Osmium

Integration Guide

Revision 1.2

R&D Centre:
GT Silicon Pvt Ltd
D201, Type 1, VH Extension,
IIT Kanpur
Kanpur (UP), India, PIN – 208016

Tel: +91 512 259 5333
Fax: +91 512 259 6177
Email: info@gt-silicon.com
URL: www.gt-silicon.com

Revision History

Revision	Revision Date	Updates
1.0	21 Oct 2014	Initial version
1.1	01 Nov 2014	Included more references in “Purpose & Scope” Some minor typo corrections
1.2	29 Mar 2016	Updated for new version of firmware

Purpose & Scope

This document describes the data processing flow in Osmium MIMU22BT and Osmium MIMU4444. 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 Matlab interfacing codes for better understanding.

Please refer for architecture, design and algorithm details: **John-Olof Nilsson, Amit K Gupta, Peter Handel, "*Foot mounted inertial navigation made easy*", In Proc Indoor Positioning & Indoor Navigation (IPIN), Busan, Korea, 2014.**

Please refer following documents also for specific details:

1. MPU-9150 Register Map and Descriptions Revision 4.2
2. MPU-9150 Product Specification Revision 4.3
3. 32-bit AVR Microcontorller Specification

Introduction

Osmium MIMU22BT and Osmium MIMU4444 are Multi-IMU based inertial navigation modules. MIMU22BT, with on-board four 9-DOF IMUs, is targeted towards foot mounted pedestrian application, whereas MIMU4444, with on-board thirty two 9-DOF IMUs, is an ideal platform for carrying out research in motion sensing by using Sensor Fusion and Array Signal Processing methods.

Osmium MIMU22BT has wireless interface (Bluetooth 3.0) 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.

Osmium MIMU22BT can also be used as a normal wireless IMU for other non foot mounted applications. 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]

The Osmium MIMU4444 on the other hand supports USB communication only.

This document describes the data processing flow in Osmium MIMU22BT and Osmium MIMU4444. 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.

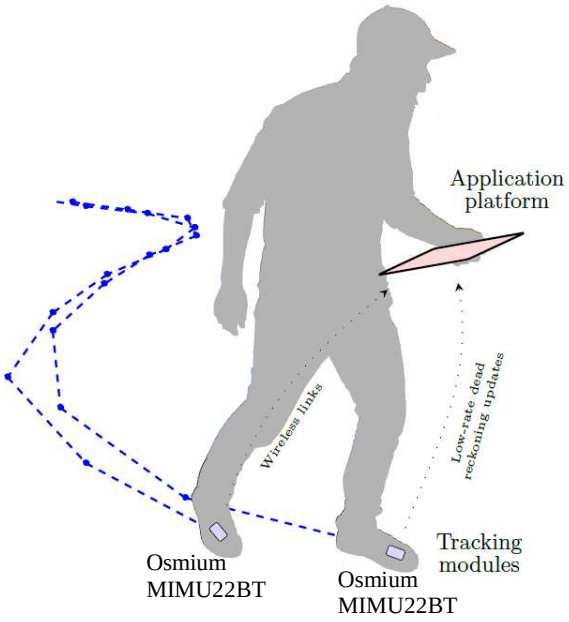


Fig. 1: Tracking with foot-mounted Osmium MIMU22BT and data collection using BlueTooth

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

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.

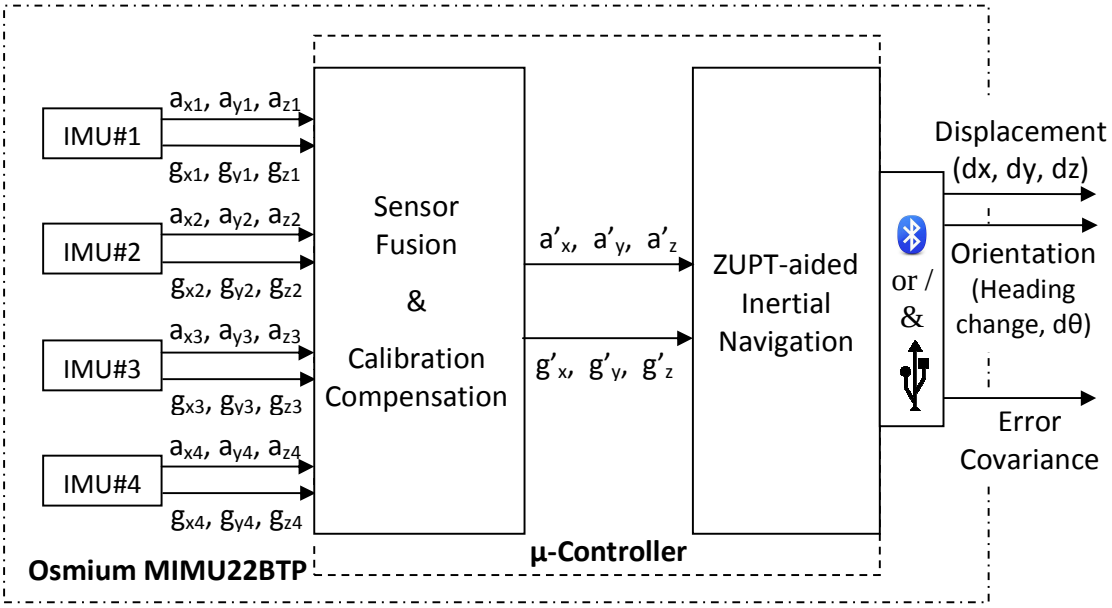


Figure 2 Illustration of data flow sequence in Osmium MIMU22BT with ZUPT-aided inertial navigation. *Rotation of the coordinate system w.r.t. the original one.

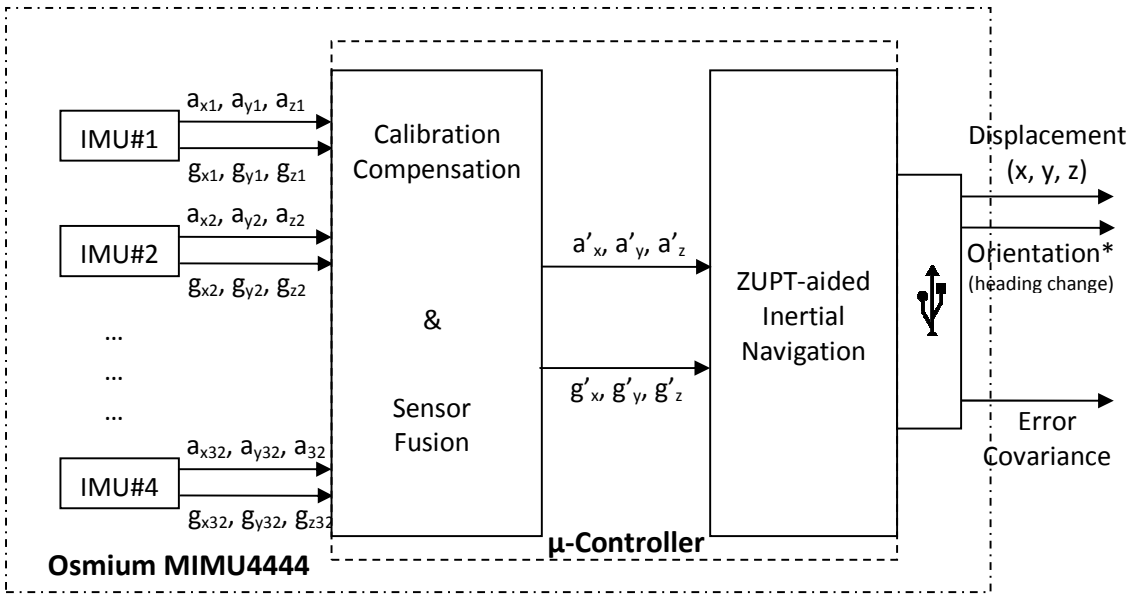


Figure 3 Illustration of data flow sequence in Osmium MIMU22BT with ZUPT-aided inertial navigation. *Rotation of the coordinate system w.r.t. the original one.

Communication Protocol

The interfacing application platform (Computer, Phone, Tab etc) sends a particular command (data request) to MIMU22BT / MIMU4444 who serves the request by sending out required data to the application platform. Each command refers to a particular state, which can send out variety of data depending upon command options.

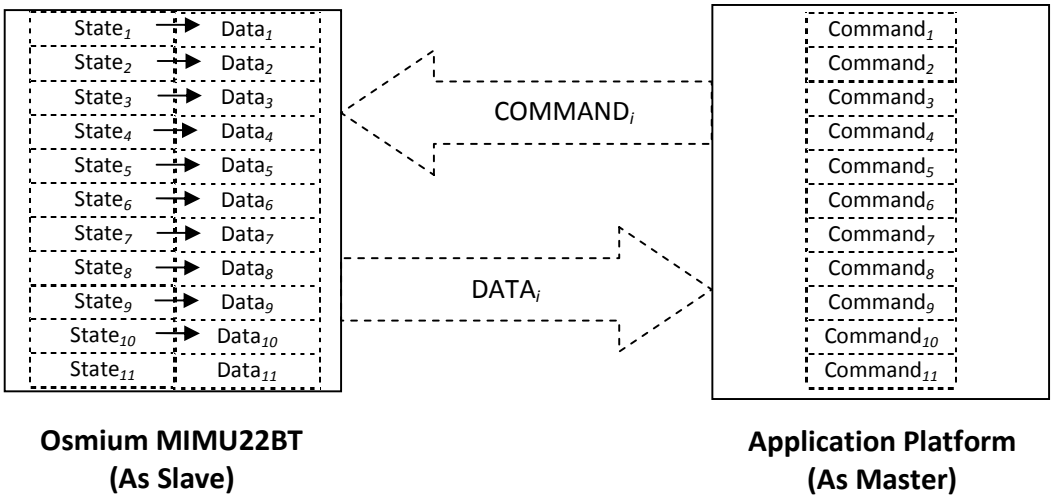


Figure 4 Data communication protocol between Osmium MIMU22BT and the application platform

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

Checksum is of two bytes. Last two bytes in any command are checksum.

Table 1 Summary of some useful commands

S. No.	Command	Description	**Syntax** (in decimal)
1	Stepwise Dead Reckoning (PDR)	The Osmium device 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	Processing off	The Osmium device (controller) switches off all the processing. It makes the	[50 0 50] cs1 = 0 cs2 = 50

		processing sequence empty	
3	All output off	The Osmium device turns off output of all the states and stops transmitting data to the application platform	[34 0 34] cs1 = 0 cs2 = 34
4	Read inertial data	The Osmium starts sampling onboard IMUs' (acceleros' and gyros' data)	[48 19 0 0 67] cs1 = 0 cs2 = 67
5	Output inertial data	The Osmium starts transmitting the inertial sensors' raw data a_{xi}, a_{yi}, a_{zi}, g_{xi}, g_{yi}, g_{zi} of the selected IMU_i, 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 are used to select IMUs. pl5* is the actual rate divider + 64. cs1, cs2: checksum
6	Read IMU temperature data	The Osmium starts reading internal temperature of the onboard IMUs'	[53 0 53] cs1 = 0 cs2 = 53
7	Output IMU temperature data	The Osmium 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 actual rate divider + 128. cs1, cs2: checksum
8	High precision IMU (Normal IMU)	The Osmium 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

*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 the Osmium device to the application platform. The maximum transmission rate is 1 KHz 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	1000
2	500
3	250
4	125
5	62.5
6	31.25
7	15.625

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 the Osmium device for stepwise dead reckoning with an (Android) application platform, communicating wirelessly. *[Though the description under this section has used MIMU22BT / MIMU22BTP as reference, everything is equally applicable to MIMU4444 / MIMU4X4C also.]*

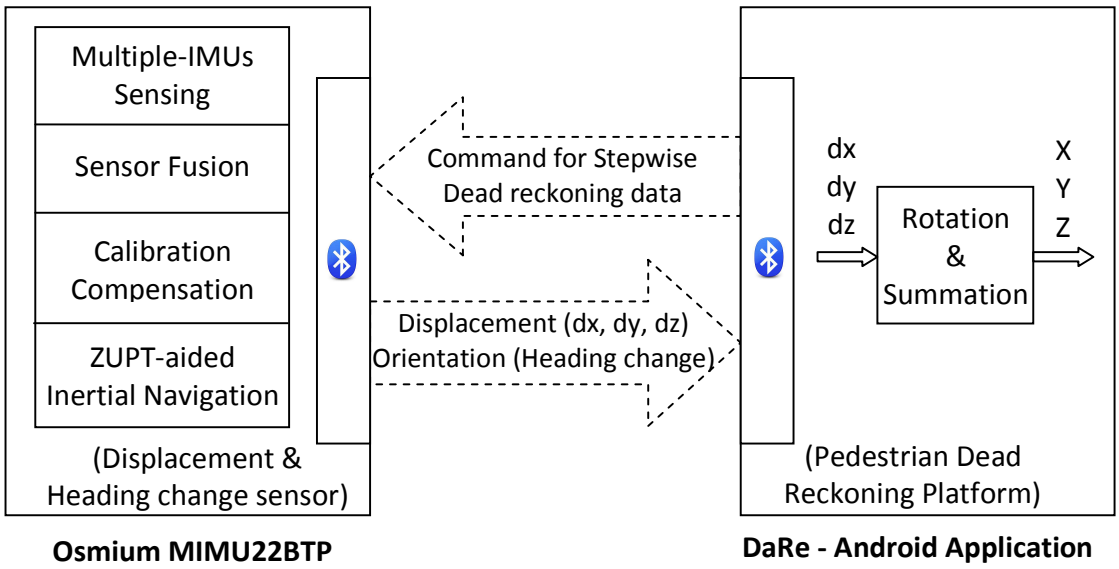
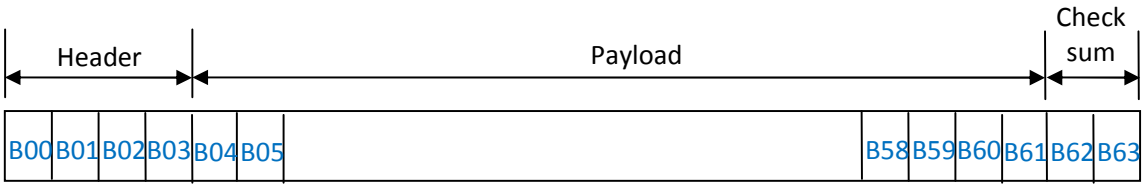


Figure 5 Interfacing of Osmium MIMU22BT with an application platform

The Osmium device sends 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 Osmium devices in 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 the Osmium device goes down below predefined threshold. Hence it is the number of steps taken by the wearer, if the Osmium device is 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

The Osmium device 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 the Osmium device. It is rotation of the x-y plane about the z-axis, and z-axis is always aligned with the gravity. The da 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 The Osmium MIMU 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
   Command: [52 0 52]

   The Osmium device sends out data packet for stepwise dead reckoning in response to this command.

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

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

3. **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.

4. **State**: Read Inertial Data
   Command: [48 19 0 rd cs1 cs2]

   This command is used to initiate sampling onboard inertial sensors' data by the Osmium device.

   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 = 1 KHz)

Recommended for Bluetooth

[48 19 0 0 67]

[40 0 0 0 15 68 0 124] (o/p data rate: 4 = 125 Hz)

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

   The Osmium device 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 64 + the actual output rate divider (1, 2, 3,… etc).

   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 MIMU22BT/MIMU22BTP 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 = 1 KHz)

   Recommended for Bluetooth
   [48 19 0 0 67]

[40 0 0 0 15 68 0 124] (o/p data rate: 4 = 125 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

6. **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.

7. **State**: Output IMU Temperature Data
   Command: [40 pl1 pl2 pl3 pl4 pl5 cs]

The Osmium device 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 = 1 KHz)

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

   With this command, the Osmium devices 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 MIMU22BT 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

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;
while abort_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);
          if isempty(pdef)
             chol(dP,'lower');
          end
          [x_sw P_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_phi  cos_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
```