MIMU22BL

Integration Guide

Revision 1.3

R&D Centre:
GT Silicon Pvt Ltd
171 MIG, Awadhpuri, Block B,
Lakhanpur
Kanpur (UP), India, PIN – 208024

Tel: +91 512 258 0039
Mobile: +91-700-741-0690
Email: info@inertialelements.com
URL: www.inertialelements.com
www.gt-silicon.com

Revision History

Revision	Revision Date	Updates
1.0	06 January 2018	Initial version
1.1	23 January 2018	Minor typographical corrections
1.2	2 February 2018	Few updated commands
1.3	5 February 2018	This integration document is for firmware version 1.2 where the sampling frequency is changed from 200 Hz to 250 Hz and the heading is being calculated based on Magnetometer of IMU # 3 instead of fused Magnetometer data.

Purpose & Scope

This document describes the data processing flow in **MIMU22BL**. It also describes communication protocol using which one can access and control the data and the processing at various stages, through an external application platform.

Please refer Openshoe embedded code and available scripts for better understanding.

Please refer following documents also for specific details:

1. Datasheet –ICM-20948
2. 32-bit AVR Microcontorller Specification

Introduction

MIMU22BL is Multi-IMU based inertial navigation module. MIMU22BL, with on-board four 9-DOF IMUs, is targeted towards foot mounted pedestrian application and can also be use as a development platform for carrying out research in motion sensing, robotics, IoT etc.

MIMU22BL has wireless interface (BLE 4.1) for data transfer. Due to on-board 32-bits floating point controller, device has a simplified dead reckoning interface (for foot-mounted application). An application platform receives a stream of displacement and heading changes, which it sums up to track the carrier. This is a significant simplification compared with handling and processing high-rate raw inertial measurements. There is also option of using on-board micro-USB connector for USB data transfer. Same port is used for battery charging.

MIMU22BL can also be used as a normal wireless IMU for other non foot mounted applications such as robotics, VR/AR etc. In summary, presence of on-board 32-bits floating point controller enables on-board computing and hence simplified output data format, with significantly reduced data transfer rate. And this also results in lesser computation overhead for the receiving (attached) device and superior tracking results. [1]

This document describes the data processing flow in MIMU22BL. It also describes communication protocol [2] using which one can access and control the data and the processing at various stages, through an external application platform.

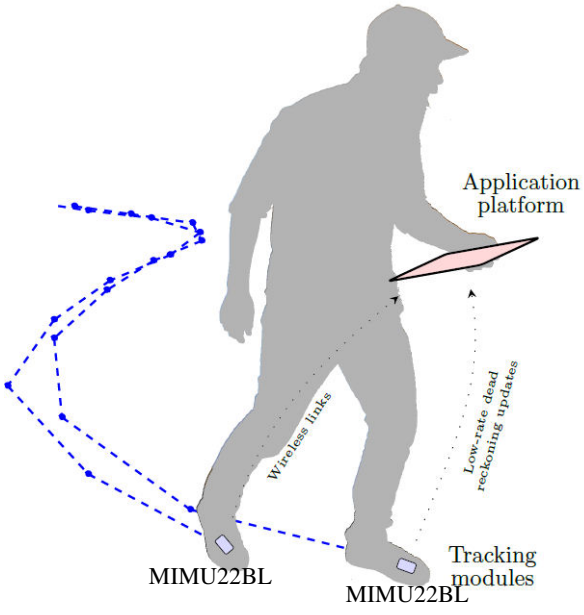


Fig. 1: Tracking with foot-mounted MIMU22BL and data collection using BLE 4.1

[1] John-Olof Nilsson, Amit K Gupta, Peter Handel, "Foot mounted inertial navigation made easy", InProc Indoor Positioning & Indoor Navigation (IPIN), Busan, Korea, 2014.

[2] Communication Protocol, www.openshoe.org

Data Flow

Sensors' raw data from IMU is read in parallel by the microcontroller through I2C buses realized in software. The sensors' data is calibration compensated and fused (averaged) to get single a_x, $a_y a_z$, g_x, g_y and g_z. ZUPT aided inertial navigation is implemented, which generates displacement and heading change data.

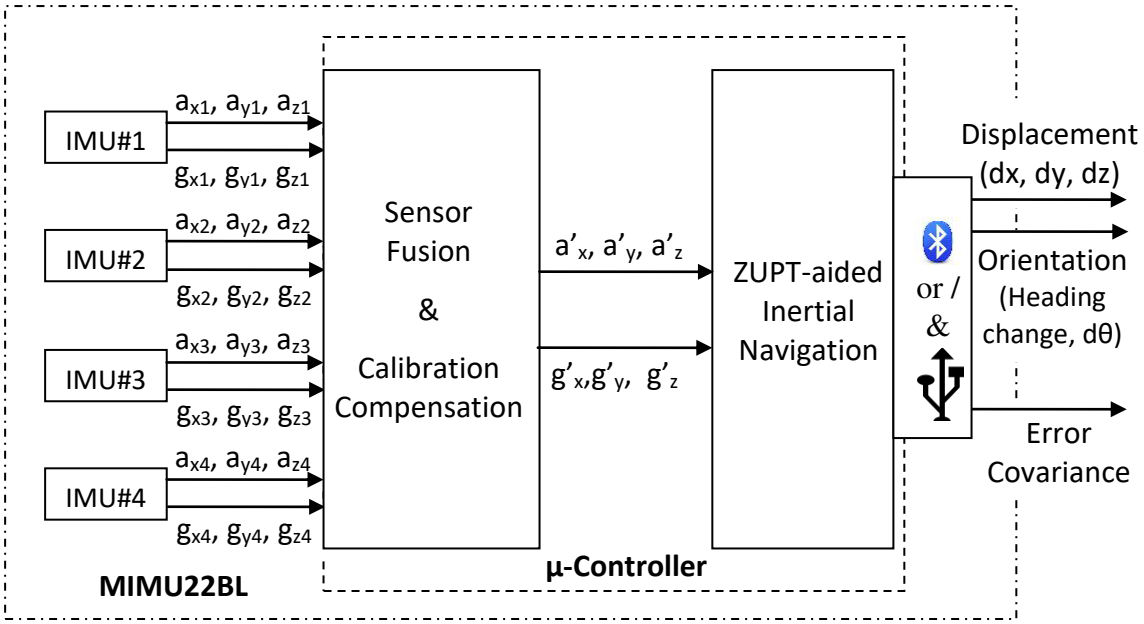


Figure 2 Illustration of data flow sequence in MIMU22BL with ZUPT-aided inertial navigation.*Rotation of the coordinate system w.r.t. the original one.

Communication Protocol

The interfacing application platform (Computer, Phone, Tab etc) sends a particular command (data request) to MIMU22BLwho serves the request by sending out required data to the application platform. Each command calls a particular state, which can send out variety of data depending upon command options.

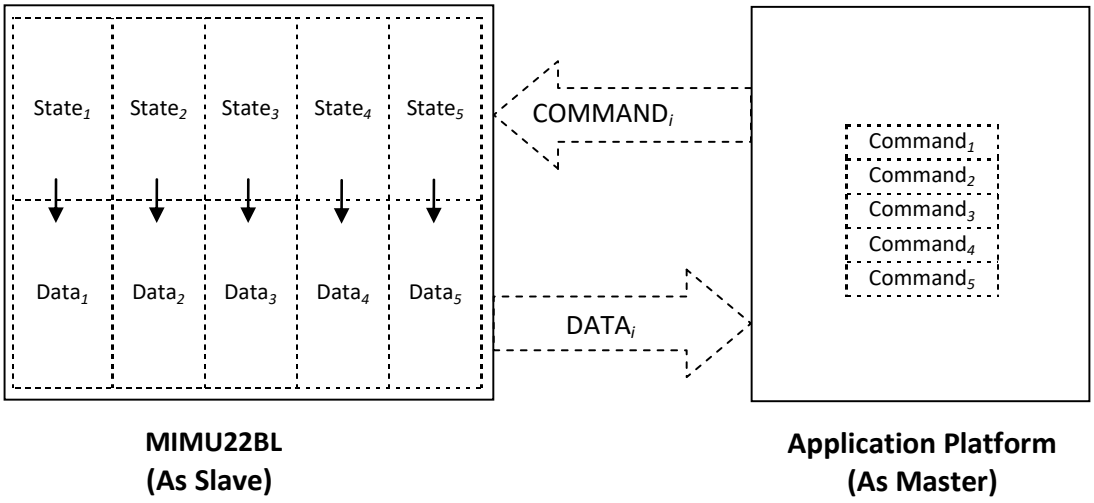


Figure 3 Data communication protocol between MIMU22BL and the application platform

Command format = [header,{payloads}, checksum*]

Checksum is of two bytes (cs1, cs2). Last two bytes in any command are checksum.

Table 1: Summary of some useful commands

Sl. No.	Command	Description	Syntax (in decimal)
1	Stepwise Dead Reckoning (PDR)	MIMU22BL outputs displacement (dx, dy, dz) & change in orientation (dθ) w.r.t the previous detected step and entries of a 4x4 symmetric covariance matrix	[52 0 52] cs1 = 0 cs2 = 52
2	Barometer Fused Altimeter	This is the height as obtained from the device by fusing barometer with IMUs.	[69 pl1 cs1 cs2] pl1 is the output rate divider
3	Processing off	MIMU22BL (controller) switches off all the processing. It makes the processing sequence empty	[50 0 50] cs1 = 0 cs2 = 50

4	All output off	MIMU22BL turns off output of all the states and stops transmitting data to the application platform	[34 0 34] cs1 = 0 cs2 = 34
5	Read inertial data	MIMU22BL starts sampling onboard IMUs' (acceleros' and gyros' data)	[48 19 0 0 67] cs1 = 0 cs2 = 67
6	Output inertial data	MIMU22BL transmitting the inertial sensors' raw data axi, ayi, azi, gxi, gyi, gzi of the selected IMUi, to the application platform.	[40 pl1 pl2 pl3 pl4 pl5 cs1 cs2] pl1, pl2, pl3, pl4 all together consists of 32bits. Each bit corresponds to particular IMU. Thus pl1-pl4 is used to select IMUs. pl5 is the output mode*. cs1, cs2: checksum
7	Read magnetometer data	MIMU22BL starts sampling and calibrating onboard IMUs' magnetometer data	[48 20 0 0 68] cs1 = 0 cs2 = 68
8	Read pressure data	MIMU22BL starts sampling onboard Pressure sensor's data	[48 21 0 0 69] cs1 = 0 cs2 = 69
9	Read Inertial and Magnetometer data	MIMU22BL starts sampling onboard IMUs' calibrated magnetometer and Inertial data simultaneously	[48 24 0 0 72] cs1 = 0 cs2 = 72
10	Read Osmium Data	MIMU22BL starts sampling onboard IMUs 'calibrated magnetometer and Inertial as well as pressure sensor data simultaneously	[48 22 0 0 70] cs1 = 0 cs2 = 70
11	Output Osmium Data	MIMU22BL transmitting the inertial sensors' raw data axi, ayi, azi, gxi, gyi, gzi, magnetometer raw data mxi, myi, mzi and temperature data of the selected IMUs and Pressure and temperature data from Pressure sensor to the application platform.	[43 pl1 pl2 pl3 pl4 pl5 cs1 cs2] pl1, pl2, pl3, pl4 all together consists of 32bits. Each bit corresponds to particular IMU. Thus pl1-pl4 is used to select IMUs. pl5 is the output rate divider cs1, cs2: checksum
12	Read IMU temperature data	MIMU22BL starts reading internal temperature of the onboard IMUs'	[53 0 53] cs1 = 0 cs2 = 53
13	Output IMU temperature data	MIMU22BL starts transmitting temperature of the selected onboard IMU(s), to the application platform.	[40 pl1 pl2 pl3 pl4 pl5 cs1 cs2] pl1, pl2, pl3 and pl4 are same as in the command for state OUTPUT_IMU_RD. pl5 is the output mode byte, cs1, cs2: checksum
14	High precision IMU (Normal IMU)	MIMU22BL outputs g'x, g'y, g'z, a'x, a'y, a'z (ref Fig 2 and Fig 3) after calibration compensation and sensor fusion.	[64 pl1 cs1 cs2] pl1 = output rate divider cs1, cs2: checksum

15	High precision IMU with bias estimation	MIMU22BL outputs g'x, g'y, g'z, a'x, a'y, a'z after calibration compensation and sensor fusion along with the estimated bias.	[65 pl1 cs1 cs2] pl1 = output rate divider cs1, cs2: checksum
16	High precision IMU with Magnetometer #3 data	MIMU22BL outputs g'x, g'y, g'z, a'x, a'y, a'z (ref Fig 2 and Fig 3) after calibration compensation and sensor fusion along with calibrated Magnetometer #3 data (m'x, m'y,m'z)	57 pl1 cs1 cs2 pl5 is the output mask mode*, cs1, cs2: checksum
17	High precision IMU with Pressure data	MIMU22BL outputs g'x, g'y, g'z, a'x, a'y, a'z (ref Fig 2 and Fig 3) after calibration compensation and sensor fusion along with pressure data.	57 pl1 cs1 cs2 pl5 is the output mask mode*, cs1, cs2: checksum
18	High precision IMU with Pressure and Magnetometer #3 data	MIMU22BL outputs g'x, g'y, g'z, a'x, a'y, a'z (ref Fig 2 and Fig 3) after calibration compensation and sensor fusion along with calibrated Magnetometer #3 data (m'x, m'y, m'z) and pressure data.	57 pl1 cs1 cs2 pl5 is the output mask mode*, cs1, cs2: checksum
19	Request output of state	Requests the state connected to the state ID to be output.	[32 pl1 pl2 cs1 cs2] pl1 = state ID pl2 = output mode cs1, cs2: checksum
20	Request output of multiple states	Requests the multiple states connected to the state IDs to be output.	[33 pl1 pl2 pl3 pl4 pl5 pl6 pl7 pl8 pl9 cs1 cs2] pl1-pl8 = state ID pl9 = output mode cs1, cs2: checksum
21	Set the gravity value	Set acc. due to gravity "g" value	[6 g0 g1 g2 g3 g4 g5 g6 cs1 cs2] g0 to g6 = g value cs1, cs2: checksum
22	Calibration data values set	Input calibration data of MIMU22BL	[36 pl1 pl2 pl3 pl4 pl5 pl6 pl7 pl8 pl9 pl10 pl11 pl12 pl13 pl14 pl15 pl16 pl17 pl18 pl19 cs1 cs2] pl1 = select IMU position pl2 – pl19 = 4 byte calibration data cs1, cs2: checksum
23	Calibration F- bias set	Input fused accelerometer bias	[8 pl1 pl2 pl3 cs1 cs2] pl1 – pl3 = 4 byte each input data cs1, cs2: checksum
24	Calibration G- bias set	Input fused gyroscope bias	[9 pl1 pl2 pl3 cs1 cs2] pl1 – pl3 = 4 byte each input data cs1, cs2: checksum
25	Set calibration F & G bias	Input the bias estimation value of individual IMU's accelerometer and gyroscope	[37 pl1 pl2 pl3 pl4 pl5 pl6 pl7 cs1 cs2] pl1 = select IMU position pl2 – pl7 = 4 byte each data cs1, cs2: checksum
26	Set FG scale	Select the Accelerometer and gyroscope scale	[68 pl1 pl2 cs1 cs2] pl1= fs_scale

			pl2 = afs_scale cs1, cs2: checksum
27	Set IMUs	Select the number of IMUs	[7 pl1 cs1 cs2] pl1= select IMUs position cs1, cs2: checksum
28	Flog set IMU	Log_2(No. of IMUs)	[5 pl1 cs1 cs2] pl1= select # of IMUs position cs1, cs2: checksum
29	Software Version	Version of the Firmware by which MIMU22BL is programmed	[41 cs1 cs2] cs1, cs2: checksum
30	Get Device ID	ID of the device	[25 cs1 cs2] cs1, cs2: checksum
31	Stepwise Dead Reckoning with true compass heading	MIMU22BL outputs tilt compensated compass heading, displacement (dx, dy, dz) & change in orientation (dθ) w.r.t the previous detected step and entries of a 4x4 symmetric covariance matrix	[67 64 01 31]
32	Output Raw IMU+ Magneto Data	MIMU22BL transmitting the inertial sensors' raw data axi, ayi, azi, gxi, gyi, gzi, magnetometer calibrated data mxi, myi, mzi	[33 57 58 59 60 185 186 187 188 rd cs1 cs2]
33	Output Calibrated IMU+ Magneto Data	MIMU22BL transmitting the inertial sensors' calibrated data axi, ayi, azi, gxi, gyi, gzi, magnetometer calibrated data mxi, myi, mzi	

*Output mode: Its 1 byte output mode. 1st to 4th bit is output rate divider (0x0 to 0x0f). 5th bit is set (add 16 to the rate divider) for lossless BLE transmission. 6th bit is set to output a single time if rate divider isset to 0. 7th bit is set for raw IMU data output. 8th bit is set for raw IMU temperature data.

Table 2: Output mode Sample

Output mode (Binary Bits)	Rate divider	Lossless	Single time output	Raw IMU	Raw IMU Temperature	Decimal Equivalent
0000 0000 -0000 1111	1-15	x	x	x	x	1-15
0100 0001 – 0100 1111	1-15	x	x	✓	x	65-79
0101 0001 – 0101 1111	1-15	✓	x	✓	x	81-95
1001 0001 – 1001 1111	1-15	✓	x	x	✓	145-159

1000 0001-1000 1111	1-15	x	x	x	✓	129-143
0110 0000	0	x	✓	✓	x	96

Output Rate Divider: It constitutes of only one byte. It is used to manipulate the rate of output of the particular state asked in this command. If the frequency of the running the whole main method/loop is 'f' then the frequency of output of this particular state will be f/ (2^ (pl2-1)). The largest value of pl2 can be 15. Thus if pl2=1 (smallest value), then the particular state will be output every main cycle or at lower frequency. (One main loop/cycle consists of tasks from reading IMU data to transmitting through USB.)
In very simple words, Output Rate Divider decides the rate of data transmission from MIMU22BLto the application platform. The maximum transmission rate is 250 Hz which is same as the inertial sensors' sampling frequency. Transmission rates for the Output Rate Divider upto 7 is given in the following table:

Output Rate Divider	Data Transmission Rate (Hz)
1	250
2	125
3	62.5
4	31.25
5	15.625
6	7.8125
7	3.90

*Output mask mode : This is basically same as output mode but we have added some extra mask for procuring data

Table 2: Output Mask mode

Output mode (Binary Bits)	Rate divider	Lossless	Single time output	Normal +Magneto (F) (40)	Normal +Pressure(80)	Normal + Pressure + Magneto (F) (60)	Decimal Equivalent
0000 0000 - 0000 1111	1-15	x	x	x	x	x	1-15
0100 0001 – 0100 1111	1-15	x	x	✓	x	x	65-79
0101 0001 – 0101 1111	1-15	✓	x	✓	x	x	81-95
1001 0001 – 1001 1111	1-15	✓	x	x	✓	x	145-159
1000 0001- 1000 1111	1-15	x	x	x	✓	x	129-143

0110 0000	0	x	✓	✓	x	x	96
0110 0001- 0110 1111	1-15	x	x	x	x	✓	97-111
0111 0001- 0111 1111	1-15	✓	x	x	x	✓	113-127

Please refer Appendix 1 for detailed description of Commands and States.

Stepwise Dead Reckoning with Android Application Platform

As an example, we consider a specific case of using MIMU22BL for stepwise dead reckoning with an (Android) application platform, communicating wirelessly.

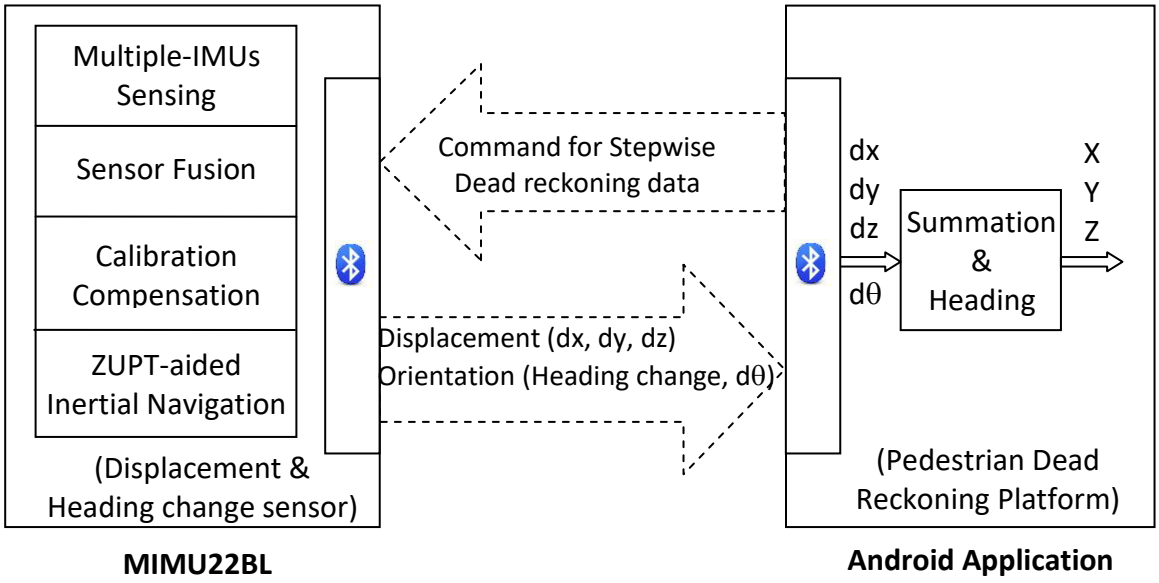


Figure 4 Interfacing of MIMU22BL with an application platform

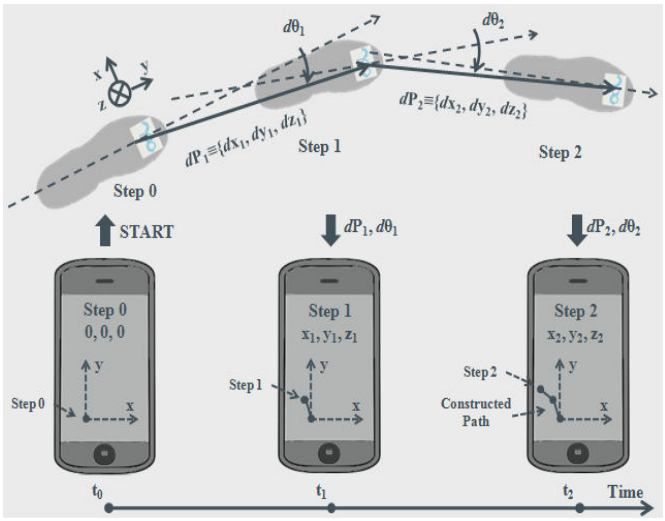
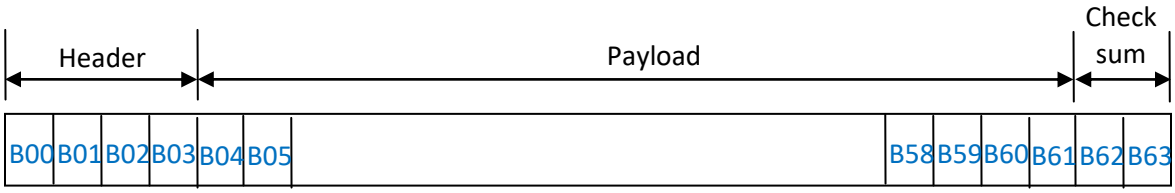


Fig. 5: Pedestrian Dead Reckoning (PDR) is simplified with the foot-mounted MIMU22BL. The device starts transmitting location data at every step, on receiving start command from the application platform.

The MIMU22BLsends out stepwise dead reckoning data in form of packets via wireless communication, in response to the command *[52 0 52]* from application platform. Below figure illustrates how the packets look like.



B00: State of the Header
B01-B02: Data packet number
B03: Number of bytes in Payload

B04-B07: Displacement in x (Type float)
B08-B11: Displacement in y (Type float)
B12-B15: Displacement in z (Type float)
B16-B19: Change in angle around z axis (Type float)

B20-B59: 10 entries (4 bytes each) of a 4x4 symmetric covariance matrix

B60-B61: Step counter

B62-B63: Check sum

Figure 5 Data packet for stepwise dead reckoning

Data Packet

These packets consisted of 64 bytes.

1. **Header**: First 4 bytes are 'Header'.
    a. Among these 4 bytes first is the header which is STATE_OUTPUT_HEADER which is 170.
    b. 2nd and 3rd byte jointly tells the data packet number sent by the device since the beginning of stepwise dead reckoning. Thus the packet number is a 16-bits variable.
    c. 4th byte depicts the payload i.e. the number of bytes this data packet contains which consists of required information asked by the user.

2. **Payload**: Next is the payload, which consisted of 14 elements of type 'float' thus comprising of 14*4=56 bytes.
    a. First 4 elements consisted of the delta vector i.e. the change in position x, y, z and the angle of rotation around z-axis (the change in the angle of the x-y plane). As these elements are of type float, thus are of 32 bits. The change in position is between two successive ZUPT instances, i.e. the displacement vector from one step to the next one.
    b. The other 10 elements of the payload are used to form the covariance matrix, which is a 4x4 symmetric matrix, thus 10 elements. These are not used in the step-wise dead reckoning. These are used in data fusion from two MIMU22BLsin case of dual foot mounted system.

3. **Step Counter**: Next two bytes consisted of step counter, which is a counter taking record of ZUPT instances observed. This is essentially number of times velocity of MIMU22BLgoes down below predefined threshold. Hence it is the number of steps taken by the wearer, if MIMU22BLis used for foot-mounted pedestrian tracking.

4. **Checksum**: The last two bytes of 64 bytes are consisted of check sum which is sum of all the bytes received prior to these. These are used to cross-check correctness of the data transferred.

Rotation in Application Platform

MIMU22BL transmits tracking data with respect to its own frame which itself is not fixed with respect to the user's global reference frame. Therefore the data should undergo rotational transformation before being presented to the end user in a global reference frame. (Here global refers to the coordinate axis in which the system is initialized.)

The first four bytes of Payload are the position and heading change vector, i.e. dx, dy, dz, dθ (dθ is the change in angle of rotation about z-axis). These values are the output from the device at every ZUPT instance. As mentioned earlier, each set of delta values describes movement between two successive steps. The delta values prior and after each ZUPT instance, are independent to each other as the system is reset every ZUPT, i.e. the delta values in one data packet is independent of data value in data packet transmitted just before, just after and all other. The position and heading change vector are with reference to the coordinate frame of last ZUPT instance.

The 'dθ' corresponds to the deviation in the orientation of MIMU22BL. It is rotation of the x-y plane about the z-axis, and z-axis is always aligned with the gravity. The data is thus used for updating the orientation of the system, and obtaining the global reference frame. The two dimensions of displacement vector (dx, dy) are transformed to the global frame by applying rotation and thus (x,y) in the global frame is obtained. As we do not consider any change in alignment in z-axis, updating the z coordinate is performed by simply adding present dz to the z obtained for previous ZUPT.

Thus x,y,z coordinates in the global reference frame are obtained at every ZUPT.
Details are covered in Application Note. You may also refer Appendix 2 for a piece of an example Matlab code for better understanding of implementation.

Appendix 1

1. **State**: Stepwise Dead Reckoning (SWDR)
   Command: [52 0 52]

   MIMU22BL sends out data packet for stepwise dead reckoning in response to this command [2]. Use this command only through Bluetooth. It will not work through USB.

2. **State**: Barometer fused Altimeter (BFA)
   Command: [69 pl1 cs1 cs2]
   This is the height as obtained from the device by fusing barometer with IMUs. In SWDR we received corrected relative altitude. This command is for relative information based on fused barometer and IMU data without any correction.

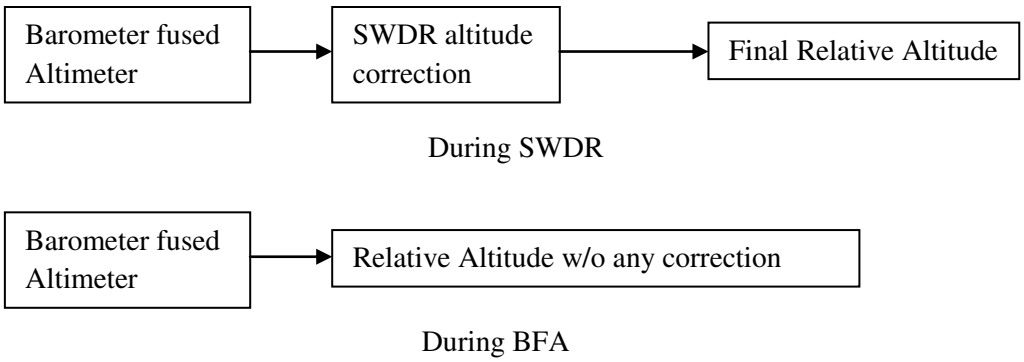


Figure 6 Flow diagram of altitude from SWDR AND BFA

3. **State**: Processing Off
   Command: [39 0 39]

   MIMU22BL terminates all the ongoing processing in response to this command. This is soft OFF button.

4. **State**: All Output Off
   Command: [33 0 33]

   If this command is passed to the system, then the transmission of data packets would stop. But there will be no change in the process going on inside, that all the functions would be running as it is, just the thing is values would not be output.

5. **State**: Read Inertial Data
   Command: [48 19 rd cs1 cs2]

   This command is used to initiate sampling onboard inertial sensors' data by MIMU22BL.

   rd: Output rate divider
   cs1, cs2: Two bytes checksum

   Example:
   Recommended for USB:
   [48 19 0 0 67]
   [40 0 0 0 15 65 0 120] (o/p data rate:1 = 250 Hz)

   Recommended for Bluetooth
   [48 19 0 0 67]
   [40 0 0 0 15 68 0 124] (o/p data rate: 4 = 25 Hz)

6. **State**: Output Inertial Data
   Command: [40 pl1 pl2 pl3 pl4 pl5 cs1, cs2]

   MIMU22BL outputs IMU sensors' readings in response to this command.

   Here pl1, pl2, pl3 and pl4 constitute one byte each. These are used for selecting IMUs. Consider the pl1, pl2, pl3 and pl4 all together constitute 4 bytes i.e. 32 bits, where each bit corresponds to one of the 32 IMUs.

   Consider if we want to select the first 4 IMUs we want to have the 4 LSBs to be 1 which can be made by having pl4 be 15 (binary: 00001111) and all the others be 0. Thus [40 0 0 0 15 1 0 56] is a sample command selecting the first four IMUs for getting the raw inertial data. pl5 here is the output mode byte.

   Here the output consists of [a_{xi}, a_{yi}, a_{zi}, g_{xi}, g_{yi}, g_{zi}] for each IMU selected, as in Fig 2. As each of the values corresponds to 2 bytes, thus the output is of 12 bytes for each IMU.

Note that MIMU22BL has only four IMUs. However the s/w can support upto thirty two IMUs.

cs1, cs2: Two bytes checksum

Example:

Set of commands to obtain data packets containing 4 accelerometers' and 4 gyroscopes' data:

Recommended for USB:

[48 19 0 0 67]

[40 0 0 0 15 65 0 120] (o/p data rate:1 = 250 Hz)

Recommended for Bluetooth

[48 19 0 0 67]

[40 0 0 0 15 68 0 124] (o/p data rate: 4 = 25 Hz)

Here are the example data packets containing 4 accelerometers' and 4 gyroscopes' data, obtained as a result of above set of commands (USB):

AA 00 01 34 21 C4 48 F7 00 6E 00 54 07 57 FF E4 00 05 00 17 FF 98 FF 81 F7 6A FF D2 00 13 00 11 00 7E 00 57 07 D9 FF F6 00 0B FF EF FF 9A FF 8C F7 60 FF F6 FF F0 FF FC 1C 8C

AA 00 02 34 21 C5 35 EC 00 6E 00 54 07 57 FF E2 00 04 00 16 FF 90 FF 7F F7 68 FF D4 00 13 00 11 00 68 00 56 07 D0 FF F5 00 0A FF F0 FF 8E FF 91 F7 54 FF F2 FF F2 FF FE 1C 2E

Description of the above packet:

Total packet size: 58 bytes

Header (4 bytes): AA 00 01 34 (AA - command; 00 01 - Data packet number; 34 - data packet size)

Data packet (52 bytes): 21 C4 48 F7 00 6E 00 54 07 57 FF E4 00 05 00 17 FF 98 FF 81 F7 6A FF D2 00 13 00 11 00 7E 00 57 07 D9 FF F6 00 0B FF EF FF 9A FF 8C F7 60 FF F6 FF F0

FF FC (First 4 bytes for time stamp; 2 bytes each for ax1,ay1,az1,gx1,gy1,gz1,ax2,ay2,az2,gx2,gy2,gz2,ax3,ay3,az3,gx3,gy3,gz3,ax4,ay4,az4,gx4,gy4,gz4)

Check sum (2 bytes): 1C 8C

7. **State:** Read magnetometer data

   Command: [48 20 rd cs1 cs2]

   This command is used to initiate sampling and calibration of the onboard IMU's magnetometer data by MIMU22BL.

   rd: Output rate divider

   cs1, cs2: Two bytes checksum

   It is recommended to set the rd to default 0. This will sample data at the maximum possible sampling rate.

8. **State**: Read Pressure data

   Command: [48 21 rd cs1 cs2]

   This command is used to initiate sampling of the onboard pressure sensor data by MIMU22BL.

   rd: Output rate divider

   cs1, cs2: Two bytes checksum

9. **State**: Read Inertial and Magnetometer data

   Command: [48 24 rd cs1 cs2]

   This command is used to initiate sampling and calibration of the onboard IMU's magnetometer and sampling of inertial sensors' data by MIMU22BL simultaneously.

   rd: Output rate divider

   cs1, cs2: Two bytes checksum

10. **State**: Read Osmium Data

Command: [48 22 rd cs1 cs2]

This command is used to initiate sampling and calibration of the onboard IMU's magnetometer , sampling of inertial sensors' data and sampling of the onboard pressure sensor data by MIMU22BL simultaneously.

rd: Output rate divider

cs1, cs2: Two bytes checksum

Table 4: Data access commands for Normal IMU, Magnetometer #3 and pressure sensor

State	Command	Inertial	Magneto	Pressure
Read Inertial Data	[48 19 0 0 67]	✓	x	x
Read magnetometer data	[48 20 0 0 68]	x	✓	x
Read Pressure data	[48 21 0 0 69]	x	x	✓
Read Inertial and Magnetometer data	[48 24 0 0 72]	✓	✓	x
Read Osmium Data	[48 22 0 0 70]	✓	✓	✓

11. **State**: Output Osmium Data

Command: [43 pl1 pl2 pl3 pl4 rd cs1 cs2]

MIMU22BL transmitting the inertial sensors' raw data axi, ayi, azi, gxi, gyi, gzi, magnetometer raw data mxi, myi, mzi and temperature data of the selected IMUs and Pressure and temperature data from Pressure sensor to the application platform.

pl1 pl2 pl3 pl4 – These payloads is for selection of IMUs.

rd- Output rate divider

cs1 cs2- Check Sum.

Example:

Set of commands to obtain data packets containing 4 accelerometers' and 4 gyroscopes' data, 4 magnetometers, 1 pressure sensor data, 4 sets of temperature from 4 IMUs, 1 set of temperature from pressure sensor.

Example for USB

[48 22 rd cs1 cs2]

[43 pl1 pl2 pl3 pl4 rd cs1 cs2]

Data packets received from USB:

AA 04 F6 7E B6 77 68 0D 00 3F 28 E7 E7 43 FF FE FF 48 07 B0 FF ED 00 03 00 03 00 39 FF 60 08 10 00 0C FF FF 00 08 FF E4 FF 3A 07 D1 00 13 FF DC 00 0F 00 0A FF 31 07 8A 00 05 FA E4 00 08 0E 90 0D 50 0E 00 0F D0 C2 98 00 00 41 48 00 00 42 9A 00 00 C2 CF 00 00 C3 8D C0 00 42 7C 00 00 C3 1B 00 00 41 CC 00 00 42 95 00 00 41 88 00 00 41 30 00 00 42 F2 00 00 C2 0A 00 00 C1 12 00 00 C2 43 80 00 2A F1.

Data Packets: 1st 4 bytes : AA 04 F6 7E: Command header-AA; 04 F6- Data packet number, 7E-Data size.

2nd 4 bytes : B6 77 68 0D: Time stamp

3rd 4 bytes: 00 3F 28 E7: Raw pressure data

4th 2 bytes: E7 43: Pressure sensor temperature data

Next 26 bytes (2 bytes for each for ax1, ay1, az1, gx1, gy1, gz1, temp1) and then(4bytes each for mx1,my1,mz1)respectively.

Similarly the next 26 bytes are for IMU 2 in a similar fashion and so on.

Last 2 bytes : 2A F1- Check sum.

12. **State**: Read IMUs' temperature

    Command: [53 0 53]

    This command is used to initiate reading of internal temperature of onboard inertial sensors by the internal controller.

13. **State**: Output IMU Temperature Data

    Command: [40 pl1 pl2 pl3 pl4 pl5 cs]

    MIMU22BL outputs temperature of the selected IMUs in response to this command. The output consisted of 2 bytes containing the value of temperature, for every selected IMU.

    pl1, pl2, pl3 and pl4 are same as in the command for state OUTPUT_IMU_RD. pl5 is 128 + the actual output rate divider (1, 2, 3,… etc).

    cs1, cs2: Two bytes checksum

    Example:

    [53 0 53] (Enables reading IMUs' internal temperature)

    [40 0 0 0 15 129 0 184] (o/p data rate:1 = 250 Hz)

Here are the example data packets containing temperature information data of all the 4 IMUs, obtained as a result of above set of commands (USB):
AA D0 28 0C 00 57 00 19 BF 16 00 1B 0F 2E 3F 9A 04 24

14. **State**: High precision IMU (Normal IMU)
Command: [64 pl1 cs1 cs2]
pl1 = output rate divider
cs1, cs2: checksum

With this command, MIMU22BL can be used as a single high precision IMU. The modules are capable of giving calibration compensated and fused data from the IMU array. This means the module can be used as a single precision IMU which outputs 3-axis acceleration and 3-axis gyroscope data.The device outputs g'_x, g'_y, g'_z, a'_x, a'_y, a'_z (ref Fig 2 and Fig 3) – 4 bytes each, float type.
Below is the sample data from MIMU22BL lying tilted on a table. The data was collected at data rate 250Hz. Each packet contains 34 bytes. First 4 bytes are header (1st byte command id, 2 bytes packet number, 1 byte payload size) and last 2 bytes are checksum. Remaining 28 bytes (also known as payload) in sequence: 4 bytes for time stamp,12 bytes for a'_x, a'_y, $a'_z$and next 12 bytes for g'_x, g'_y, g'_z.

AA 17 6C 1C C3 E6 7A 98 40 74 22 11 40 7B 3D 0F 41 03 7D 75 3A 16 A9 04 BC 38 C7 5C 3B 44 67 1A 0B 3C
AA 17 6D 1C C3 EA 1A 5A 40 72 2A 66 40 74 6B 15 41 02 00 56 BB 81 90 AE BB C4 02 4C 3C 15 68 7D 0B F7
AA 17 6E 1C C3 EE 02 2C 40 6E 2F AD 40 72 CF 04 41 00 D0 13 3B CE D8 92 3C 7B 8D 07 BB 6F B7 7A 0D 76
AA 17 6F 1C C3 F1 EA 29 40 71 61 98 40 79 05 2B 41 02 66 13 39 DB E9 56 3C DD 6F 45 BB 88 B3 AC 0E 24
AA 17 70 1C C3 F5 D2 27 40 76 1F C3 40 79 BB 38 41 04 59 68 3B 3B F5 01 3C 54 2B 53 BB 4C 5A C9 0C EC
AA 17 71 1C C3 F9 BA 52 40 72 AA BA 40 7C 8D A5 41 03 A5 AA BB 8B 3A FE BB 25 09 28 3B CB 04 99 0E DF
AA 17 72 1C C3 FD A2 29 40 73 B4 98 40 73 6C E6 41 02 5C 5B BB 4E 0E 26 BB F8 19 C5 3C 6C 9E F2 0E DE
AA 17 73 1C C4 01 8A 63 40 6C 6C 7B 40 76 29 56 41 01 49 FF 3A 85 78 8A 3C 05 64 A7 BC 1B 9A 3C 0C 0E

15. **State**: High precision IMU with bias estimation
Command: [65 pl1 cs1 cs2]
pl1 = output rate divider
cs1, cs2: checksum

This command gives the combined fused precision IMU data after calibration compensation and adding the estimated bias. The output format will be same as the normal IMU data except the estimated bias is added with each data set.

16. **State**: High precision IMU with Magnetometer #3 data
Command: 57 pl1 cs1 cs2
Refer the Output mask mode table for how to use pl1.

With this command, MIMU22BL can be used as a single high precision IMU along with fused and calibrated magnetometer data. The modules are capable of giving calibration compensated and fused data from the IMU array. This means the module can be used as a single precision IMU which outputs 3-axis acceleration and 3-axis gyroscope data. The device outputs a'_x, a'_y, a'_z, g'_x, g'_y, g'_z , m'_x , m'_y, m'_z (ref Fig 2 and Fig 3) – 4 bytes each, float type respectively. Total 46 data bytes will be there in each packet.
Below is the data packet received through USB with the device placed firmly on the table.

AA 17 6F 28 C3 F1 EA 29 40 71 61 98 40 79 05 2B 41 02 66 13 39 DB E9 56 3C DD 6F 45 BB 88 B3 AC FF 60 08 10 00 0C FF FF 00 08 FF E4 13 9C

First 4 bytes are header (1st byte command id, 2 bytes packet number, 1 byte payload size) and last 2 bytes are checksum. Remaining 28 bytes (also known as payload) in sequence: 4 bytes for time stamp,12 bytes for a'_x, a'_y, $a'_z$and next 12 bytes for g'_x, g'_y, g'_z, next 12 bytes are for m'_x , m'_y, m'_z.

17. **State**: High precision IMU with pressure data
Command: 57 pl1 cs1 cs2
Refer the Output mask mode table for how to use pl1.

With this command, MIMU22BL can be used as a single high precision IMU along with pressure sensor data. The modules are capable of giving calibration compensated and fused data from the IMU array. This means the module can be used as a single precision IMU which outputs 3-axis acceleration and 3-axis gyroscope data. The device outputs g'_x, g'_y, g'_z, a'_x, a'_y, a'_z ,P_r– 4 bytes each, float type respectively. Total 38 data bytes will be there in each packet.
Below is the data packet received through USB with the device placed firmly on the table.

AA 17 6E 1C C3 EE 02 2C 40 6E 2F AD 40 72 CF 04 41 00 D0 13 3B CE D8 92 3C 7B 8D 07 BB 6F B7 7A 00 3F 28 E7 0D 76

First 4 bytes are header (1st byte command id, 2 bytes packet number, 1 byte payload size) and last 2 bytes are checksum. Remaining 28 bytes (also known as payload) in sequence: 4 bytes for time stamp,12 bytes for a'_x, a'_y, $a'_z$and next 12 bytes for g'_x, g'_y, g'_z, next 4 bytes are for P_r.

18. **State**: High precision IMU with pressure and Magnetometer #3 data
Command: 57 pl1 cs1 cs2

With this command, MIMU22BL can be used as a single high precision IMU along with fused and calibrated magnetometer data plus pressure sensor data will be outputted. A total of 50 data bytes will be there in each packet each data points are 4 bytes float format.

First 4 bytes are header (1st byte command id, 2 bytes packet number, 1 byte payload size) and last 2 bytes are checksum. Remaining 28 bytes (also known as payload) in sequence: 4 bytes for time stamp,12 bytes for a'_x, a'_y, $a'_z$and next 12 bytes for g'_x, g'_y, g'_z, next 12 bytes are for m'_x , m'_y, m'_z and the next 4 bytes are for pressure data.

19. **State**: Request output of state
Command: [32 pl1 pl2 cs1 cs2]
pl1 = state ID
pl2 = output mode
cs1, cs2: checksum

Requests the state connected to the state ID to be output. The output mode byte controls how the state is output. The states are ordered according to their IDs in the response.

Example command
[32 01 32 00 65]

Here is the output for the state ID (0X01) as a result of above set of commands (USB):
AA 06 76 04 1C FB 65 D9 03 7F

20. **State**: Request output of multiple states
Command:[33 pl1 pl2 pl3 pl4 pl5 pl6 pl7 pl8 pl9 cs1 cs2]
pl1-pl8 = state ID
pl9 = output mode
cs1, cs2: checksum
Requests the states connected to the state IDs to be output. The output mode byte controls how the state is output. The states are ordered according to their IDs in the response. This command is

the same as just as above with more IDs. If zeros are given instead of the IDs, no output will be given. Consequently, the command may be used to request output of up to 8 states.

Example command
[33 16 17 21 22 00 00 00 00 04 00 113]

Here is the output for the state ID (0X010, 0X011, 0X015, 0X016) as a result of above set of commands (USB):

AA 05 AF 38 00 18 C4 00 00 0D 30 00 FC 2F 88 00 FE 88 00 FC B8 00 FD 94 00 00 1A68 00 00 0B 80 00 FC 2E 38 00 80 00 FA B8 00 FD 40 00 00 02 66 A4 00 00 01 7317 84

21. **State**: Set the gravity value
Command:[6 g0 g1 g2 g3 g4 g5 g6 cs1 cs2]
g0 to g6 = g value
cs1, cs2: checksum

With this command the value of g can be input without programming the device. For e.g.to input value of acc. due gravity as 9.81 m/s^2, then input command would be [6 9 8 1 0 0 0 0 0 24].

22. **State**: Calibration data value set.
Command: [36 pl1 pl2 pl3 pl4 pl5 pl6 pl7 pl8 pl9 pl10 pl11 pl12 pl13 pl14 pl15 pl16 pl17 pl18 pl19 cs1 cs2]
pl1 = select IMU position
pl2 – pl19 = 4 byte calibration data
cs1, cs2: checksum

With this command the calibration file can be input to MIMU22BL without need to reprogram the device. pl1 is the IMU# and pl2 to pl19 are the calibration data. pl2 –pl4 are the x-axis accelerometer calibration components, pl5-pl7 and pl8 – pl10 are y axis and z axis accelerometer calibration components respectively. Similarly pl11-pl13, pl14-pl16 and pl17-pl19 are the x-axis, y axis and z axis gyroscope calibration components respectively.

23. **State**: Input fused Accelerometer Bias
Command:[8 pl1 pl2 pl3 cs1 cs2]
pl1 – pl3 = 4 byte each input data
cs1, cs2: checksum

With this command the fused bias estimation value of the accelerometer can be input without reprogramming the device. pl1,pl2 and pl3 are the bias estimation of the accelerometer along x, y and z axis respectively.

24. **State**: Input Fused Gyroscope Bias
Command:[9 pl1 pl2 pl3 cs1 cs2]
pl1 – pl3 = 4 byte each input data

cs1, cs2: checksum

With this command the fused bias estimation value of the gyroscope can be input without reprogram. pl1,pl2 and pl3 are the bias estimation of the gyroscope along x, y and z axis respectively.

25. **State**: Calibration F & G bias
Command: [37 pl1 pl2 pl3 pl4 pl5 pl6 pl7 cs1 cs2]
pl1 = select IMU position
pl2 – pl7 = 4 byte each data
cs1, cs2: checksum;

With this command we can give the individual Bias for each IMU. Pl1 is the IMU selected and pl2 to pl7 is the bias along x, y and z axis of accelerometer and gyroscope respectively

26. **State**: Set FG scale
Command: [68 pl1 pl2 cs1 cs2]
pl1= fs_scale
pl2 = afs_scale
cs1, cs2: checksum;

With this command the range of the accelerometer and gyroscope can be selected. The values of pl1 can be 0, 8, 16, 24 and pl2 can be 0, 8, 16, and 24 respectively. Please refer to Table 3 for details:
Table 6: Accelerometer and Gyroscope scale

Accelero scale	**Range**
0	±2g
1	±4g
16	±8g
17	±16g
Gyro scale	**Range**
0	500°/s
1	1000°/s
16	1500°/s
17	2000°/s

27. **State**: Set IMUs
Command: [7 pl1 cs1 cs2]
pl1= select IMUs position
cs1, cs2: checksum;

With this command selection of IMUs is possible. Only different combinations of 1, 2 and 4 IMUs can be selected. To select IMU #3 the command should be [7 8 0 15].

28. **State**: Software Version
Command: [41 cs1 cs2]
cs1, cs2: checksum;

MIMU22BL output the version of the programmed firmware.

The example output data transmitted via USB
AA 00 01 08 00 00 00 00 01 00 00 00 01 00 B5
Header: AA 00 01 08
Version:00 00 00 00 01 00 00 00 01 (1.1)
Check sum: 00 B5

29. **State**: Get Device Id
Command: [25 cs1 cs2]
cs1, cs2: checksum;

MIMU22BL gives the device ID with this command.
The example output of the command is given below
AA 00 02 08 00 00 16 08 03 00 00 06 00 DB
Header:AA 00 02 08
ID:00 00 16 08 03 00 00 06
Check Sum:00 DB

N.B: For all the commands from Sl No. 12 to 18 resetting MIMU22BL will set all the values to the default programmed values.

30. **State**: Stepwise Dead Reckoning with true compass heading
Command: [67 64 01 31]

MIMU22BL sends out data packet for stepwise dead reckoning along with true compass heading in response to this command. Use this command only through Bluetooth. It will not work through USB.

31. **State**: Raw IMU and Magnetometer from 4 IMUs

Command1: [48 24 0 0 72]

Command2: [33 57 58 59 60 185 186 187 188 rd cs1 cs2]

MIMU22BL transmitting the inertial sensors' raw data axi, ayi, azi, gxi, gyi, gzi, magnetometer raw data mxi, myi, mzi to the application platform.

rd- Output rate divider

cs1 cs2- Check Sum.

Example:

Set of commands to obtain data packets containing 4 accelerometers' and 4 gyroscopes' data, 4 magnetometers,.

Example for USB

[48 24 0 0 72]

[33 57 58 59 60 185 186 187 188 rd cs1 cs2]

Data packets received from USB:

[AA 1A CB 60 FF 98 F7 B1 FF 03 FF EE 00 00 00 00 FF C2 F7 E3 FF 3B 00 0E 00 08 FF FF FF BB F7 B4 FF 2C 00 08 FF DF 00 09 FF BD F7 B4 FE B7 00 0A FF F6 00 00 C2 4A 00 00 C3 6E 00 00 43 24 80 00 C2 6C 00 00 C3 FA 80 00 43 0A 00 00 C2 1E 00 00 C3 6C 00 00 43 10 80 00 42 5C 00 00 C3 7D 00 00 43 15 80 00 29 14]

Data Packets: 1st 4 bytes : AA 1A CB 60: Command header-AA; 1A CB- Data packet number, 60-Data size.

Next 48 bytes (2 bytes for each for ax1, ay1, az1, gx1, gy1, gz1,) then(2bytes each for ax2, ay2, az2, gx2, gy2, gz2)respectively and so on for all the four IMUs.

Next 48 bytes (4 bytes for each for mx1, my1, mz1,) then (4 bytes each for mx2, my2, mz2)respectively and so on for all the four IMUs.

Last 2 bytes: 29 14- Check sum.

32. **State**: Calibrated IMU and Magnetometer data from 4 IMUs

Command1: [49 24 26 00 00 00 00 00 00 00 99]

Command2: [33 37 38 39 40 185 186 187 188 rd cs1 cs2]

MIMU22BL transmitting the inertial sensors' calibrated data axi, ayi, azi, gxi, gyi, gzi, magnetometer calibrated data mxi, myi, mzi to the application platform.

rd- Output rate divider

cs1 cs2- Check Sum.

Example:

Set of commands to obtain data packets containing 4 accelerometers' and 4 gyroscopes' calibrated data, and 4 magnetometers calibrated data.

Example for USB

[49 24 26 00 00 00 00 00 00 00 99]

[33 37 38 39 40 185 186 187 188 rd cs1 cs2]

Data packets received from USB:

[AA 69 1E 90 3E 66 26 BF BE 82 63 06 41 17 06 D2 3C 72 54 98 BC C5 3A 50 BC 0A EC 42 3F 26 F2 12 BE E3 E0 FD 41 1A 44 99 BC 0D 3A E9 BC 56 DE 1A BC 19 CC DB 3C 04 D5 C1 BE 84 25 F1 41 1D F7 52 3B AB E7 39 3B 49 B4 68 BC 23 E8 19 3E 5F E3 46 BE E3 2E 2A 41 15 40 30 3C 98 FB 48 3B EB CF 12 BB 18 51 7B C0 40 00 00 C0 00 00 00 42 A8 00 00 C0 40 00 00 C3 AC 00 00 42 0E 00 00 C1 FC 00 00 BF C0 00 00 42 2E 00 00 43 69 80 00 C0 80 00 00 43 16 80 00 3A 44]

Data Packets: 1st 4 bytes : AA 69 1E 90: Command header-AA; 69 1E - Data packet number, 90-Data size.

Next 96 bytes (4 bytes for each for calibrated ax1, ay1, az1, gx1, gy1, gz1,) then(4 bytes each for ax2, ay2, az2, gx2, gy2, gz2)respectively and so on for all the four IMUs.

Next 48 bytes (4 bytes for each for mx1, my1, mz1,) then (4 bytes each for mx2, my2, mz2) respectively and so on for all the four IMUs.

Last 2 bytes: 3A 44- Check sum.

Appendix 2

Sample code (Matlab) for Stepwise Dead Reckoning

```
% Reset stepwise deadreckoning INS
fwrite(com, [52 0 52],'uint8');
fread(com,4,'uint8')

package_number_old = nan;
whileabort_flag==0
if(com.BytesAvailable>=64)
header = fread(com,2,'uint8'); % Header + number + Payload size (-) 4, (+)2
payload = fread(com,14,'float');
step_counter = fread(com,1,'uint16'); % Step counter
fread(com,1,'uint16'); % Checksum
fprintf(file,'%i %i %12.9f %12.9f %12.9f %12.9f %12.9f %12.9f %12.9f %12.9f %12.9f %12.9f %12.9f %12.9f
%12.9f\n',step_counter,payload);% for USB
dx = payload(1:4);
dP = Pvec2Pmat(payload(5:14));
pdef=find(eig(dP)<=0);
ifisempty(pdef)
chol(dP,'lower');
end
        [x_swP_sw] = stepwise_dr_tu(x_sw,P_sw,dx,dP);
figure(1)
plot([x_sw(1) x_sw_old(1)],[x_sw(2) x_sw_old(2)],'b');
axis equal;
drawnow;
x_sw_old = x_sw;
P_sw_old = P_sw;
end
end

% Stop output
fwrite(com, [50 0 50],'uint8'); % Processing off
fwrite(com, [34 0 34],'uint8'); % all output off

----------------------------------------

% stepwise_dr_tu
function [x2_out P2_out] = stepwise_dr_tu(x2_in,P2_in,dx2,dP)

% Input matrix
sin_phi = sin(x2_in(4));
cos_phi = cos(x2_in(4));

dR = [cos_phi -sin_phi 0 0;
```

```
sin_phicos_phi 0 0;
    0      0      1 0;
    0      0      0 1];

F = [1 0 0 -sin_phi*dx2(1)-cos_phi*dx2(2);
    0 1 0 cos_phi*dx2(1)-sin_phi*dx2(2);
    0 0 1 0;
    0 0 0 1];

% Update upper layer system
x2_out = x2_in+dR*dx2;

P2_out = F*P2_in*F' + dR*dP*dR';

end
```

Appendix 3:

How to set calibration parameters using Matlab without compiling MIMU22BL's firmware

Calibration file consists of fused accelerometer bias (ACC_BIAS) and fused gyroscope bias (GYRO_BIAS). It also consists of calibration matrix. The first 18 elements of each row is the calibration gain and the next 6 element is the individual bias of each IMUs. MIMU22BL consists of 4 IMUs so rest of the elements 28 calibration rows is 0. An example of the calibration file is given below.

```
#define FLOG2_NR_IMU 2
#define ACC_BIAS {  321455, -381480, -1784994 }
#define GYRO_BIAS { -109378,  807557, -314605 }
#define CALIBRATION_MATRIX \
{ \
 { 32737,    0,    0,   -21,  32889,    0,  113,   -76,  32779,  32768,    0,    0,    0,  32768,    0,    0,    0,  32768, -189318, -
1285370, -10636239, 217070, 771033,  89464 },\
  {   -258, -32733,    112, -32949,    259,   -115,    127,   -270, -32657,   -258, -32766,    112, -32766,    257,   -115,    113,   -113, -
32767, 1031829, 231703, 2583073, -332858, 132476, -843920 },\
  {  32582,   -44,   181,    43,  32848,   -47,  -408,     4,  32552,  32767,   -44,   182,    44,  32768,   -47,  -182,    48,  32767, -
233539, -870378, -4025710, -585304, -602875,  44841 },\
  {     56, -32663,    128, -32840,    -62,   -281,    420,     96, -32611,     60, -32767,    129, -32765,    -62,   -280,    281,   -128, -32765,
727064, 381943, 4940699, 261427, 2930711, -535587 },\
  {0, 0, 0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},\
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
```

```
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, \
```

Each of the values represented in the above example is in float (4 bytes). The example file is converted to bytes, so each element of the example calibration file consists of 4 bytes each. So the fused accelerometer and gyroscope bias consists of (3x4) 12 bytes each. For e.g. the first element of ACC_BIAS is 321455 and that of GYRO BIAS is

Below are the examples

(i) 321455 = 0x4E7AF;

Hexadecimal	00	04	E7	AF
Decimal	0	4	231	175

Now 321455 is replaced by 0 4 231 175

(ii) -109378 = 0XFFFE54BE

Hexadecimal	FF	FE	54	BE
Decimal	255	254	84	190

Similarly -109378 is replaced by 255 254 84 190

This format is followed for all the elements for calibration output file.

Each row of the calibration matrix consists of (18x4) 72 bytes representing the calibration gain shown below as imu_mask(i) and the individual IMU bias consists of (6x4) 24 bytes each depicted as imu_bias(i) .

```
f_log2 = 4;
f_bias = [0 4 231 175 255  25 0 45 216 255  228 195 94];
g_bias = [255 254 84 190  0 12 82 133 255 251 51 19];
imu_pos0 = 0;
imu_pos1 = 1;
imu_pos2 = 2;
imu_pos3 = 3;

imu_mask0 = [0 0 127 225   0 0 0 0   0 0 0 0   255 255 255 235   0 0 128 121   0 0 0 0   0 0 0 113255 255 255 1800 0 128 11 0 0 128 0   0 0 0 0
0 0 0 0   0 0 0 0  0 0 128 0  0 0 0 0   0 0 0 0   0 0 0 0   0 0 128 0];
imu_mask1 = [255 255 254 254255 255 128 350 0 0 112255 255 127 750 0 1 3255 255 255 1410 0 0 127   255 255 254242255 255 128111255
255 254 254   255 255 128 20 0 0 112255 255 128 20 0 1 1255 255 255 14100 0 113255 255 255 143255 255 128 1        ];
imu_mask2 = [ 0 0 127 70   255 255 255 2120 0 0 1810 0 0 430 0 128 80   255 255 255 209255 255 254 1040 0 0 40 0127 40   0 0 127 255   255
255 255  2120 0 0 1820 0 0 440 0 128 0   255 255 255 209255 255 255 740 0 0 480 0 127 255  ];
imu_mask3 = [0 0 0 56255 255 128 1050 0 0 128   255 255 127184   255 255 255 194   255 255 254231   0 0 1 164   0 0 0 96   255 255 128157
0 0 0 60255 255 128 10 0 0 129   255 255 128 3   255 255 255194   255 255 2542320 0 1 25255 255 255 128255 255 128 3];
imu_bias0 = [255 253 28 122255 236 99 6255 93 180 490 3 79 238   0 11 195 2170 1 93 120];
imu_bias1 = [ 0 15 190 1490 3 137 230 39 106 33255 250 235 1980 2 5 124 255 243 31 112 ];
imu_bias2 = [ 255 252 111 189255 242 184 22255 194 146 146255 247 17 168255 246 205 50 0 175 41];
imu_bias3 = [ 0 11 24 240 5 211 2470 75 99 1550 3 253 510 44 184 23255 247 211 221];
```

Matlab script for loading above calibration file into MIMU22BL is shown below.

```
header_f_log = 5;
header_g_bias = 9;
header_f_bias = 8;
header_c_bias = 37;

command_f_log = [header_f_loglog2(f_log2)];
command_f_log= [command_f_log (sum(command_f_log)-mod(sum(command_f_log),256))/256 mod(sum(command_f_log),256)];
fwrite(com,command_f_log,'uint8');
command_g_bias = [header_g_biasg_bias];
command_g_bias = [command_g_bias (sum(command_g_bias)-mod(sum(command_g_bias),256))/256 mod(sum(command_g_bias),256)];
fwrite(com,command_g_bias,'uint8');
command_f_bias = [header_f_biasf_bias];
command_f_bias = [command_f_bias (sum(command_f_bias)-mod(sum(command_f_bias),256))/256 mod(sum(command_f_bias),256)];
fwrite(com,command_f_bias,'uint8');
command_calib_mat = [header_calib_mat imu_pos0 imu_mask0];
command_calib_mat = [command_calib_mat (sum(command_calib_mat)-mod(sum(command_calib_mat),256))/256
mod(sum(command_calib_mat),256)];
fwrite(com,command_calib_mat,'uint8');
command_calib_mat = [header_calib_mat imu_pos1 imu_mask1];
command_calib_mat = [command_calib_mat (sum(command_calib_mat)-mod(sum(command_calib_mat),256))/256
mod(sum(command_calib_mat),256)];
```

```
fwrite(com,command_calib_mat,'uint8');
command_calib_mat = [header_calib_mat imu_pos2 imu_mask2];
command_calib_mat = [command_calib_mat (sum(command_calib_mat)-mod(sum(command_calib_mat),256))/256
mod(sum(command_calib_mat),256)];
fwrite(com,command_calib_mat,'uint8');
command_calib_mat = [header_calib_mat imu_pos3 imu_mask3];
command_calib_mat = [command_calib_mat (sum(command_calib_mat)-mod(sum(command_calib_mat),256))/256
mod(sum(command_calib_mat),256)];
fwrite(com,command_calib_mat,'uint8');
command_c_bias = [header_c_bias imu_pos0 imu_bias0];
command_c_bias = [command_c_bias (sum(command_c_bias)-mod(sum(command_c_bias),256))/256 mod(sum(command_c_bias),256)];
fwrite(com,command_c_bias,'uint8');
command_c_bias = [header_c_bias imu_pos1 imu_bias1];
command_c_bias = [command_c_bias (sum(command_c_bias)-mod(sum(command_c_bias),256))/256 mod(sum(command_c_bias),256)];
fwrite(com,command_c_bias,'uint8');
command_c_bias = [header_c_bias imu_pos2 imu_bias2];
command_c_bias = [command_c_bias (sum(command_c_bias)-mod(sum(command_c_bias),256))/256 mod(sum(command_c_bias),256)];
fwrite(com,command_c_bias,'uint8');
command_c_bias = [header_c_bias imu_pos3 imu_bias3];
command_c_bias = [command_c_bias (sum(command_c_bias)-mod(sum(command_c_bias),256))/256 mod(sum(command_c_bias),256)];
fwrite(com,command_c_bias,'uint8');
fwrite(com,[50 0 50],'uint8'); % Stop processing
fwrite(com,[34 0 34],'uint8'); % Stop output (and empty buffers)
whilecom.BytesAvailable
fread(com,com.BytesAvailable,'uint8');
end
fclose(com);
```

Here header_f_log = Log_2(No. of IMUs)