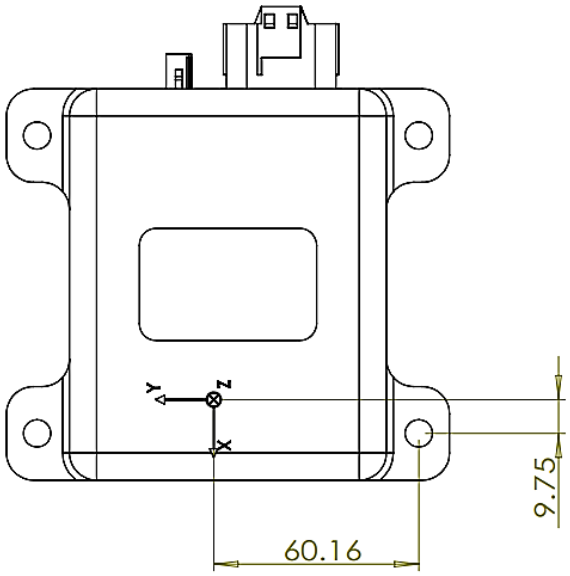
How to configure INS401 when mounting in vehicle?



Actual mounting in vehicle

We need to calculated 3 group parameters based on actual mounting condition, include: orientation, antenna position, interesting point position (location point which will be output in INS401 position message), vehicle reference point position (position of odometer used, usually it is center of 2 rear wheels).
Then update configurations based on the 3 group parameters.
There 2 ways to update the configurations.

Orientation

We recommend the mounting position of INS401 is that X/Y/Z of IMU is same with Front/Right/Down direction of vehicle, pls refer to User Manual (D04 Version).
In current project, followed Aceinna recommendation and **no need to update the rotation parameters**.

Target values based on mounting condition

Based on customer input, we calculate/measured other lever arm parameters based on vehicle coordinate system (X-forward, Y-right, Z-Down), respect to origin of INS401.

Target values as below: Unit is meter(m)

- the coordinates of antenna
  [-2.036, -0.386, -0.938]
- the coordinates of vehicle reference point for odometer
  [-1.684, -0.004, 0.241]
- the coordinates of interesting point (use **odometer reference position** as target interesting point position firstly)
  [-1.684, -0.004, 0.241]

First Method：Use acenav.exe to update configurations

Preparation:

By ethernet transfer board (100base-t1 to tx) or similar instrument, INS401 should be connected with PC by network cables.

Then pls use Aceinna official SW: acenave.exe,
acenav download link (no need to install):
https://github.com/Aceinna/acenav-cli/releases/download/v2.6.3/Windows.zip
scapy download link (need to be installed):
https://navview.blob.core.windows.net/forum/upload/npcap-1.55.exe-kv7fw14u.zip

Steps:

1st Step-Set new value in JSON file

open ‘.\setting\INS401\RTK_INS\ins401.json’, Change the 'value' of id 1-9 in the JSON file, then save and close the file.


```
"userParameters": [
        {
            "paramId": 1,
            "name": "gnss lever arm x",
            "value": -2.036
        },
        {
            "paramId": 2,
```

```
        "name": "gnss lever arm y",
        "value": -0.386
    },
    {
        "paramId": 3,
        "name": "gnss lever arm z",
        "value": -0.938
    },
    {
        "paramId": 4,
        "name": "vrp lever arm x",
        "value": -1.684
    },
    {
        "paramId": 5,
        "name": "vrp lever arm y",
        "value": -0.004
    },
    {
        "paramId": 6,
        "name": "vrp lever arm z",
        "value": 0.241
    },
    {
        "paramId": 7,
        "name": "user lever arm x",
        "value": -1.684
    },
    {
        "paramId": 8,
        "name": "user lever arm y",
        "value": -0.004
    },
    {
        "paramId": 9,
        "name": "user lever arm z",
        "value": 0.241
    },
]
```


2nd Step-Running acenav.exe with parameter input

Run acenav.exe by PowerShell in the same folder, with the command(cmd): '.\acenav.exe -i 100base -t1 -s' (see 1 in Figure 1)'

After running the command, “Success (12)” will be received as Figure 1, which means new parameters are saved.

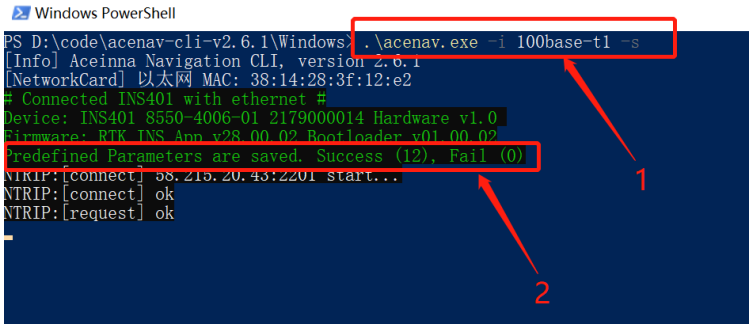


Fig1. Configuration Process (will updated FW28.02 later)

Close the window of PowerShell, and power reset INS401.

3rd Step-Check current configuration values after power reset

Running acenav.exe directly and the device will be connected again as Figure 2, FW and Bootloader version is printed.

Open ‘.\data\ins401_log_20211026_101911\configuration.json’ to check the current configurations, like Figure 3 and know that all new values are configured to INS401.

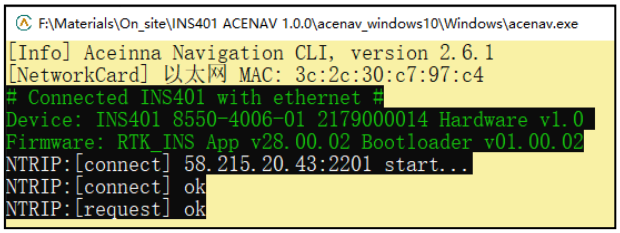


Fig2. Check Configuration (will update FW28.02 later)

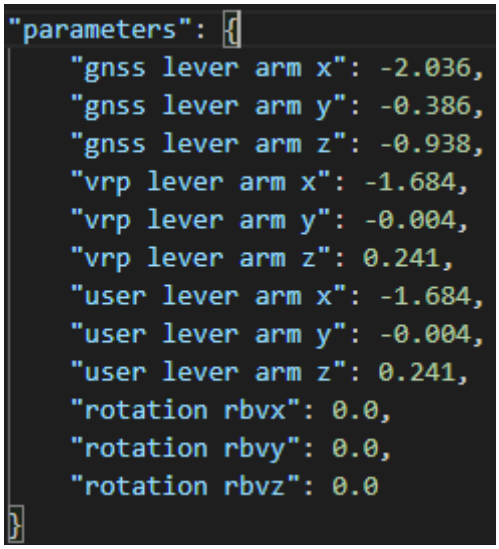


Fig3. Check Current Configuration

Second Method：Use ethernet cmd to update configurations

Preparation:

INS401 should be connected by our controller with 100base-t1 protocol communication.
The INS401 ethernet TX/RX data frame conforms to IEEE802.3, the format is shown below. The binary packets and ASCII messages should be filled in user data. Note it is also designed to contain two packets/messages in the same one data frame. If one packet/message is more than 1500 bytes, it is split into two continuous data frames.

Table3. Ethernet Data Frame Definition

Destination Address	Source Address	Length	User Data	Data Frame Checksum
6 bytes	6 bytes	2 bytes	46 to 1500 bytes	4 bytes

- Destination Address: destination MAC address
- Source Address: source MAC address
- Length: length of user data (0~1500), MSB for output message from INS401 to ECU or the 100Base-TX/100Base-T1 converter, LSB for input message.
- **User Data**: Min/Max data length is 46/1500. Zero bytes will be filled if less than 46 bytes
- Data Frame Checksum: CRC checksum bytes

Table4. Packet format of sending cmd

Header (uint16)	Message ID (uint16)	Length (uint32)	Payload	Checksum (uint16)

- Header: packet starts with header bytes 0x5555
- Message ID: the ID of the current message, LSB-first
- Length: the number of bytes in the payload, LSB-first
- Payload: user data contents, LSB-first
- Checksum: CRC16 check. Bytes from the beginning of the “Message ID” to the end of the “Data content” are included in the checksum calculation, and a sample of the checksum algorithm C code is shown in Appendix A. Binary data of CRC is in Little Endian format

Steps:

You can set the parameters by sending commands (it is User Data in Ethernet Data Frame) to INS401 in 100base-t1 protocol communication in your controller which is connected with INS401, steps and cmd in below.

1st Step-Get target device information

Send 'Get Command 0xcc01':

55 55 01 CC 00 00 00 00 CC 0d

(Ethernet Data Frame: **ff ff ff ff ff ff 38 14 28 3f 12 e2** 0a 00 55 55 01 cc 00 00 00 00 cc 0d, **'38 14 28 3f 12 e2'** is an example of PC MAC address)

This command is used to get device info of INS401

Response message: (take the unit in my hands as example)

55 55 01 CC 57 00 00 00 49 4E 53 34 30 31 20 38 35 35 30 2D 34 30 30 36 2D 30 31 20 32 31 37 39 30 30 30 30 31 34 20 48 61 72 64 77 61 72 65 20 76 31 2E 30 20 52 54 4B 5F 49 4E 53 20 41 70 70 20 76 32 38 2E 30 30 2E 30 32 20 42 6F 6F 74 6C 6F 61 64 65 72 20 76 30 31 2E 30 30 2E 30 32 11 49

(**INS401 8550-4006-01 2179000014 Hardware v1.0 RTK_INS App v28.00.02 Bootloader v01.00.02**) (will updated FW28.02 later)

(Ethernet Data Frame: **38 14 28 3f 12 e2 7c c9 b4 7d 00 28** 00 87 55 55 … … 11 49, **'7c c9 b4 7d 00 28'** is MAC address of INS401)

After the response msg is received, device info (mac of INS401) is clear now.

2nd Step-Set all parameters we want

Table5. Send 'SET Command 0xcc03' to INS401:

Name and value	**Command**	**feedback**
"gnss lever arm x" -2.036	55 55 03 cc 08 00 00 00 01 00 00 00 d3 4d 02 c0 89 58	Same feedback message for each set command: 55 55 03 CC 04 00 00 00 00 00 00 00 6F 20
"gnss lever arm y" -0.386	55 55 03 cc 08 00 00 00 02 00 00 00 cb a1 c5 be 1a 83	
"gnss lever arm z" -0.938	55 55 03 cc 08 00 00 00 03 00 00 00 c5 20 70 bf 02 b9	
"vrp lever arm x" -1.684	55 55 03 cc 08 00 00 00 04 00 00 00 50 8d d7 bf c4 5f	
"vrp lever arm y" -0.004	55 55 03 cc 08 00 00 00 05 00 00 00 6f 12 83 bb ad 7c	
"vrp lever arm z" 0.241	55 55 03 cc 08 00 00 00 06 00 00 00 b4 c8 76 3e 5c 59	
"user lever arm x" -1.684	55 55 03 cc 08 00 00 00 07 00 00 00 50 8d d7 bf 0c 2a	

"user lever arm y" -0.004	55 55 03 cc 08 00 00 00 08 00 00 00 6f 12 83 bb fb 18	
"user lever arm z" 0.241	55 55 03 cc 08 00 00 00 09 00 00 00 b4 c8 76 3e 85 9b	

3rd step-Save current configurations

Send 'SAVE Command 0xcc04':

55 55 04 CC 00 00 00 00 8F 0C

This command is used for permanent save the current configurations

Response command:

55 55 04 CC 04 00 00 00 00 00 00 00 73 DA

After the response msg is received, the command takes effect.

4th Step-Check ID parameters in current configuration

Table6. Send 'GET Command 0xcc02' to INS401:

Name	Get command and feedback
"gnss lever arm x" -2.036	55 55 02 cc 04 00 00 00 01 00 00 00 76 d1
	55 55 02 cc 08 00 00 00 01 00 00 00 d3 4d 02 c0 f2 39
"gnss lever arm y" -0.386	55 55 02 cc 04 00 00 00 02 00 00 00 ed 0d
	55 55 02 cc 08 00 00 00 02 00 00 00 cb a1 c5 be 61 e2
"gnss lever arm z" -0.938	55 55 02 cc 04 00 00 00 03 00 00 00 9b b9
	55 55 02 cc 08 00 00 00 03 00 00 00 c5 20 70 bf 79 d8
"vrp lever arm x" -1.684	55 55 02 cc 04 00 00 00 04 00 00 00 ca 94
	55 55 02 cc 08 00 00 00 04 00 00 00 50 8d d7 bf bf 3e
"vrp lever arm y" -0.004	55 55 02 cc 04 00 00 00 05 00 00 00 bc 20
	55 55 02 cc 08 00 00 00 05 00 00 00 6f 12 83 bb d6 1d
"vrp lever arm z" 0.241	55 55 02 cc 04 00 00 00 06 00 00 00 27 fc
	55 55 02 cc 08 00 00 00 06 00 00 00 b4 c8 76 3e 27 38
"user lever arm x" -1.684	55 55 02 cc 04 00 00 00 07 00 00 00 51 48
	55 55 02 cc 08 00 00 00 07 00 00 00 50 8d d7 bf 77 4b
"user lever arm y" -0.004	55 55 02 cc 04 00 00 00 08 00 00 00 85 a6
	55 55 02 cc 08 00 00 00 08 00 00 00 6f 12 83 bb 80 79
"user lever arm z" 0.241	55 55 02 cc 04 00 00 00 09 00 00 00 f3 12
	55 55 02 cc 08 00 00 00 09 00 00 00 b4 c8 76 3e fe fa

Appendix-A: CRC calculation

The following is the 16-bit CRC sample code used in most of the code development.

```
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);
}
```