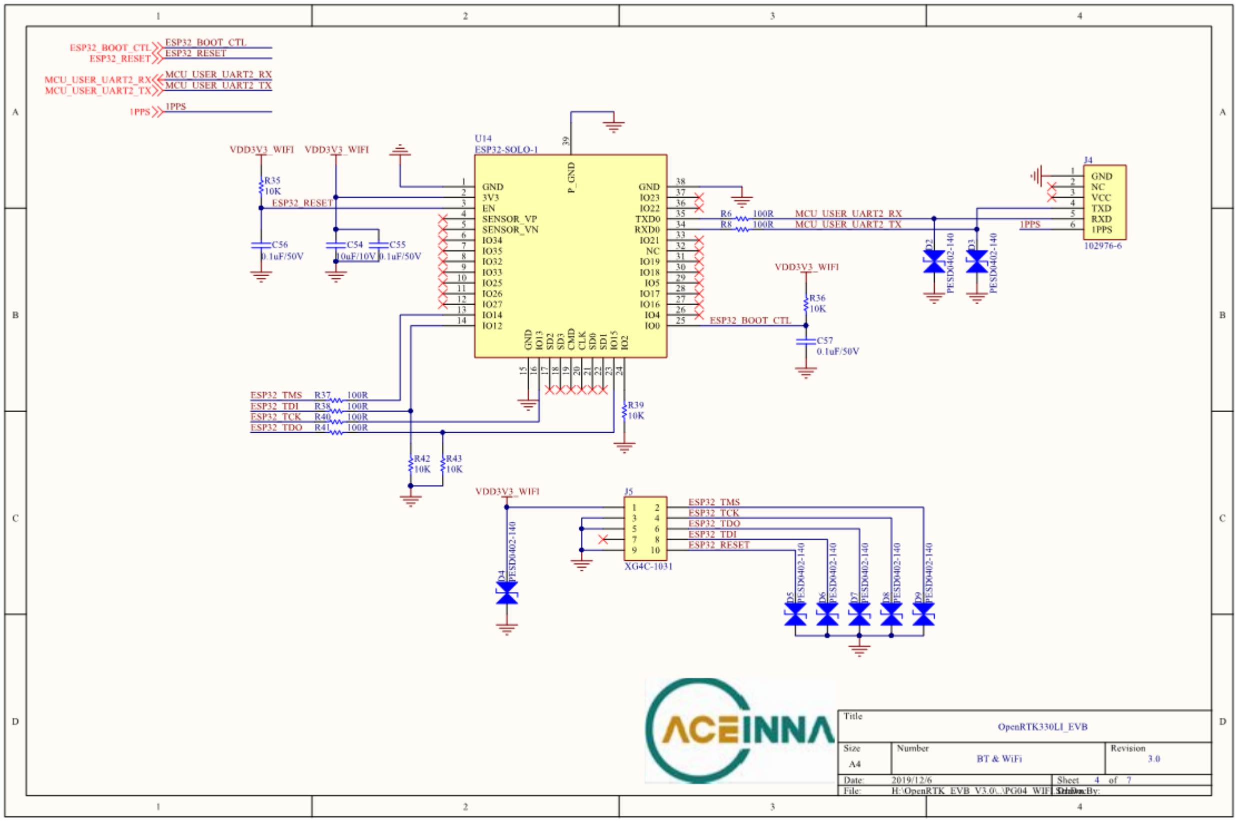
Note: Use the browser's back button to return to this page.

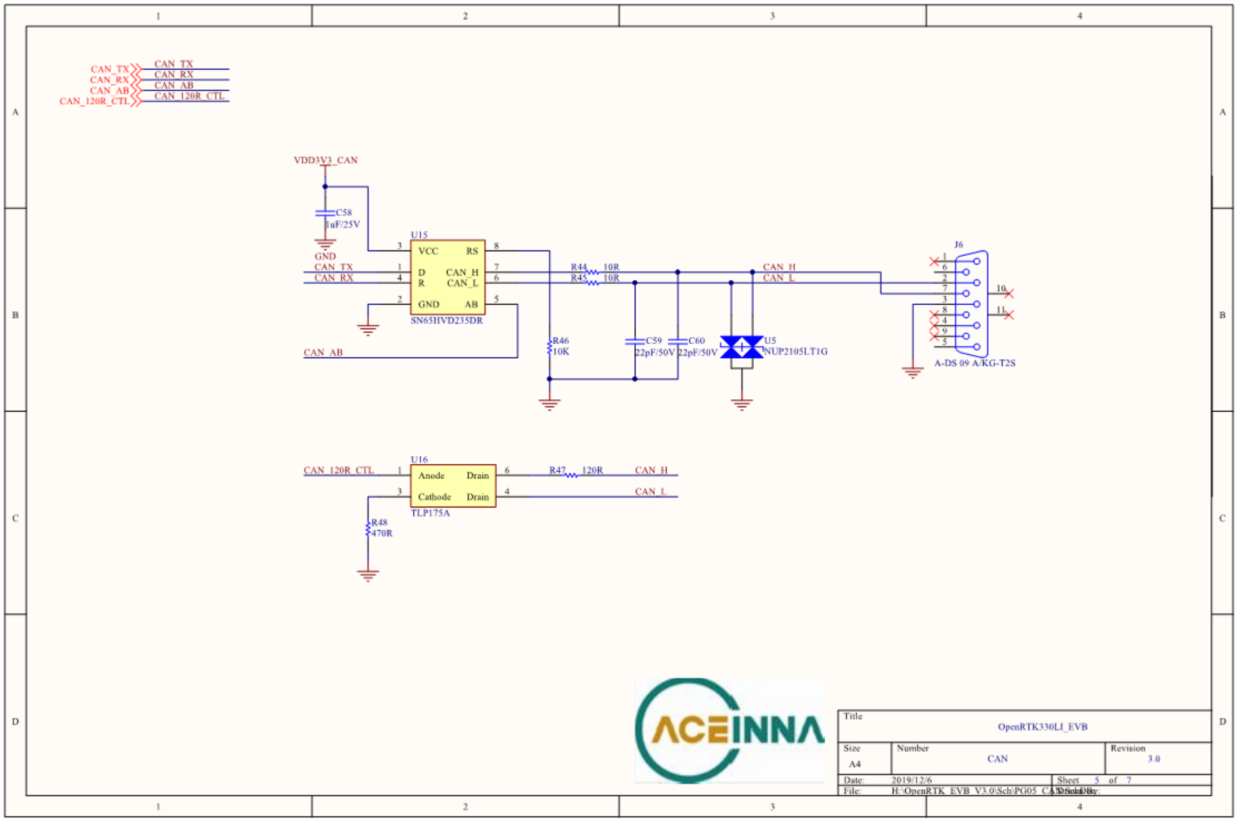
7.2.2 EVB Schematic

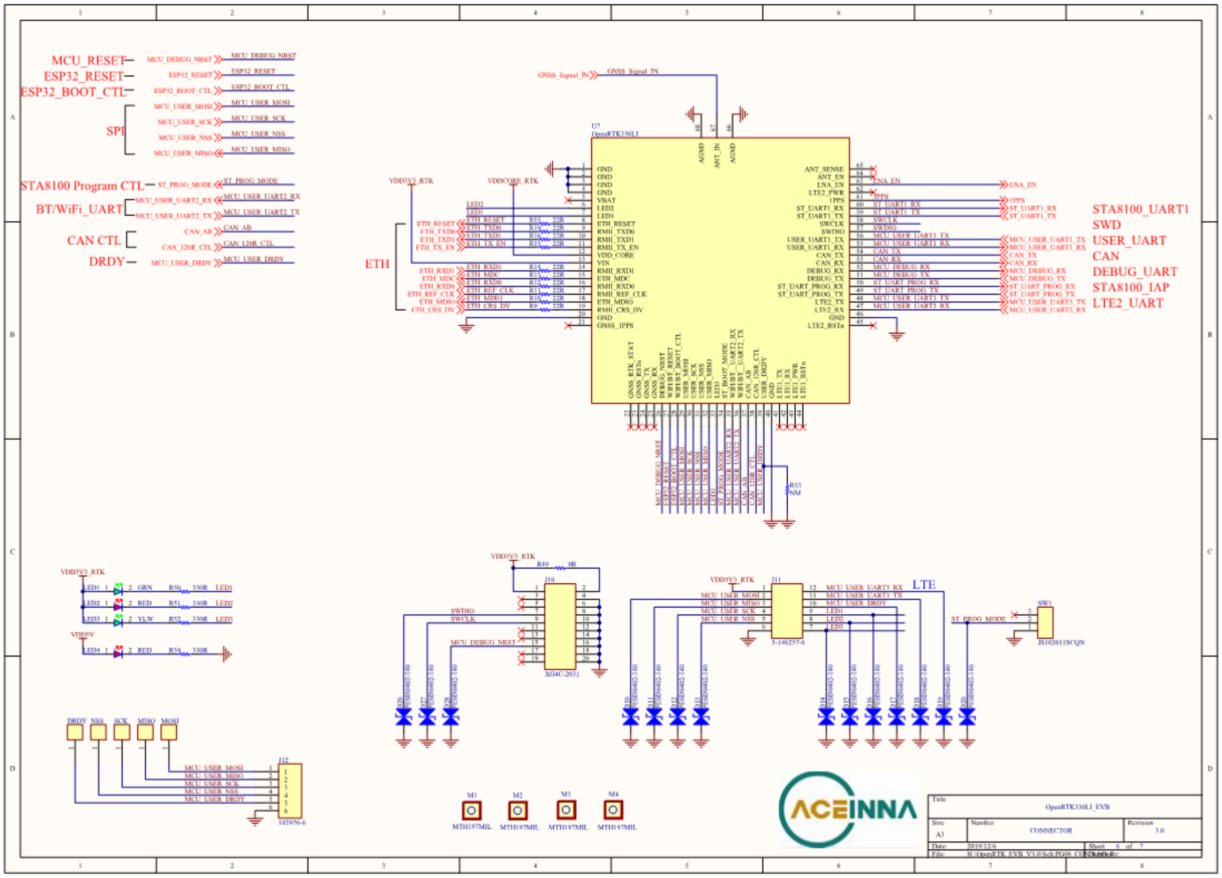


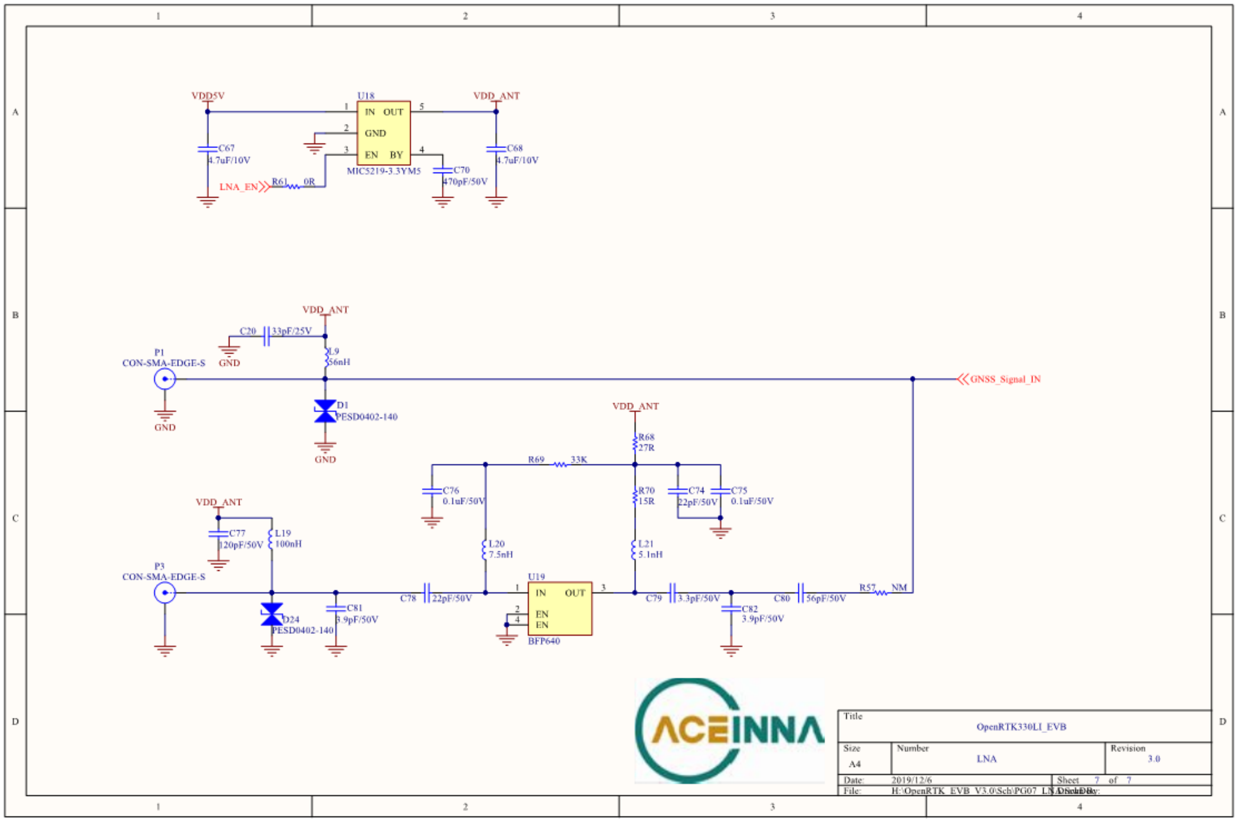

FT4232HL
IMU
STA8100 UART1
MCU DEBUG
STA8100 Program
ACEINNA
OpenRTK330LI_EVB
USB INTERFACE






CAN_TX
CAN_RX
CAN_AB
CAN_120R_CTL
VDD3V3_CAN
C58
1uF/25V
GND
U15
VCC
RS
D
R
CAN_H
CAN_L
GND
AB
SN65HVD235DR
R44 10R
R45 10R
R46
10K
C59
22pF/50V
C60
22pF/50V
U5
NUP2105LT1G
J6
A-DS 09 A/KG-T2S
U16
Anode
Cathode
Drain
TLP175A
R47 120R
R48
470R
ACEINNA
Title
OpenRTK330LI_EVB
Size
A4
Number
CAN
Revision
3.0
Date:
2019/12/6
Sheet 5 of 7
File:


MCU_RESET
ESP32_RESET
ESP32_BOOT_CTL
SPI
STA8100 Program CTL
BT/WiFi_UART
CAN CTL
DRDY
ETH
STA8100_UART1
SWD
USER_UART
CAN
DEBUG_UART
STA8100_IAP
LTE2_UART
LTE
ACEINNA



Schematic `download link`

Part V

Communication protocol

CHAPTER 8

ACEINNA protocol data format definition

Start 1	Start 2	Frame type 1	Frame type 2	Data length 1	Data content	Check 1	Check 2

Description:

- Start: Each frame of data starts with this, 2 bytes: 0x55 0x55.
- Frame type: 2 bytes, high byte first.
- Data length: 1 byte, refers to the byte length of the data content.
- Data content: maximum 255 bytes.
- Check: crc16 check, 2 bytes, low byte first, bytes from the beginning of the "Frame type" to the end of the "Data content" are included in the check calculation, and the check algorithm C code is as follows:

```
uint16_t CalculateCRC (uint8_t *buf, uint16_t  length)
{
   uint16_t crc = 0x1D0F;

   for (int i=0; i < length; i++) {
       crc ^= buf[i] << 8;
       for (int j=0; j<8; j++) {
           if (crc & 0x8000) {
               crc = (crc << 1) ^ 0x1021;
           }
           else {
               crc = crc << 1;
           }
       }
   }

   return ((crc << 8 ) & 0xFF00) | ((crc >> 8) & 0xFF);
}
```

CHAPTER 9

USER UART Data Packet

9.1 Get the hardware version number

9.2 Get the software version number

9.3 Get user parameters

Frame type	
Description	
Request frame	Start
	0x55 0x55
Return frame	Start
	0x55 0x55

Offset	Variable type
0	uint16
2	uint16
4	char * 2
6	uint16
8	float
12	float
16	float

20	float
24	float
28	float
32	float
36	float
40	float
44	uint8
45	uint8 * 4
49	uint8 * 4
53	uint8 * 4
57	uint8 * 6
63	char * 23
86	uint16
88	char * 20
108	char * 16
124	char * 24
148	uint16
150	uint16
152	uint16
154	uint16
156	uint16
158	uint16

9.4 Set user parameters

9.5 Save user parameters

Frame type	**“sC”**				
Description	Save user parameters				
Request frame	Start	Frame type	Data length	Data comment	Check
	0x55 0x55	**0x73 0x43**	0	None	CRC_L CRC_H
Return frame	Start	Frame type	Data length	Data content	Check
	0x55 0x55	**0x73 0x43**	0	None	CRC_L CRC_H
If saving is successful, return as it is; if saving fails, return NAK frame					

9.6 Failed frame

Frame type	**0x15 0x15**				
Description	NAK frame				
Request frame	Start	Frame type	Data length	Data comment	Check
	0x55 0x55	**0x15 0x15**	2	Failed frame type	CRC_L CRC_H

9.7 IMU raw data packet

Frame type	**"s1"**				
Description	IMU raw data				
Data Frame	Start	Frame type	Data length	Data comment	Check
	0x55 0x55	**0x73 0x31**	36	see below	CRC_L CRC_H
Data content:					

Offset	Variable type	Name	Unit	Description
0	uint32	week		GPS week, seconds within GPS week: GPS time
4	double	timeOfWeek	s	
12	float * 3	accel_g[3]	m/s^2	accelerometer(x,y,z)
24	float * 3	rate_dps[3]	deg/s	gyroscope (x,y,z)

9.8 Combined solution PVA packet

Frame type	
Description	
Data Frame	Start
	0x55 0x55

Offset	Variable type
0	uint32
4	double
12	uint32
16	double
24	double
32	double
40	uint32
44	float
48	float
52	uint32
56	uint32
60	uint32
64	float
68	float
72	float
76	float
80	float
84	float
88	float
92	float

96	float
100	float
104	float
108	float
112	float
116	float
120	float

9.9 Satellite information for positioning solution

Frame type	**“sK”**				
Description	Satellite information				
Data Frame	Start	Frame type	Data length	Data comment	Check
	0x55 0x55	**0x73 0x4B**	21*n	see below	CRC_L CRC_H
Data content: a frame of data contains multiple satellite information n					
Offset	Variable type	Name	Unit	Description	
0+n*21	double	timeOfWeek	s	GPS week, seconds within GPS week: accurate to milliseconds within a week	
8+n*21	uint8	satelliteId		atellite number	
9+n*21	uint8	systemId		system number: 0: GPS 1: GLONASS 2: Galileo 3: QZSS 4: BeiDou 5: SBAS	
10+n*21	uint8	antennaId		antenna number: 0: Main antenna 1: Secondary antenna	
11+n*21	uint8	l1cn0		S/N ratio 1: L1	
12+n*21	uint8	l2cn0		S/N ratio 2: L2 / L5	
13+n*21	float	azimuth	deg	azimuth	
17+n*21	float	elevation	m	height	

CHAPTER 10

DEBUG UART Data Packet

10.1 Protocol packet format

Debug uart port data package (P1 package) includes four types of data: “imu”, “gnss”, “vel” and “ins”. Each piece of data contains three parts: packet header, content and check code.

packet header			
Offset	Variable type	Name	Description
0	uint8	sync1	sync 1: 0xAA
1	uint8	sync2	sync 2: 0x44
2	uint8	sync3	sync 3: 0x12
3	uint8	header_length	Length of packet header: 0x1C
4	uint16	message_id	data id: 268-“imu” 42-“gnss” 99-“vel” 507-“ins”
6	uint8	message_type	N/A
7	uint8	port_address	N/A
8	uint16	message_length	Data length: not including header and check code
10	uint16	sequence	N/A
12	uint8	idle	N/A
13	uint8	time_status	N/A
14	uint16	gps_week	GPS week
16	uint32	gps_millisecs	GPS seconds within a week: unit: ms
20	uint32	status	N/A
24	uint16	Reserved	N/A
26	uint16	version	N/A

Check code:

```
#define CRC32_POLYNOMIAL 0xEDB88320L

static unsigned long CRC32Value(int i)
{
```

(continues on next page)

(continued from previous page)

```
    int j;
    unsigned long ulCRC;
    ulCRC = i;
    for (j = 8; j > 0; j--)
    {
        if (ulCRC & 1)
            ulCRC = (ulCRC >> 1) ^ CRC32_POLYNOMIAL;
        else
            ulCRC >>= 1;
    }
    return ulCRC;
}
unsigned long CalculateBlockCRC32(unsigned long ulCount,
                                  unsigned char *ucBuffer)
{
    unsigned long ulTemp1, ulTemp2;
    unsigned long ulCRC = 0;
    while (ulCount-- != 0)
    {
        ulTemp1 = (ulCRC >> 8) & 0x00FFFFFFL;
        ulTemp2 = CRC32Value(((int)ulCRC ^ *ucBuffer++) & 0xff);
        ulCRC = ulTemp1 ^ ulTemp2;
    }
    return (ulCRC);
}
```

10.2 Original IMU packet

"imu"			
Offset	Variable type	Name	Description
0	OpenRTKPacketHeader	header	header
28	uint32	gps_week	GPS week
32	double	gps_millisecs	GPS seconds within a week (ms)
40	uint32	imuStatus	N/A
44	float	z_acceleration	Accelerometer data on z-axis, y-axis, x-axis (g)
48	float	y_acceleration	
52	float	x_acceleration	
56	float	z_gyro_rate	Gyroscope data on z-axis, y-axis, x-axis (rad/s)
60	float	y_gyro_rate_neg	
64	float	x_gyro_rate	
68	int8 * 4	crc[4]	check code

10.3 GNSS position solution

"gnss"			
Offset	Variable type	Name	Description
0	OpenRTK-PacketHeader	header	header
28	uint32	solution_status	N/A
32	uint32	position_type	Positioning mode: 0: Invalid 1: Single point solution 4: Fixed solution 5: Floating point solution
36	double	latitude	longitude (deg)
44	double	longitude	Latitude (deg)
52	double	height	Altitude (m)
60	float	undulation	N/A
64	uint32	datum_id	Geodetic datum coordinate system
68	float	longitude_standard_deviation	Longitude standard deviation
72	float	latitude_standard_deviation	Latitude standard deviation
76	float	height_standard_deviation	height standard deviation
80	int8 * 4	base_station_id[4]	N/A
84	float	differential_age	N/A
88	float	solution_age	
92	uint8	number_of_satellites	The number of satellites used in the positioning solution
93	uint8	number_of_satellites_in_solution	N/A
94	uint8	num_gps_plus_glonass_l1	N/A
95	uint8	num_gps_plus_glonass_l2	N/A
96	uint8	reserved	N/A
97	uint8	extended_solution_status	N/A
98	uint8	reserved2	N/A
99	uint8	signals_used_mask	N/A
100	int8 * 4	crc[4]	check code

10.4 GNSS velocity solution

"vel"			
Offset	Variable type	Name	Description
0	OpenRTKPacketHeader	header	header
28	uint32	solution_status	N/A
32	uint32	position_type	N/A
36	float	latency	N/A
40	float	age	N/A
44	double	horizontal_speed	Horizontal speed (m/s)
52	double	track_over_ground	Ground speed (m/s)
60	double	vertical_speed	Vertical speed (m/s)
68	float	reserved	N/A
72	int8 * 4	crc[4]	check code

10.5 INS position, velocity and attitude solution

"ins"			
Offset	Variable type	Name	Description
0	OpenRTKPacketHeader	header	header
28	uint32	gps_week	GPS week
32	double	gps_millisecs	GPS seconds within a week (ms)
40	double	latitude	Latitude (deg)
48	double	longitude	Longitude (deg)
56	double	height	Height (m)
64	double	north_velocity	Velocity (north) (m/s)
72	double	east_velocity	Velocity (East) (m/s)
80	double	up_velocity	Velocity (up) (m/s)
88	double	roll	Roll angle (deg)
96	double	pitch	Pitch angle (deg)
104	double	azimuth	Yaw angle (deg)
112	int32	status	Combined solution status: 0: invalid 1: INS alignment ongoing 2: INS solution is unreliable 3: INS solution is good 4 :INS free(no GNSS update) 5: Estimating installation angle 6: Completed estima installation angle estimation
116	int8 * 4	crc[4]	check code

CHAPTER 11

Port command

11.1 Get module configuration information

Command: **get configuration\r\n**

Return: string in json format


```
{
        "openrtk configuration":
    {
                "Product Name":          "",
                "Product PN":            "",
                "Product SN":            "",
                "Version":                   "",
                "userPacketType":        "s1",
                "userPacketRate":        100,
                "leverArmBx":            0.0,
                "leverArmBy":            0.0,
                "leverArmBz":            0.0,
                "pointOfInterestBx":     0.0,
                "pointOfInterestBy":     0.0,
                "pointOfInterestBz":     0.0,
                "rotationRbvx":          0,
                "rotationRbvy":          0,
                "rotationRbvz":          0
        }
}
```


At the same time, the module will close the P1 packet output of the DEBUG port.

11.2 Enable P1 packet output

Command: **log debug on\r\n**

Return: N/A, the module will directly output P1 packet data after a delay of 1 second.

CHAPTER 12

CAN Interface Data Protocol

12.1 CAN port settings

- Save parameters

Frame type	Save parameters				
Description	Save user parameters, no loss after power failure				
Set frame		**PF**	**ps**	**PGN**	**Data content length**
		255	**81**	**65361**	**3**
Data content:					
Byte	Description		Value		
0	Frame type		0: Request frame 1: Reply frame		
1	Destination address				
2	Reply frame is valid		0: Save failed 1: Save successfully		

- Set CAN packet type

Frame type	Set CAN data packet type				
Description	Set the type of CAN data sent cyclically				
Set frame	**PF**	**PS**	**PGN**	**Data content length**	
	255	**86**	**65366**	**3**	
Data content:					
Byte	Description		Value		
0	destination address				
1	Data packet type (low byte)		0x01-accelerometer 0x02-Gyroscope 0x04-latitude and longitude 0x08-attitude Note: The package can be sent together, such as 0x03, both accelerometer and gyroscope		
2	Packet type (high byte)				

- Set CAN data frequency

Frame type	Set CAN data frequency		
Description	Set the frequency of CAN data sent cyclically		
Set frame	**PF**	**PS**	**PGN**	**Data content length**
	255	**85**	**65365**	**2**
Data content:				
Byte	Description	Value		
0	destination address			
1	Data frequency	0-Quiet Mode 1-100(default) 2-50 4-25 5-20 10(0x0a)-10 20(0x14)-5 25(0x19)-4 50(0x32)-2		

12.2 Get parameters through CAN port

Frame type	Get parameters			
Description	Get specified parameters			
Get frame	**PF**	**PS**	**PGN**	**Data content length**
	234	**255**	**60159**	**3**
Data content:				
Byte	Description	Value		
0	N/A			
1	Parameter PF	PF and PS of specific parameters, see below 2 parameter PS		
2	Parameter PS			

- Get the software version number

Frame type	Get software version number			
Description	Get specified parameters			
Get frame	**PF**	**PS**	**PGN**	**Data content length**
	254	**218**	**65242**	**5**
Data content:				
Byte	Description	Value		
0	Major Version Number			
1	Minor Version Number			
2	Patch Number			
3	Stage Number			
4	Build Number			

- Get ECU ID

Frame type	Get ECU ID			
Description	Get specified parameters			
Get frame	**PF**	**PS**	**PGN**	**Data content length**
	253	**197**	**64965**	**8**
Data content:				

Bits	Description	Value
bits 0	Arbitrary Address	Arbitrary Address
bits 1:3	Industry Group	Industry Group
bits 4:7	Vehicle System Instance	Vehicle System Instance
bits 8:14	System Bits	System Bits Vehicle system domain
bits 15	Reserved	Reserved Reserved
bits 16:23	Function Bits	Function Bits Function domain
bits 24:28	Function Instance	Function Instance
bits 29:31	ECU Bits	ECU Bits ECU instance domain
bits 32:42	Manufacturer code	Manufacturer code Manufacturer code field
bits 43:63	ID bits	ID bits number

- Get CAN packet type

Frame type	Get CAN data packet type			
Description				
Get frame	**PF**	**PS**	**PGN**	**Data content length**
	225	**86**	**65366**	**3**
Data content:				

Byte	Description	Value
0	destination address	
1	Packet type (low byte)	
2	Packet type (high byte)	

- Get CAN data frequency

Frame type	Get CAN data frequency			
Description				
Get frame	**PF**	**PS**	**PGN**	**Data content length**
	255	**85**	**65365**	**2**
Data content:				

Byte	Description	Value
0	destination address	
1	Data frequency	

- Latitude and longitude position

Frame type	Latitude and longitude position			
Description				
Data frame	**PF**	**PS**	**PGN**	**Data content length**
	254	**243**	**65267**	**8**
Data content:				

Byte	Description	Value
0:3	Latitude	0.0000001 deg/bit
4:7	Longitude	0.0000001 deg/bit

- Attitude

Frame type	Attitude				
Description					
Data frame	**PF**	**PS**	**PGN**	**Data content length**	
	241	**25**	**127257**	**8**	
Data content:					
Byte	Description	Value			
0	SID				
1:2	yaw angle	0.0001 rad/bit			
3:4	pitch angle	0.0001 rad/bit			
5:6	roll angle	0.0001 rad/bit			
7	Latitude				

- Accelerometer data

Frame type	Accelerometer data			
Description				
Data frame	**PF**	**PS**	**PGN**	**Data content length**
	240	**45**	**61485**	**8**
Data content:				
Byte	Description	Value		
0:1	Accelerometer x axis	0.01 m/s**2/bit	-320 m/s**2	
2:3	Accelerometer y axis	0.01 m/s**2/bit	-320 m/s**2	
4:5	Accelerometer z axis	0.01 m/s**2/bit	-320 m/s**2	
6:7	reserved			

- Gyroscope data

Frame type	Gyroscope data			
Description				
Data frame	**PF**	**PS**	**PGN**	**Data content length**
	240	**42**	**61482**	**8**
Data content:				
Byte	Description	Value		
0:1	gyroscope x axis	1/128 deg/second/bit	-250 deg	
2:3	gyroscope y axis	1/128 deg/second/bit	-250 deg	
4:5	gyroscope z axis	1/128 deg/second/bit	-250 deg	
6:7	reserved			

CHAPTER 13

NMEA

$GNGGA

Format: $GNGGA,<1>,<2>,<3>,<4>,<5>,<6>,<7>,<8>,<9>,M,<10>,M,< 11>,<12>*xx<CR><LF> E.g: $GNGGA,072446.00,3130.5226316,N,12024.0937010,E,4,27,0.5,31.924,M,0.000,M,2.0,*44 Field explanation:

- **<0>** $GNGGA
- **<1>** UTC time, the format is hhmmss.sss
- **<2>** Latitude, the format is ddmm.mmmmmmm
- **<3>** Latitude hemisphere, N or S (north latitude or south latitude)
- **<4>** Longitude, the format is dddmm.mmmmmmm
- **<5>** Longitude hemisphere, E or W (east longitude or west longitude)
- **<6>** GNSS positioning status: 0 not positioned, 1 single point positioning, 2 differential GPS fixed solution, 4 fixed solution, 5 floating point solution
- **<7>** Number of satellites used
- **<8>** HDOP level precision factor
- **<9>** Altitude
- **<10>** The height of the earth ellipsoid relative to the geoid
- **<11>** Differential time
- **<12>** Differential reference base station label
- ***** Statement end marker
- **xx** XOR check value of all bytes starting from $ to *
- **<CR>** Carriage return, end tag
- **<LF>** line feed, end tag

$GNRMC

Format: $GNRMC,<1>,<2>,<3>,<4>,<5>,<6>,<7>,<8>,<9>,<10>,<11>,< 12>*xx<CR><LF> E.g: $GNRMC,072446.00,A,3130.5226316,N,12024.0937010,E,0.01,0.00,040620,0.0,E,D*3D Field explanation:

- **<0>** $GNRMC
- **<1>** UTC time, the format is hhmmss.sss
- **<2>** Positioning status, A=effective positioning, V=invalid positioning
- **<3>** Latitude, the format is ddmm.mmmmmmm
- **<4>** Latitude hemisphere, N or S (north latitude or south latitude)
- **<5>** Longitude, the format is dddmm.mmmmmmm
- **<6>** Longitude hemisphere, E or W (east longitude or west longitude)
- **<7>** Ground speed
- **<8>** Ground heading (take true north as the reference datum)
- **<9>** UTC date, the format is ddmmyy (day, month, year)
- **<10>** Magnetic declination (000.0~180.0 degrees)
- **<11>** Magnetic declination direction, E (east) or W (west)
- **<12>** Mode indication (A=autonomous positioning, D=differential, E=estimation, N=invalid data)
- * Statement end marker
- **XX** XOR check value of all bytes starting from $ to *
- **<CR>** Carriage return, end tag
- **<LF>** line feed, end tag

$GNGSA

format: $GNGSA,<1>,<2>,<3>,<3>,,,,,<3>,<3>,<3>,<4>,<5>,<6>,<7> *xx<CR><LF> E.g: $GNGSA,A,3,03,06,09,17,19,23,28,,,,,,3.0,1.5,2.6,1*25 $GNGSA,A,3,65,66,67,81,82,88,,,,,,,2.4,1.3,2.1,2*36 $GNGSA,A,3,02,05,09,15,27,,,,,,,,10.8,2.7,10.4,3*3A $GNGSA,A,3,01,02,07,08,10,13,27,28,32,33,37,,2.1,1.0,1.9,5*33 Field explanation:

- **<1>** Mode: M=Manual, A=Auto
- **<2>** Positioning type: 1=not positioned, 2=two-dimensional positioning, 3=three-dimensional positioning
- **<3>** PRN code (Pseudo Random Noise Code), channels 1 to 12, up to 12
- **<4>** PDOP position precision factor
- **<5>** HDOP level precision factor
- **<6>** VDOP vertical precision factor
- **<7>** GNSS system ID: 1(GPS), 2(GLONASS), 3(GALILEO), 5(BEIDOU)
- * Statement end marker
- **xx** XOR check value of all bytes starting from $ to *
- **<CR>** Carriage return, end tag
- **<LF>** line feed, end tag

Part VI

RTKlib tools

CHAPTER 14

Overview

14.1 What is RTKlib

RTKLIB is an open source program package for standard and precise positioning with GNSS (global navigation satellite system). It supports standard and precise positioning algorithms with GPS, GLONASS, Galileo, QZSS, BeiDou and SBAS.

14.2 RTKlib tools supporting Aceinna Format

RTKlib tools supporting Aceinna Format is s special version of RTKlib which supports aceinna data format to display data, decode data, save data, and also plotting and RTK processing.

RTKLIB_with Aceinna format binary version: https://github.com/Aceinna/rtklib_bin_aceinna

RTKLIB_with Aceinna format_source version: https://github.com/Aceinna/rtklib_aceinna

14.3 Aceinna data format

Aceinna-user and aceinna-raw are two data formats exported from Openrtk330LI; They are output from serial port 1 and serial port 3 of Openrtk330LI; Aceinna-user data include imu raw data, rtk and ins solution; Aceinna-raw includes rover, base RTCM data and imu raw data.

14.3.1 Aceinna-User Format

Data format definition

Start 1	Start 2	Frame type 1	Frame type 2	Data length 1	Data content	Check 1	Check 2

Description

- Start: Each frame of data starts with this, 2 bytes: 0x55 0x55.
- Frame type: 2 bytes, high byte first.
- Data length: 1 byte, refers to the byte length of the data content.
- Data content: maximum 255 bytes.
- Check: crc16 check, 2 bytes, low byte first, bytes from the beginning of the "Frame type" to the end of the "Data content" are included in the check calculation, and the check algorithm C code is as follows:

```
uint16_t CalculateCRC (uint8_t *buf, uint16_t  length)
{
   uint16_t crc = 0x1D0F;

   for (int i=0; i < length; i++) {
       crc ^= buf[i] << 8;
       for (int j=0; j<8; j++) {
           if (crc & 0x8000) {
               crc = (crc << 1) ^ 0x1021;
           }
           else {
               crc = crc << 1;
           }
       }
   }

   return ((crc << 8 ) & 0xFF00) | ((crc >> 8) & 0xFF);
}
```

Frame types

Aceinna-user has five types of data, namely "S1", "G1", "I1", "O1" and "Y1"; For the specific structure of each type of format, please refer to the openrtk documentation https://openrtk.readthedocs.io/en/latest/communication_port/User_uart.html#imu-raw-data-packet

14.3.2 Aceinna-raw Format

Aceianna-raw is composed of four format types of $GPGGA$GPIMU$GPROV$GPREF;

$GPGGA

$GPGGA is the standard NMEA GGA format.

$GPIMU

$GPIMU is the IMU information in NMEA format.

$GPIMU	time of week	accel-x	accel-y	accel-z	gyro-x	gyro-y	gyro-z

$GPROV

$GPROV contains the RTCM package from Rover.

$GPROV	time of week	left length	RTCM bin	

$GPREF

$GPREF contains the RTCM package from Base.

$GPRRF	time of week	left length	RTCM bin	

CHAPTER 15

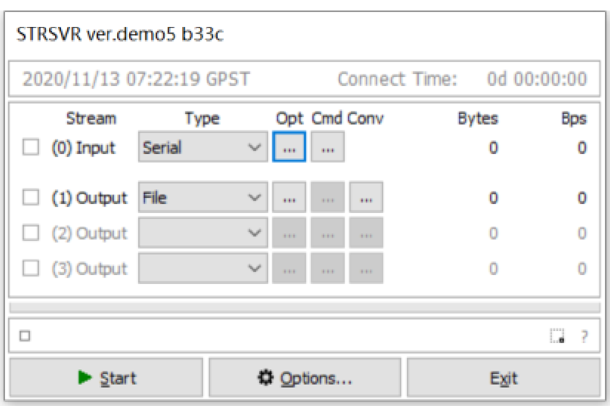
Instructions

15.1 Use strsvr to decode aceinna-user data

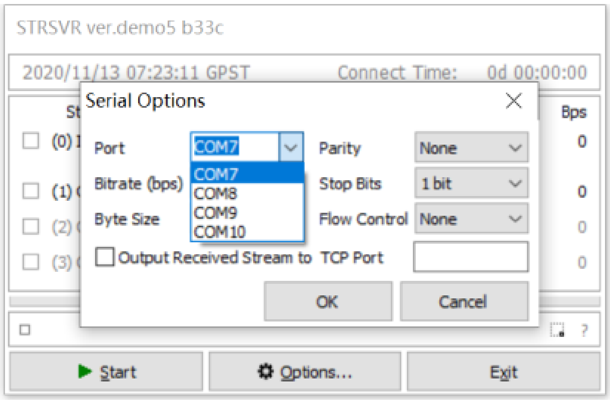
Use strsvr to decode aceinna-user data format. Decode aceinna format data and display information in monitor dialog and save it in files.

15.1.1 Set input stream parameter

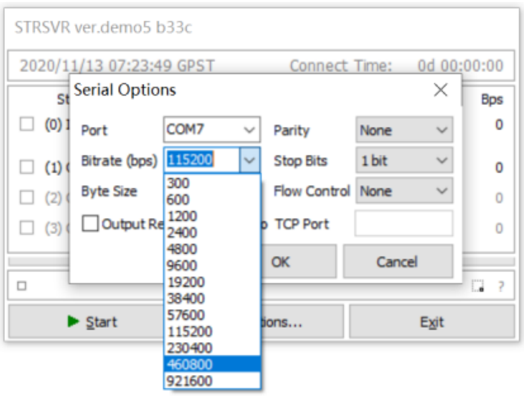
Select serial for (0) input. Click "opt" button to open the Serial Options dialog.



Select the first serial port in the serial Options dialog.

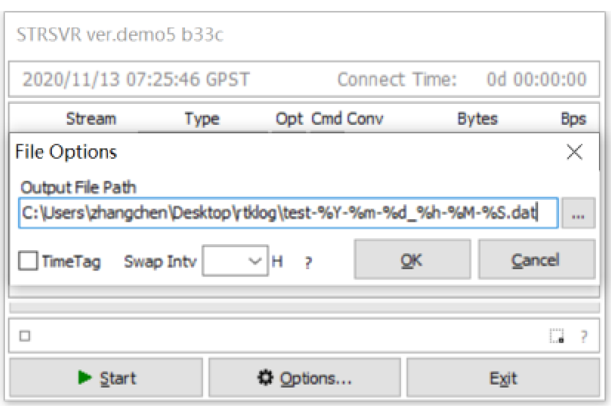


Bitrate is selected as 460800.



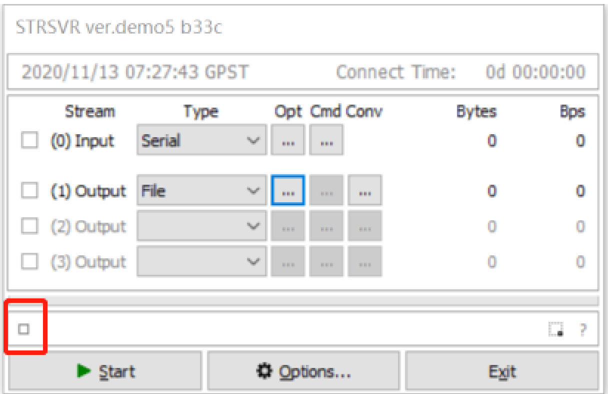
15.1.2 Set output files path

Select the path to save the file. For example: C:/Users/zhangchen/Desktop/rtklog/.

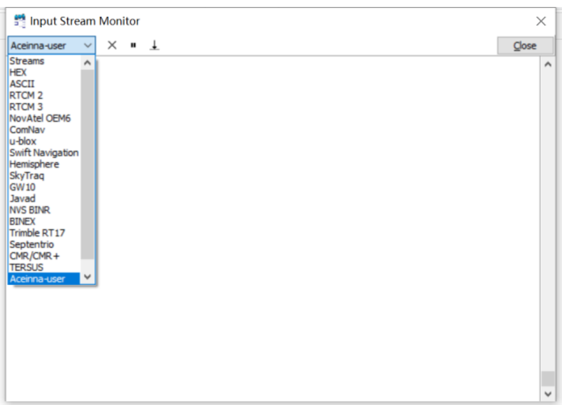


15.1.3 Show the data in monitor dialog and save file

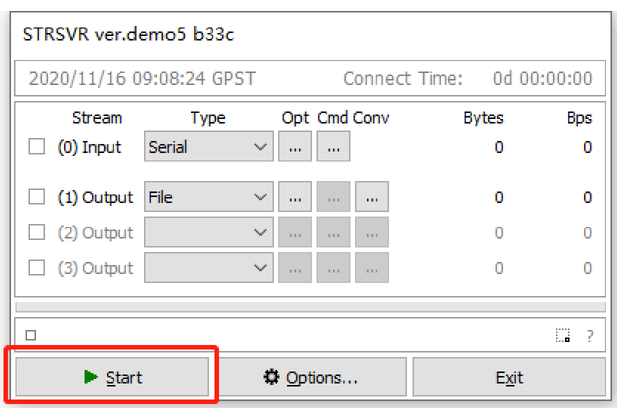
Click the small square button to open Input Stream Monitor dialog.



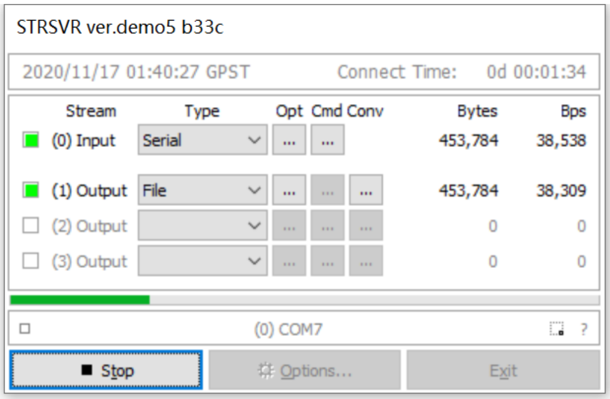
Select the aceinna-user format.



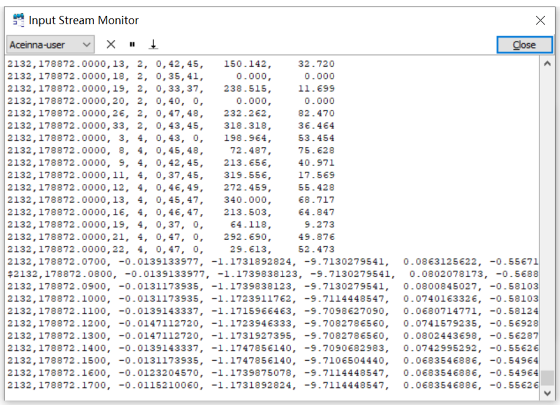
Click "start" button to start receiving data.



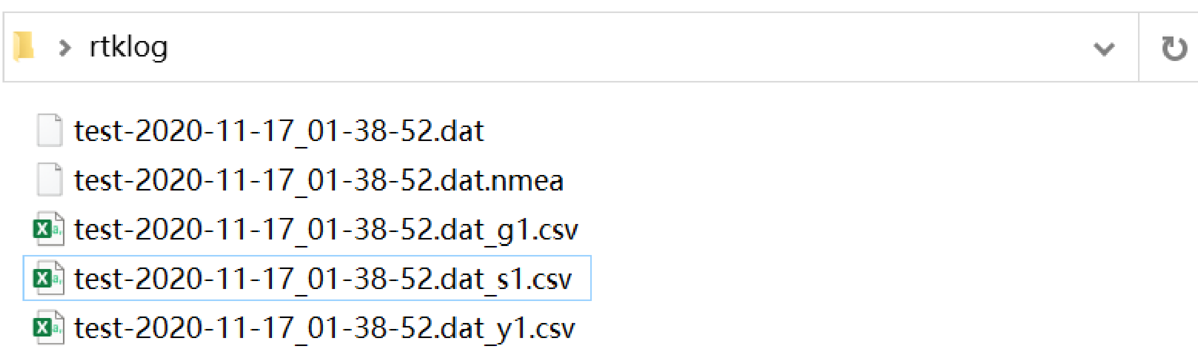
Strsvr is running.



The data decoding information is showed in monitor dialog.

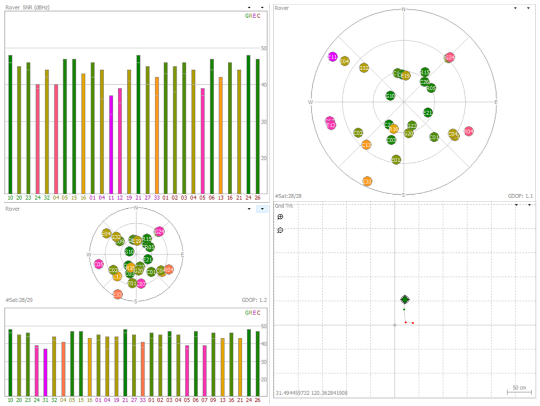


The file is saved in the previous output path.



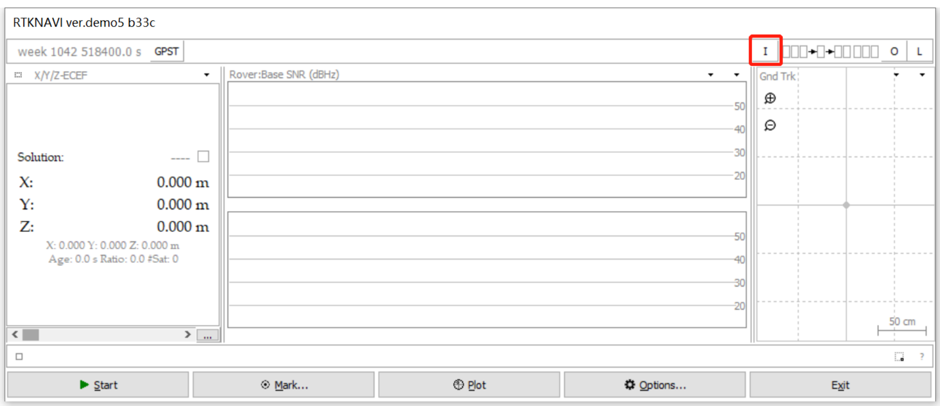
15.2 Use RTKLIBNAVI to decode aceinna-user data

Aceinna-raw data is the result outputfrom OpenRTK330. Using rtklibnavi to connect the first serial port of openrtk330, the RTK processing result data can be recognized These data can be displayed by SNR plot, sky map and GND Trk.

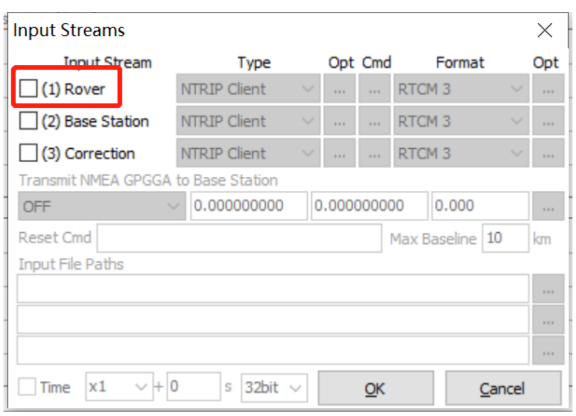


15.2.1 Set input stream parameter

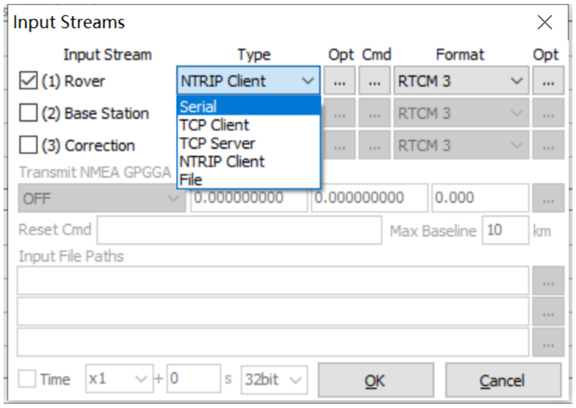
Click the ‘I’ button to open Input Streams dialog.



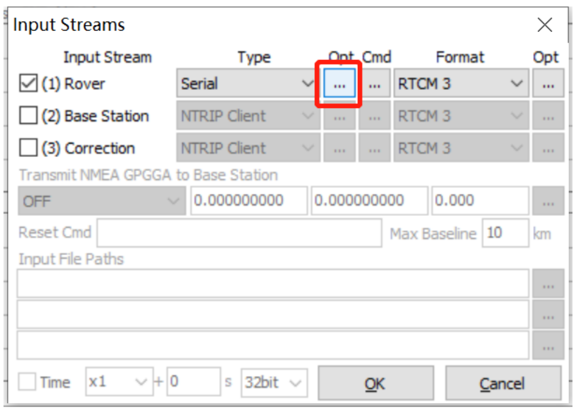
Check (1) Rover in the Input Streams dialog.



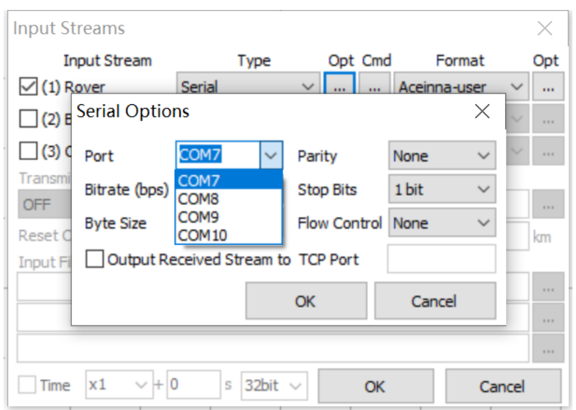
Select “serial” in the type option.



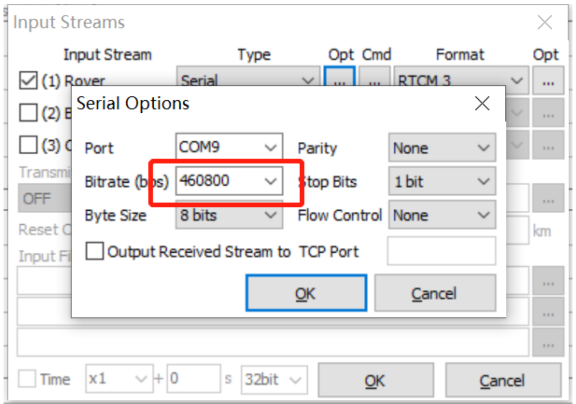
Click "opt" button to open the Serial Options dialog.



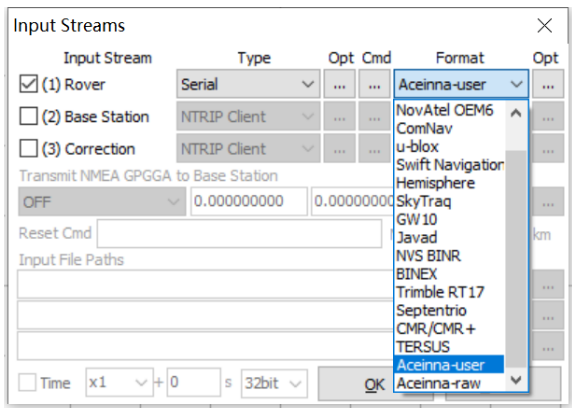
Select the frist serial port in the serial Options dialog.



Bitrate is selected as 460800.



Format is selected as Aceinna-raw.

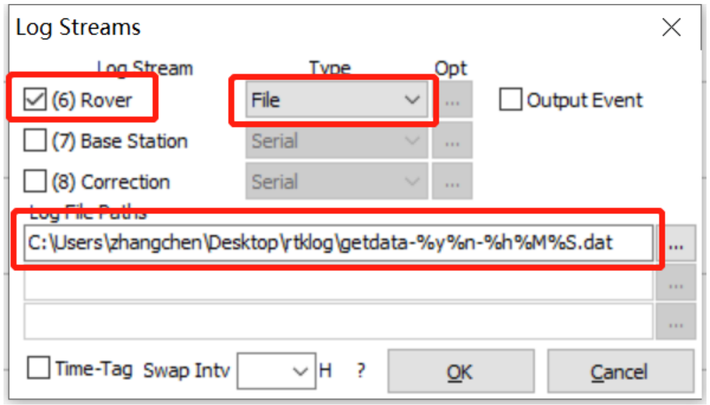


15.2.2 Set output log files path

Select the path to save the file. For example: C:/Users/zhangchen/Desktop/rtklog/.

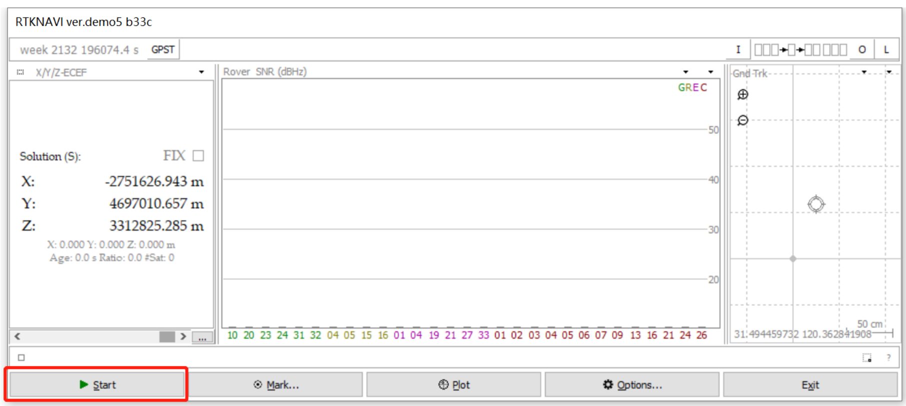
Click the 'L' button to open Log Streams dialog.

Check (6) Rover ,select File type and input the log file paths. Click "OK" button.

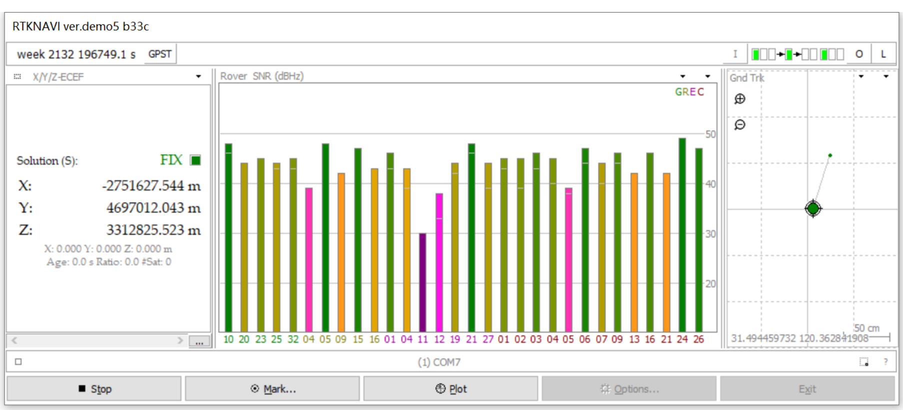


15.2.3 Start to receive data

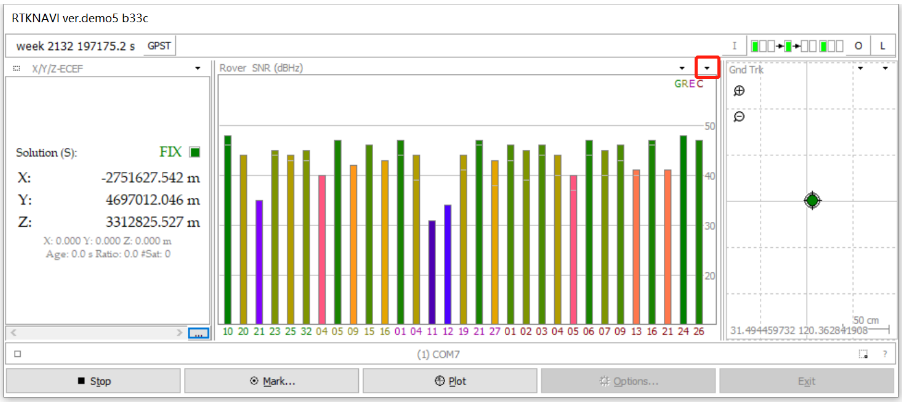
Click the "start" button to start receiving the data.



When receiving the data, the SNR bar is plotted.



Click the arrow button to switch view (SNR bar, sky map, positioning coordinates, horizontal error scatter, position error timeseries in north, east and up).



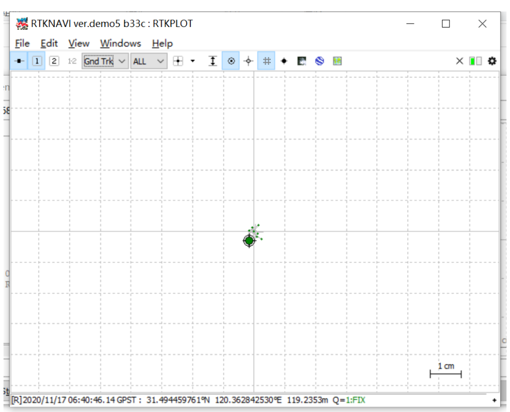
The sky map.

Both sky map and SNR plot.

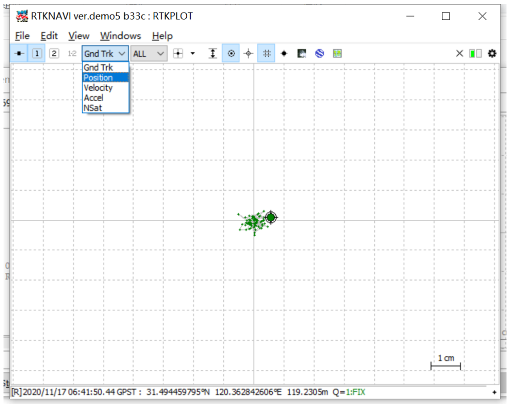
The Gnd Trk.

Click the "Plot" button to open RTKPLOT.

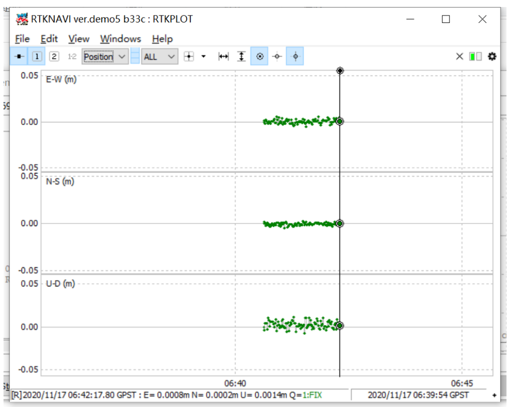
The RTKPLOT dialog.



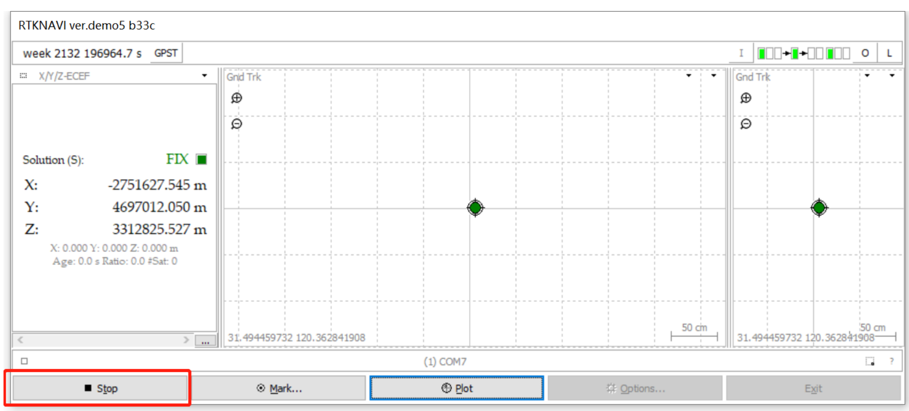
Select the drop-down list to switch views.



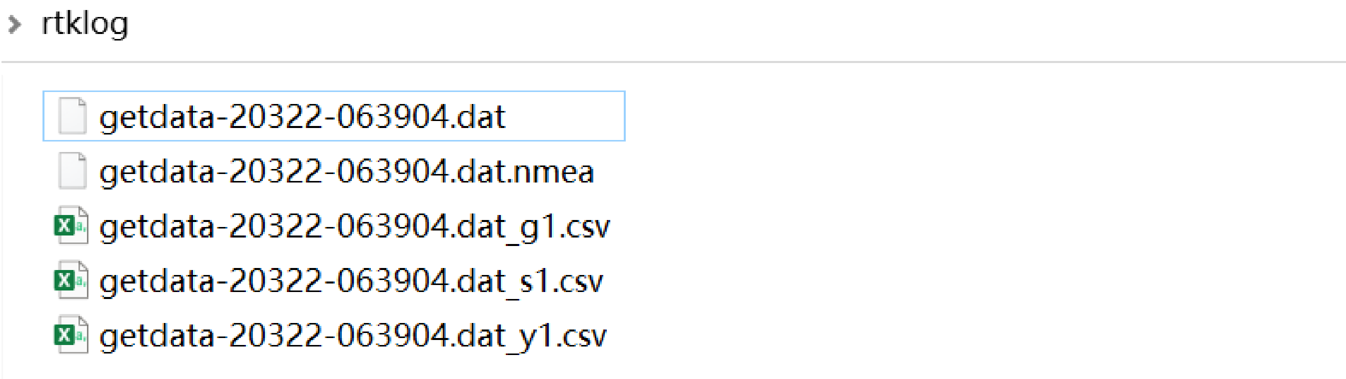
The Position views.



Click "stop" button to stop receiving data.

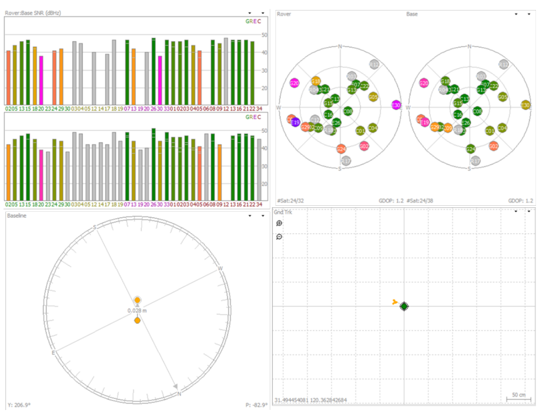


The file is saved in the previous output path.



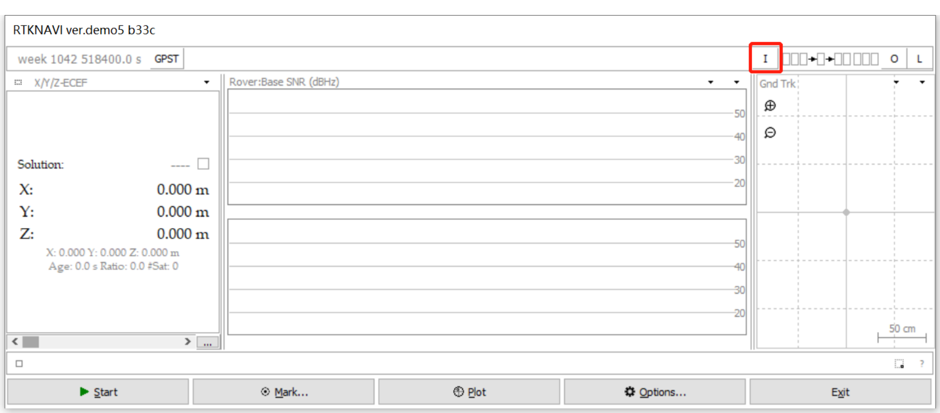
15.3 Use RTKLIBNAVI to decode aceinna-raw data

Aceinna-raw data contains the original data of rover station and base station. Using rtklibnavi to connect the third serial port of openrtk330, the rover station and the base station information can be read at the same time. These data can be displayed by SNR plot, sky map, baseline and GND Trk. At the same time, these data can also be used for RTK processing.

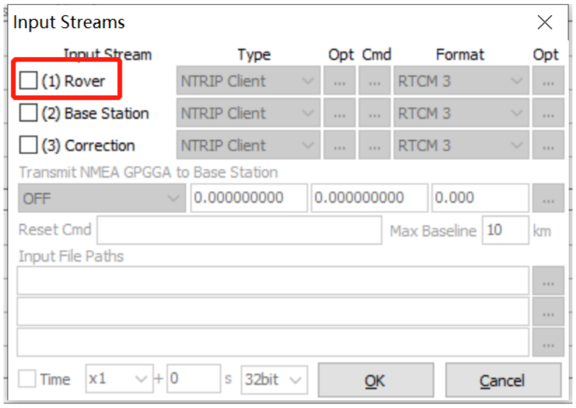


15.3.1 Set input stream parameter

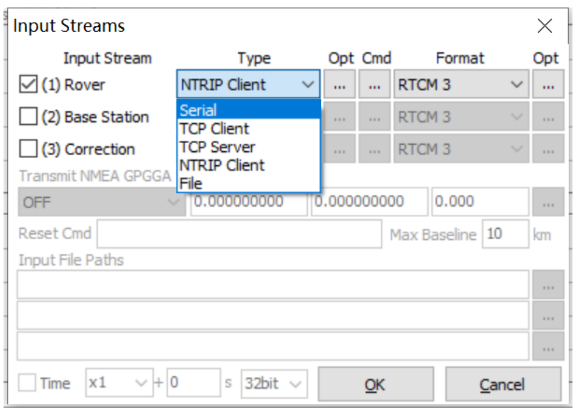
Click the 'I' button to open Input Streams dialog.



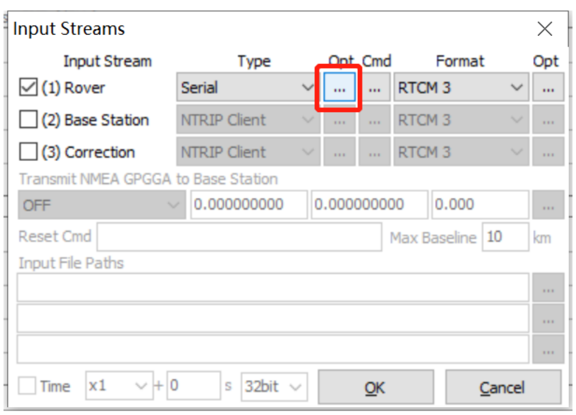
Check (1) Rover in the Input Streams dialog.



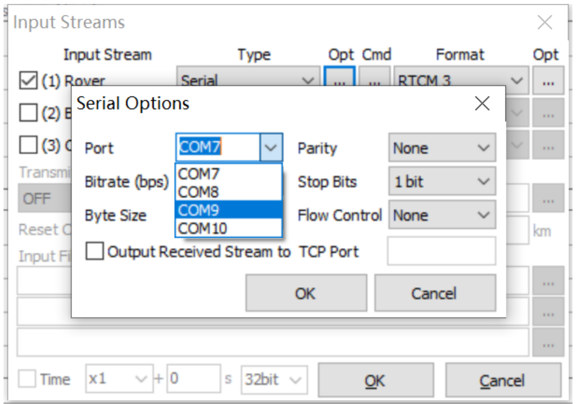
Select serial in the type option.



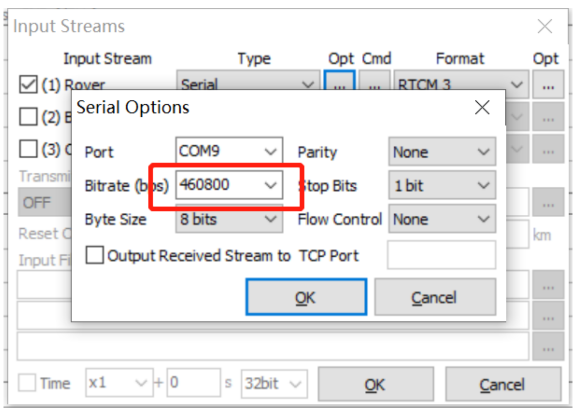
Click "opt" button to open the Serial Options dialog.



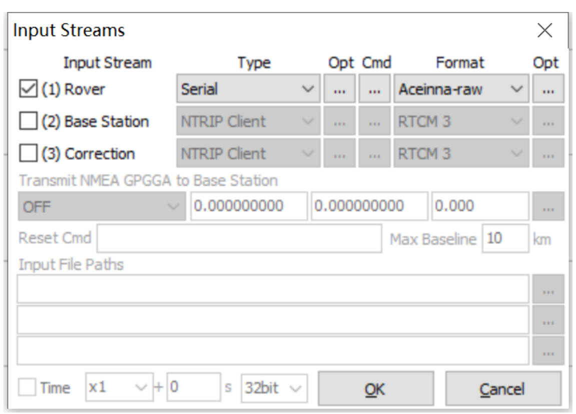
Select the third serial port in the serial Options dialog.



Bitrate is selected as 460800.

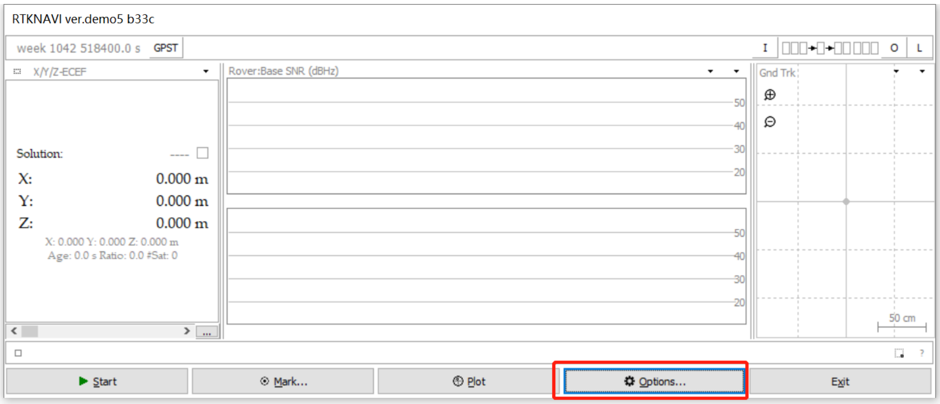


Format is selected as Aceinna-raw.

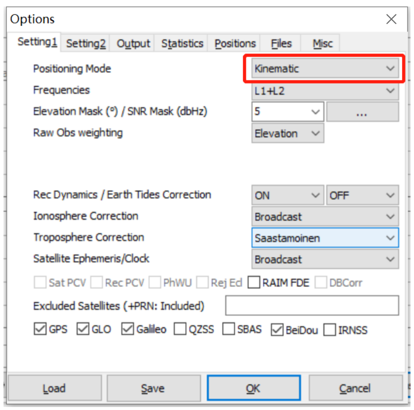


15.3.2 RTK processing config

Close the Input Streams dialog and click the “options” button to open the options dialog.

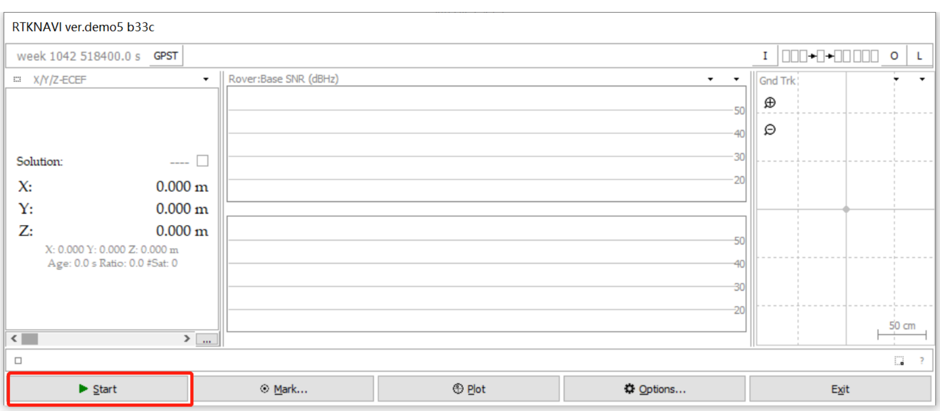


In the options dialog, choose the RTK posting mode option as "kinematic" or "static".

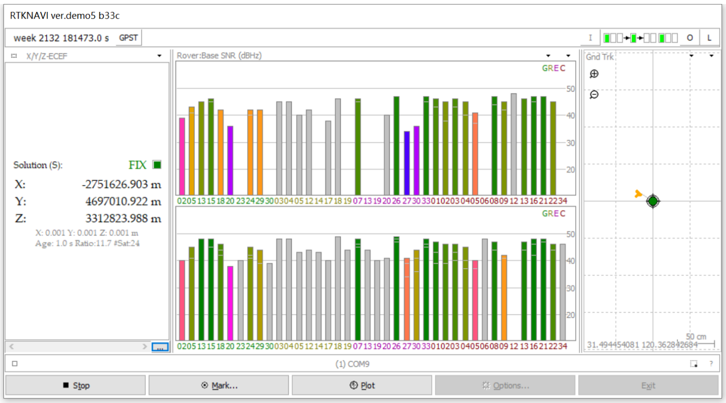


15.3.3 Start to receive data

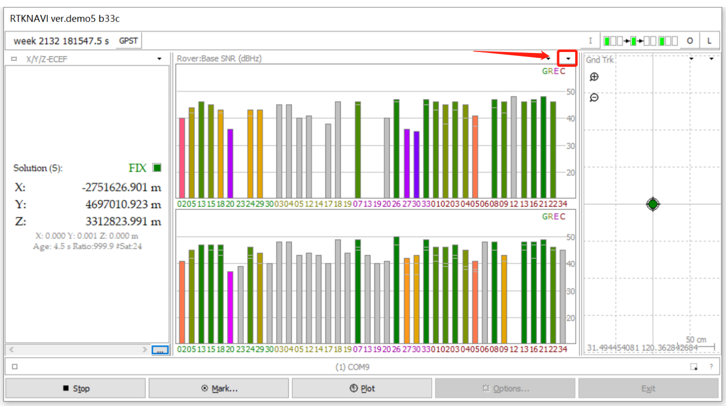
Click "start" button to start receiving the data.



When receiving the data, the SNR map of Rover and base according to the data will appear in GUI, and RTK results will be displayed.

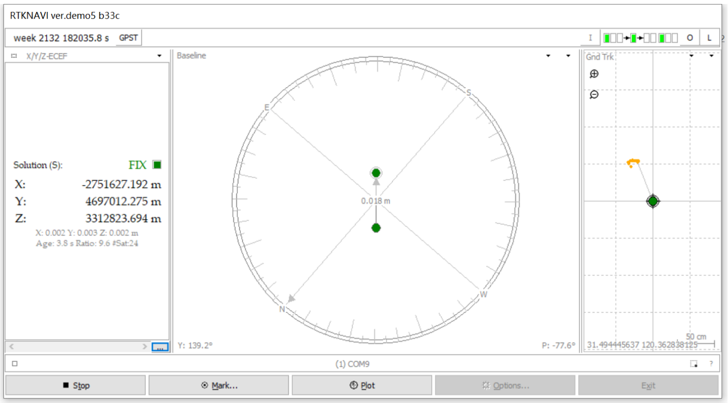


Click the arrow button to switch view (SNR bar, sky map, positioning coordinates, horizontal error scatter, position error timeseries in north, east and up).

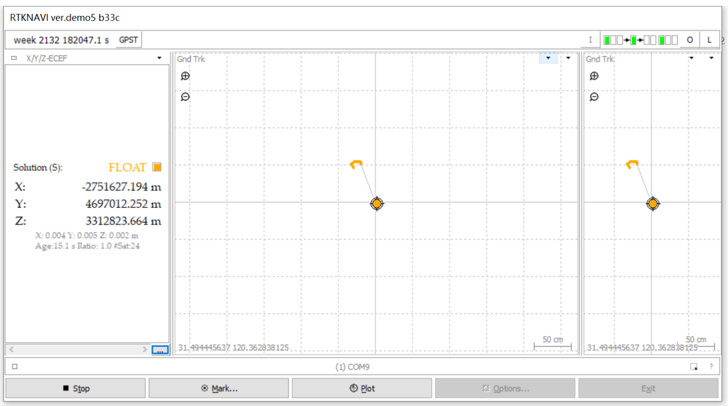


The sky maps.

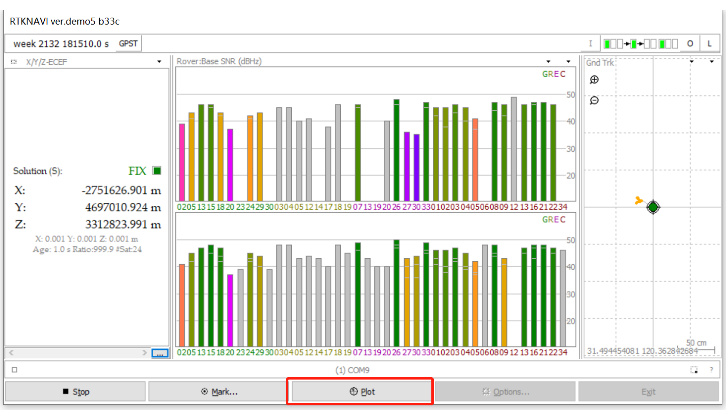
The baseline.



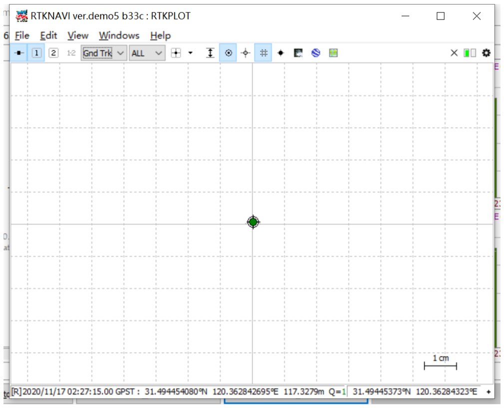
The Gnd Trk.



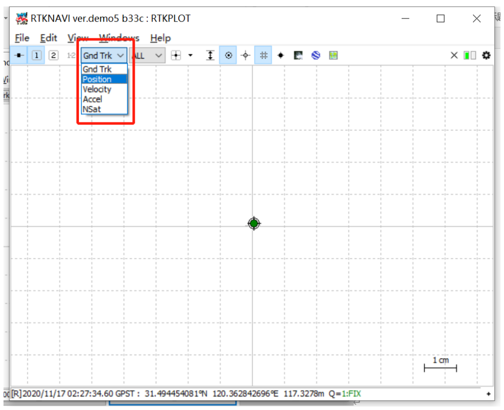
Click "Plot" button to Open RTKPLOT.



The RTKPLOT dialog.



Select the drop-down list to switch views.



The Position views.