OPTICALLY ENHANCED ATTITUDE AND HEADING REFERENCE SYSTEM

OptoAHRS

Software Development Kit (SDK) Manual

Revision 1.1

Inertial Labs Inc.[TM] Address: 39959 Catoctin Ridge Street, Paeonian Springs, VA 20129, USA
Tel: +1 (703) 880-4222, Fax: +1 (703) 991-5378 Website: www.inertiallabs.com

Table of Contents

OptoAHRS SDK Versions

Version	Date	By	Changes
2.1.17.21	Dec. 07, 2011	TvD	First version
2.1.18.23	Mar. 26, 2012	TvD	1) Removed the bug causing OptoAHRS incorrect heading output (i.e. jumps by 295°) 2) Removed the OptoAHRS output angle data smoothing filter 3) Added an additional adjustment to optics azimuth output data
4.0.2.3	May 21, 2012	TvD	**1) Changed the function definitions. Now, in place of calling a function with many parameter, several functions with small number of parameters shall be called** 2) Extended the parameter block (from 750 bytes up to 2 Kbytes) 3) Added the Qt library dependencies (the optics software is redeveloped with C++) 4) Added functions to run the calibrations 5) Added options to select the applicable magnetometer correction parameter set before and during operation (Start with… and Continue with…) 6) Added a new parameter to the AHRS memory
4.0.6.6	June 1, 2012	TvD	1) Added the creation of a *.csv file to save the zone calibration results 2) Removed the bug causing saving incorrect magnetic heading into the *.csv file created for "fires"
4.0.7.7	June 7, 2012	TvD	**1) Redeveloped the OptoAHRS SDK_Demo with C# and removed all the other demo versions** 2) Removed the possibility to set the position from within the calibration options. Now the program reads the position from the AHRS memory 3) Removed the callback functions (from the Pointer_SDK_manual) 4) Removed the record in the *.csv file when calling P_GetCurrentOutputs since now it is used in SDK_Demo in place of callback 5) Removed the mistake causing zeroing the OptoAHRS parameters in case of an unstopped process in the previous start (if the OptoAHRS was not stopped for some reason, the next start zeroed the parameters) 6) Added an additional call of P_Stop() before P_Close() in case of a running stream to stop it
4.0.9.9	June 9, 2012	TvD	Added the P_Result P_Fire (float TargetHeading) function which, when called, writes the current OptoAHRS output

			(averaged over the 100 last samples)into the *.csv file. The Pointer_SDK_Demo interface is supplemented with the corresponding button**(Fire)**.
4.0.10.10	June 11, 2012	TvD	1) Added record of magnetic declination into the SDKZoneClb*.csv and FlyZoneClb*.csv files. 2) Record of true geographical azimuth instead of magnetic in mentioned above files.
4.0.11.11	June 12, 2012	TvD	Wrong taking into account of magnetic deviation bugs fixed.
4.0.14.14	June 21, 2012	TvD	1) Pointer_SDK_Demo has been renamed into OptoAHRS_Demo. 2) Boresighting functions added. 3) **Caution: current output data structure was changed – azimuth and elevation angles were added.** 4) After each using of P_Fire function current frames are writing into *<current-date-time>*.vrw file.
4.0.17.17	July 2, 2012	TvD	1) Boresighting procedure improved. 2) Added creation of reference frames during zone calibration.
4.0.20.20	July 11, 2012	TvD	Added new type for a parameter InitClbType – "Simple Clb" while others have been incremented.
4.0.26.26	August 2, 2012	TvD	Magnetic field calibrations improved.
15.0.1.1	August 21, 2012	TvD	1) Pointer SDK renamed into OptoAHRS SDK. 2) Library was optimized and compiled in C++ for the first time. 3) Fly-zone calibration was simplified. Functions P_FlyZoneClbClear, P_FlyZoneClbAdd andP_FlyZoneClbAccept were deleted, P_AllowFlyZoneClb – added.
15.0.2.2	September 4, 2012	TvD	New function P_GetReferenceFrame was introduced. Auto creation of reference frames implemented. Performance improved.
15.0.3.3	September 12, 2012	TvD	OptoAHRS algorithms were improved.
15.0.4.4	September 13, 2012	TvD	New parameter RFlimit was added.
15.0.6.7	September 22, 2012	TvD	Some improvements in optical algorithms.
15.0.7.9	September 25, 2012	TvD	Crash error sometimes appeared after OptoAHRS stop was eliminated. Improvement of reference frames auto creation algorithm. Deleted recording into TestFullData*.csv during reference frames creation (but not during firing which still records).
15.0.13.23	November 13, 2012	TvD	Function P_GetAdditionalInfo was introduced. Several improvements in creation of reference frames.
15.0.16.448	December 3, 2012	TvD	Optics auto correction algorithms. Function P_EnableDebugLog was introduced.
15.0.19.459	December 14, 2012	TvD	Several improvements in auto creation of reference frames.

			New parameter RefMode was introduced.
15.0.26.476	March 5, 2013	SD	The principal OptoAHRS KF algorithm was streamlined.
15.0.28.486	April 29, 2013	SD	1) The library Optics.dll was replaced with libvisualgyro.dll. 2) A bug in the OptoAHRS bore-sighting algorithm was fixed: The 0.05-deg resolution of the OptoAHRS angles was removed.
15.0.29.505	July 4, 2013	SD	1) The manual reference frame creation function was transferred into the optic thread. 2) The recalculation of the OptoAHRS Euler angles into the tube azimuth and elevation and vice versa was implemented. 3) The possibility to perform "fires" and to create reference frames manually based on the tube azimuth was added. Also a file named TestFullTubeData-<DateTime>.csv is created to write tube azimuths and elevations. 4) The following new functions were added: – P_TubeFire(…) to perform a "fire" based on tube azimuth – P_RecalcOptoIMUHeadingIntoTubeAzimuth(…) to recalculate OptoAHRS heading into tube azimuth based on specified Euler angles – P_RecalcTubeAzimuthIntoOptoIMUHeading(…) to recalculate tube azimuth into OptoAHRS heading **Caution:** – **The** P_Reference **structure was changed: Roll and pitch angles were added.** – **The** P_CurrentOutputs **structure was changed: The OptoIMUHBR, OptoIMUPBR, OptoIMURBR fields were added.**
15.0.30.517	August 30, 2013	SD	The Extrinsics angles (misalignments between the camera and OptoAHRS axes) were added into the algorithm.
15.0.33.527	October 23, 2013	SD	Software 2x2 image binning was implemented to be used when hardware binning is off.
15.0.42.550	January 27, 2014	SD	Several new functions were added: – P_LoadParameters(char *PrmFileName): to load a parameter file into the device memory – P_ComputeMagneticDeclination(): to calculate magnetic declination. – P_BoresightingSetAngles() and P_BoresightingGetAngles(): to write and read tube's offset angles
15.0.44.553	February 19, 2014	SD	The error of pulling optic angles down to zero upon recovering optic orientation validity was passed around.
15.0.48.565	June 20, 2014	SD	1) The crash error due to creation of too many reference frames was fixed. 2) Optical obstruction detection was added.
15.0.49.574	July 29, 2014	SD	LoopClosure flag was added: LoopClosure is set to 1 upon Loop Closure occurs. **Caution:** Data structure was changed

1. Overview

This document gives a high-level description of the API to be used with the OptoAHRS SDK in pseudo-code form. The API is precisely defined in demo projects sources accompanying the SDK.

2. System Requirements

For correct OptoAHRS operation, the OptoAHRS SDK requires the following:

- 1.5 GHz or faster processor
- 1 Gb internal RAM
- 256 Mb video RAM
- Windows 7, Windows Vista, Windows XP SP3
- OpenGL 2.0 and later
- DirectX 9.0b or higher
- Microsoft Visual C++ 2008 Redistributable Package
- Qt Libraries (libgcc_s_dw2-1.dll, mingwm10.dll, QtCore4.dll)
- iCube Camera Device Driver
- iCube Camera API (ICubeSDK.dll)
- Optic Library (libvisualgyro.dll)

3. Data Types

Enumerations	Description

enum P_Result {
P_SUCCESS = 0x00,
P_ERROR = 0x01
};
	Return values of API functions

enum P_ClbType {
CLB_2D = 0x11,
CLB_2D2T = 0x12,
CLB_3D = 0x13,
CLB_ZONE = 0x14
};
	Calibration types

enum P_ClbStatusBits {
CS_SUCCESS = 0x0001,
CS_IS_STARTED = 0x0002,
CS_INIT_ALIGNMENT = 0x0004,
CS_DATA_ACCUMULATING = 0x0008,
CS_DATA_CALCULATING = 0x0010,
CS_NEXT_REQUESTED = 0x0020,
CS_STOP_REQUESTED = 0x0040,
CS_ACCEPT_REQUESTED = 0x0080,
CS_EXIT_REQUESTED = 0x0100
};
	Calibration status bits

enum P_ClbBiasType { BT_DEGS = 0x00, BT_MILS = 0x01	Heading bias types for zone 3D calibration

};
enum P_UsedClbType {
UC_SIMPLE_CLB = 0x00,
UC_FACTORY_CLB = 0x01,
UC_2D_2T_CLB = 0x02,
UC_ZONE_CLB = 0x03,
UC_AUTO_CLB = 0x04
};
	Used calibration types

enum P_BoresightingStatusBits {
BS_STEP1_STARTED = 0x0001,
BS_STEP1_INIT_ERROR = 0x0002,
BS_STEP1_COMPLETED = 0x0004,
BS_STEP2_STARTED = 0x0008,
BS_STEP2_INIT_ERROR = 0x0010,
BS_STEP2_CAN_STOP = 0x0020,
BS_STEP2_OPTIC_ERROR = 0x0040,
BS_STEP2_COMPLETED = 0x0080
}
	Boresighting status bits

enum P_ReferencesStatusBits {
RSB_AUTO_UNDEFINED = 0x00,
RSB_AUTO_CREATING = 0x01,
RSB_AUTO_CORRECTING = 0x02,
RSB_AUTO_COMPLETED = 0x04
};
	References status bits

Structures	Description

#pragma pack(1)
struct P_Params{
float Mdec;
float Latitude;
float Longitude;
float Altitude;
float Mdate;
uchar ClbInitType;
uchar RFlimit;
char Reserved0;
char Reserved1;
};
	Parameters data

#pragma pack(1)
struct P_CurrentOutputs{
float Azimuth; // Tube azimuth angle
float Elevation; // Tube elevation angle
float OptoIMUH; // OptoAHRS heading after rounding
float OptoIMUP; // OptoAHRS pitch after rounding
float OptoIMUR; // OptoAHRS roll after rounding
float OptoIMUHBR; // OptoAHRS heading before rounding
float OptoIMUPBR; // OptoAHRS pitch before rounding
	Current output data

float OptoIMURBR; // OptoAHRS roll before rounding
float IMUH; // AHRS heading
float IMUP; // AHRS pitch
float IMUR; // AHRS roll
float OpticH; // Optic heading
float OpticP; // Optic pitch
float OpticR; // Optic roll
float AccMagH; // heading, pitch and
float AccMagP; // roll calculated by
float AccMagR; // accelerometers and magnetometers
uchar MagInterference; //magnetic interference
uchar OptInterference; //optical interference
float Vdd; // supply voltage
ushort USW; // status word
};	

#pragma pack(1)
struct P_AdditionalInfo {
short RefNum;
short RefId;
short RefStatus;
uchar RefMode;
uchar LoopClosure;
uchar Reserved[92];
};
	Additional info data

#pragma pack(1) struct P_Reference{ int Ref_ID; float Heading; float Pitch; float Roll; };	Reference data

4. Initialization

There are several functions available that should be called before starting the OptoAHRS in order to initialize its parameters.

Opening OptoAHRS SDK

```
P_Result P_Open()
```

This function allocates memory from the operating system and thus should be called ones only at the very beginning of the work with the OptoAHRS SDK library.

Setting Serial Port

```
P_Result P_SetPortNumber(unsigned short PortNumber)
```

It is used for setting necessary serial port OptoAHRS connected to.

Allowing Data Saving

```
P_Result P_AllowDataSaving(bool Allow)
```

This function enables/disables permission for data saving in all functions mentioned below.

***Note*: data saving is disabled on default.**

Allowing Saving a Log

```
P_Result P_EnableDebugLog(boolEnable)
```

This function enables/disables permission for saving a debug log which helps to eliminate application bugs.

***Note*: log saving is disabled on default. If data saving is disabled then this function doesn't influence.**

Allowing Data Writing

```
P_Result P_AllowWriting(bool Allow)
```

This function starts/stops writing data into a binary file.

***Note*: If data saving is disabled then this function doesn't matter.**

Allowing Auto Calibration

```
P_Result P_AllowAutoCalibration(bool Allow)
```

This function enables/disables permission for auto calibration during operation.

Allowing Fly-zone Calibration

```
P_Result P_AllowFlyZoneCalibration(bool Allow)
```

This function enables/disables permission for fly-zone calibration during operation.

Setting Camera Preview

```
P_Result P_SetCameraPreview(HWND ViewHandle)
```

This function makes it possible to preview camera images in a window with indicated descriptor. ***Note*: if ViewHandle = 0 then preview is disabled (on default).**

5. OptoAHRS Parameters

There are four functions available to work with the OptoAHRS parameters:

Getting Current Parameters

```
P_Result P_GetParams(P_Params *Params)
```

Setting Necessary Parameters

```
P_Result P_SetParams(P_Params *Params)
```

Restoring Parameters (Loading a Set of OptoAHRS Parameters into the Device)

```
P_Result P_LoadParameters(char *PrmFileName)
```

Compute Magnetic Declination Parameter

```
P_Result P_ComputeMagneticDeclination (
    float latitude, float longitude, float altitude,
    int year, int month, int day, float *magDeclination)
```

6. Calibrations

There are several functions intended for magnetic field calibrations (2D-2T, 3D, 2D and Zone 3D).

Setting Accumulation Time

```
P_Result P_SetClbAccumulationTime
(unsigned short AccumulationTime)
```

A call to this function sets data accumulation time (in seconds) for a calibration.

Setting Reference Azimuth

```
P_Result P_SetClbRefAzimuth(float RefAzimuth)
```

A call to this function sets reference azimuth (for the Zone 3D calibration).

Setting Azimuth Shift

```
P_Result P_SetClbRefAzimuth(float RefAzimuth)
```

A call to this function sets azimuth shift relative to reference azimuth (for the Zone 3D calibration).

Getting Calibration Status

```
P_Result P_GetClbStatus(unsigned short *Status)
```

This function gets current calibration status to indicate the following statuses. A received status contains hints in each bit:

Bit Number	Description
0	If == 1 then current calibration is successful
1	If == 1 then calibration is in progress
2	If == 1 then initial alignment is in progress
3	If == 1 then data are accumulating.
4	If == 1 then accumulated data are being calculated.
5	If == 1 then function P_ClbNext is available
6	If == 1 then function P_ClbStop is available
7	If == 1 then function P_ClbAccept is available
8	If == 1 then function P_ClbExit is available

Getting Created Reference Frames Status

P_Result P_GetClbRFStatus(unsigned short *Status)

This function gets the reliability of created reference frames during zone calibration. Each bit of a received Status contains the reliability of a corresponding reference frame (e.g. Status=18 (10010 in binary) means the reference frames in positions 2 and 5 – unreliable).

Clearing Calibration Parameters

P_Result P_ClbClear()

A call to this function clears the current magnetic field calibration parameters.

Starting Calibration

P_Result P_ClbStart(P_ClbTypeClbType)

This function starts the selected calibration sequence.

Continuing Calibration

P_Result P_ClbNext()

This function is only for 2D-2T and Zone 3D calibrations when next positions are required.

Stopping Calibration

P_Result P_ClbStop()

This function stops data accumulation and starts data calculation.

Accepting Calibration

P_Result P_ClbAccept()

This function saves the calibration parameters and terminates the calibration.

Exiting Calibration

P_Result P_ClbExit()

This function terminates calibration without saving calibration parameters.

7. Boresighting Procedure

There are several API functions intended for the boresighting procedure:

Setting Boresighting Angles

```
P_Result P_BoresightingSetAngles(float heading, float pitch)
```

Getting Boresighting Angles

```
P_Result P_BoresightingGetAngles(float *heading, float     *pitch)
```

Getting Boresighting Status

```
P_Result P_BoresightingGetStatus(unsigned short *Status)
```

This function gets current boresighting status to indicate the following statuses. The received status contains hints in each bit:

Bit Number	Description
0	If == 1 then step1 is currently running
1	If == 1 then step1 data accumulation failed
2	If == 1 then step1 data accumulation succeeded
3	If == 1 then step2 is currently running
4	If == 1 then step2 data accumulation failed
5	If == 1 then function P_BoresightingStopStep2 is available
6	If == 1 then step2 optic data accumulation failed
7	If == 1 then step2 data accumulation succeeded

Starting Boresighting Step1 and Step2 Respectively

```
P_Result P_BoresightingStartStep1 (short ElevationMils)
```

```
P_Result P_BoresightingStartStep2 ()
```

where **ElevationMils** is the initial elevation set in mils.

Stopping Step2

```
P_Result P_BoresightingStopStep2()
```

This function calls the P_BoresightingStartStep2 function and waits for the initial alignment completion when status bit (5) is set to 1.

Accepting Boresighting Results

```
P_Result P_BoresightingAccept()
```

This function saves the results of boresighting and saves them into the OptoAHRS memory.
***Note*: This function is available only when step1 and step2 data accumulation are successful bit (2) and bit (7) are set to 1.**

Exiting Boresighting Procedure

```
P_Result P_BoresightingExit()
```

This function terminates the boresighting procedure without saving its results.

8. OptoAHRS Operation

To start the OptoAHRS, the following API function is available:

StartingOptoAHRS

```
P_Result P_StartOptoIMU()
```

Upon this and until operation is stopped, there are several functions available:

Getting Output Data

```
P_Result P_GetCurrentOutputs(P_CurrentOutputs *Data)
```

Getting Additional Info

```
P_Result P_GetAdditionalInfo(P_AdditionalInfo *Info)
```

"Firing"

```
P_Result P_Fire(float TargetHeading)
```

A call to this function makes the current output data to be automatically saved to the *.csv file.

"Tube Firing"

```
P_Result P_TubeFire(float TubeAzimuth, float Tubeelevation)
```

A call to this function automatically saves the current azimuth and elevation angles to the TestFullTubeData -<DateTime>.csv file

All current reference frames can be deleted by calling the following function:

Deleting All References

```
P_Result P_ClearReferences()
```

Upon calling this function, the optical orientation angles will be set to zero.

For correct OptoAHRS operation it is necessary to create at least one reference frame using the following function:

Creating a Reference Manually

```
P_Result P_AddReferenceFrame(float TargetHeading)
```

A call to this function creates a reference frame with a specified geographical target azimuth (in degrees) and two data files:

- *.pgm – picture of accepted reference frame;
- *.csv – full data protocol of the OptoAHRS output at the moment of the reference frame creation.

The following functions can be used to get and set specified reference frames:

Getting the Number of Existing References

```
P_Result P_GetNumOfReferences(int *NumOfReferences)
```

where **NumOfReferences** is the number of existing references

Getting an Existing Reference

```
P_Result P_GetReference
(intNumOfReference, P_Reference *Reference)
```

```
P_Result P_GetReferenceFrame
(intNumOfReference, uchar **Frame, int *Size);
```

Where NumOfReference is the input number of the requested reference; Reference is the requested reference; Frame is the pointer to the reference frame in bitmap format; Size is the size of the mentioned reference frame.

Setting an Existing Reference

```
P_Result P_SetReference(P_Reference *Reference)
```

It's also possible to change the current magnetic field calibration parameter set during operation by calling the following function:

Changing Calibration Type

```
P_Result P_ChangeUsedClbType(P_UsedClbTypeUsedClbType)
```

For recalculating OptoAHRS heading into tube azimuth and back the following function is available:

Recalculating Heading into Tube Azimuth based on OptoAHRS Euler angles

```
P_Result P_RecalcOptoIMUHeadingIntoTubeAzimuth(P_Reference*
CurrentAngles, float* RecountedAzimuth)
```

where CurrentAngles (input parameter) is the structure containing the three OptoAHRS Euler angles.

RecountedAzimuth (output parameter) is tube azimuth. A caller must pre-allocate memory for this parameter.

If the function returns P_SUCCESS , recounting is successful. RecountedAzimuth contains the recount azimuth value.

Recalculating Tube Azimuth into OptoAHRS Heading Based on OptoAHRS Euler Angles

```
P_Result P_RecalcTubeAzimuthIntoOptoIMUHeading(P_Reference*
CurrentAngles,float* InitialAzimuth,float* RecountedHeading)
```

where CurrentAngles (input parameter) is the structure three angles structure. A caller must pre-allocate memory for this parameter.

To stop OptoAHRS operation, the following function is available:

Stopping OptoAHRS

```
P_Result P_Stop()
```

9. Converter

To convert saved binary files (*.bin, *.par) into text files, the following function is available:

Converting Data

```
P_Result P_ConvertBinParToTxt(char * BinFilename)
```

10. Finalization

Closing OptoAHRS SDK

```
P_Result P_Close()
```

This function closes the OptoAHRS SDK and returns resources to the operating system.

***Note*: Before using this function you should call P_Stop() function mentioned above in order to terminate the operation properly.**

11. API Functions Availability

API Function	Availability
P_Open	Prior opening the OptoAHRS SDK only
P_SetPortNumber	Before starting the OptoAHRS only
P_AllowDataSaving	Upon opening the OptoAHRS SDK only
P_EnableDebugLog	
P_AllowWriting	
P_AllowAutoCalibration	
P_AllowFlyZoneCalibration	
P_SetCameraPreview	Before starting the OptoAHRS only
P_GetParams	
P_SetParams	
P_LoadParameters	
P_ComputeMagneticDeclination	
P_SetClbAccumulationTime	
P_SetClbRefAzimuth	
P_SetClbBias	
P_GetClbStatus	
P_GetClbRFStatus	
P_ClbClear	
P_ClbStart	
P_ClbNext	
P_ClbStop	

P_ClbAccept	
P_ClbExit	
P_BoresightingSetAngles	
P_BoresightingGetAngles	
P_BoresightingGetStatus	
P_BoresightingStartStep1	
P_BoresightingStartStep2	
P_BoresightingStopStep2	
P_BoresightingAccept	
P_BoresightingExit	
P_StartOptoIMU	
P_GetCurrentOutputs	During OptoAHRS operation only
P_GetAdditionalInfo	
P_Fire	
P_TubeFire	
P_AddReferenceFrame	
P_ClearReferences	
P_GetNumOfReferences	
P_GetReference	
P_GetReferenceFrame	
P_SetReference	
P_ChangeUsedClbType	
P_RecalcOptoIMUHeadingIntoTubeAzimuth	
P_RecalcTubeAzimuthIntoOptoIMUHeading	
P_Stop	
P_ConvertBinParToTxt	Upon opening OptoAHRS SDK only
P_Close	

Appendix A. Proper Sample Sequences to Perform the Magnetometer Calibrations

A.1. 3D Calibration Procedure

Step 1) before starting calibration in order to allocate memory from the operating system you should call the following function

```
P_Result   P_Open()
```

Step 2) it's also vital to set proper serial port by using

```
P_Result   P_SetPortNumber   (unsigned short PortNumber)
```

where **PortNumber** – necessary serial port number.

Step 3) you should set necessary data accumulation time

```
P_Result   P_SetClbAccumulationTime
        (unsigned short AccumulationTime)
```

where **AccumulationTime** sets in seconds.

Step 4) the next thing is to set current coordinates by using

```
P_Result   P_SetClbCoordinates
        (float Latitude, float Longitude,
        float Altitude, float Date)
```

where **Latitude** and **Longitude** set in degrees, **Altitude** – in meters, Date – in years (e.g. May-20-2012 = 2012 + 5 / 12 + 20 / 365 = 2012.4714).

Step 5) after this moment you can start calibration

```
P_Result   P_ClbStart(P_ClbType)
```

where **P_ClbType** for 3D calibration is CLB_3D = 0x13.

Step 6) once you have started 3D calibration in order to get current calibration status you should call the following function from time to time

```
P_Result   P_GetClbStatus(unsigned short *Status)
```

where **Status** contains information about actions available currently (full explanation of status bits see in the end of this document).

If you used this function right after starting calibration, received Status is equal to **"00000000 00000110"** which means that calibration and initial alignment are currently in progress. Note, that you shouldn't move the device until initial alignment is done (bit(2) == 0).

After initial alignment is done, Status will change to **"0000000**1 **0**1**00**1**0**1**0"** which means that data are accumulating and you can also stop calibration by calling P_ClbStop() function or terminate calibration procedure by using P_ClbExit(). In this stage you should rotate the device. During the data accumulation the weapon should be rotated in full azimuth, pitch and roll ranges.

Step 7) as it has been said before, you can stop data accumulating by using

```
P_Result   P_ClbStop()
```

which terminates data accumulating and moves to step 9.

Step 8) you are also eligible to terminate the whole calibration procedure by using

```
P_Result   P_ClbExit()
```

Step 9) after data have been accumulated or P_ClbStop() function has been called calibration status will change to **"00000000 0001**0**0**1**0"** which means that accumulated data are currently calculating and you should wait until this process is done. In this stage you can stop rotating the device.

Step 10) after calibration data have been calculated calibration status will change to **"0000000**1 **1**0000**0**11**"** or **"0000000**1 **1**0000**0**10**"** where bit(0) indicates calibration success and other non-zero bits mean that it is expected that you will call whether P_ClbExit() function without saving calibration results or accept them by using P_ClbAccept() function.

Step 11) if you are satisfied with calibration results you can save them by using

```
P_Result   P_ClbAccept()
```

Step 12) after the calibration is done you are expected to finalize working with Pointer_SDK by using

```
P_Result   P_Close()
```

This function closes Pointer_SDK and returns resources to the operating system.

Table A1: Status Bit Description

Bit Number	Description
0	If == 1 then current calibration is successful
1	If == 1 then calibration is in progress
2	If == 1 then initial alignment is in progress
3	If == 1 then data are accumulating.
4	If == 1 then accumulated data are calculating.
5	If == 1 then function P_ClbNext is available
6	If == 1 then function P_ClbStop is available
7	If == 1 then function P_ClbAccept is available
8	If == 1 then function P_ClbExit is available

A.2. 2D Calibration Procedure

Step 1) before starting calibration in order to allocate memory from the operating system you should call the following function

```
P_Result   P_Open()
```

Step 2) it's also vital to set proper serial port by using

```
P_Result   P_SetPortNumber   (unsigned short PortNumber)
```

where **PortNumber** – necessary serial port number.

Step 3) you should set necessary data accumulation time

```
P_Result   P_SetClbAccumulationTime
        (unsigned short AccumulationTime)
```

where **AccumulationTime** sets in seconds.

Step 4) the next thing is to set current coordinates by using

```
P_Result   P_SetClbCoordinates
        (float Latitude, float Longitude,
        float Altitude, float Date)
```

where **Latitude** and **Longitude** set in degrees, **Altitude** – in meters, Date – in years (e.g. May-20-2012 = 2012 + 5 / 12 + 20 / 365 = 2012.4714).

Step 5) after this moment you can start calibration

```
P_Result   P_ClbStart(P_ClbType)
```

where **P_ClbType** for 2D calibration is CLB_2D = 0x11.

Step 6) once you have started 2D calibration in order to get current calibration status you should call the following function from time to time

```
P_Result   P_GetClbStatus(unsigned short *Status)
```

where **Status** contains information about actions available currently (full explanation of status bits see in the end of this document).

If you used this function right after starting calibration, received Status is equal to **"00000000 00000110"** which means that calibration and initial alignment are currently in progress. Note, that you shouldn't move the device until initial alignment is done (bit(2) == 0).

After initial alignment is done, Status will change to **"00000001 01001010"** which means that data are accumulating and you can also stop calibration by calling P_ClbStop() function or terminate calibration procedure by using P_ClbExit(). In this stage you should rotate the device. Rotate weapon in azimuth with pitch and roll angles close to zero as possible. This rotation must include one or more full 360 deg turns.

Step 7) as it has been said before, you can stop data accumulating by using

```
P_Result   P_ClbStop()
```

which terminates data accumulating and moves to step 9.

Step 8) you are also eligible to terminate the whole calibration procedure by using

```
P_Result   P_ClbExit()
```

Step 9) after data have been accumulated or P_ClbStop() function has been called calibration status will change to **"00000000 00010010"** which means that accumulated data are currently calculating and you should wait until this process is done. In this stage you can stop rotating the device.

Step 10) after calibration data have been calculated calibration status will change to **"00000001 10000011"** or **"00000001 10000010"** where bit(0) indicates calibration success and other non-zero bits mean that it is expected that you will call whether P_ClbExit() function without saving calibration results or accept them by using P_ClbAccept() function.

Step 11) if you are satisfied with calibration results you can save them by using

```
P_Result   P_ClbAccept()
```

Step 12) after the calibration is done you are expected to finalize working with Pointer_SDK by using

```
P_Result   P_Close()
```

This function closes Pointer_SDK and returns resources to the operating system.

Table A2: Status Bit Description

Bit Number	Description
0	If == 1 then current calibration is successful
1	If == 1 then calibration is in progress
2	If == 1 then initial alignment is in progress
3	If == 1 then data are accumulating.
4	If == 1 then accumulated data are calculating.
5	If == 1 then function P_ClbNext is available
6	If == 1 then function P_ClbStop is available
7	If == 1 then function P_ClbAccept is available
8	If == 1 then function P_ClbExit is available

A3. 2D-2T Calibration Procedure

Step 1) before starting calibration in order to allocate memory from the operating system you should call the following function

```
P_Result   P_Open()
```

Step 2) it's also vital to set proper serial port by using

```
P_Result   P_SetPortNumber   (unsigned short PortNumber)
```

where **PortNumber** – necessary serial port number.

Step 3) you should set necessary data accumulation time

```
P_Result   P_SetClbAccumulationTime
        (unsigned short AccumulationTime)
```

where **AccumulationTime** sets in seconds.

Step 4) the next thing is to set current coordinates by using

```
P_Result   P_SetClbCoordinates
        (float Latitude, float Longitude,
        float Altitude, float Date)
```

where **Latitude** and **Longitude** set in degrees, **Altitude** – in meters, Date – in years (e.g. May-20-2012 = 2012 + 5 / 12 + 20 / 365 = 2012.4714).

Step 5) after this moment you can start calibration

```
P_Result   P_ClbStart(P_ClbType)
```

where **P_ClbType** for 2D-2T calibration is CLB_2D2T = 0x12.

Step 6) once you have started 2D-2T calibration in order to get current calibration status you should call the following function from time to time

```
P_Result   P_GetClbStatus(unsigned short *Status)
```

where **Status** contains information about actions available currently (full explanation of status bits see in the end of this document).

If you used this function right after starting calibration, received Status is equal to **"00000001 00100010"** which means that calibration is currently in progress and that you are expected:

- to call P_ClbNext() function for data accumulating in the next position;
- or call P_ClbExit() function to terminate calibration procedure.

Step 7) to start data accumulating you should set the device in the position with the necessary pitch angle and call the following function

```
P_Result   P_ClbNext()
```

Step 8) you are also eligible to terminate the whole calibration procedure by using

```
P_Result   P_ClbExit()
```

which terminates further calibration and moves to step 15.

Step 9) after calling P_ClbNext() function received Status is equal to **"00000000 00000110"**, which means that initial alignment is currently in progress. Note, that you shouldn't move the device until initial alignment is done (bit(2) == 0).

After initial alignment is done, Status will change to **"00000001 01001010"** which means that data are accumulating and you can also stop calibration by calling P_ClbStop() function or terminate calibration procedure by using P_ClbExit(). In this stage you should rotate the device. Rotate weapon in azimuth with approximately constant pitch and roll angles as possible. This rotation must include one or more full 360 deg turns.

Step 10) as it has been said before, you can stop data accumulating by using

```
P_Result   P_ClbStop()
```

which terminates data accumulating and moves to step 11.

Step 11) after data have been accumulated or P_ClbStop() function has been called calibration status will change to **"00000000 00010010"** which means that accumulated data are currently calculating and you should wait until this process is done. In this stage you can stop rotating the device.

Step 12) after calibration data have been calculated calibration status will change to **"00000001 01100011"** or **"00000001 01100010"** where bit(0) indicates calibration success and other non-zero bits mean that it is expected that you:

- continue data accumulating in the next pitch by calling P_ClbNext() function and moving to step 7 (note 2D-2T calibration must include at least two runs with full 360° rotations of the weapon in azimuth with different pitch angles);
- or call P_ClbStop() function and move to step 13;
- or call P_ClbExit() function and move to step 15.

Step 13) after you used P_ClbStop() function in step 12, Status will change to **"00000001 10000011"** or **"00000001 10000010"** where bit(0) indicates calibration success and other non-zero bits mean that it is expected that you:

- call P_ClbExit() function without saving calibration results and move to step 15;
- or accept them by using P_ClbAccept() function.

Step 14) if you are satisfied with calibration results you can save them by using

```
P_Result  P_ClbAccept()
```

Step 15) after the calibration is done you are expected to finalize working with Pointer_SDK by using

```
P_Result  P_Close()
```

This function closes Pointer_SDK and returns resources to the operating system.

Table A3: Status Bit Description

Bit Number	Description
0	If == 1 then current calibration is successful
1	If == 1 then calibration is in progress
2	If == 1 then initial alignment is in progress
3	If == 1 then data are accumulating.
4	If == 1 then accumulated data are calculating.
5	If == 1 then function P_ClbNext is available
6	If == 1 then function P_ClbStop is available
7	If == 1 then function P_ClbAccept is available
8	If == 1 then function P_ClbExit is available

A4. Zone 3D Calibration Procedure

Step 1) before starting calibration in order to allocate memory from the operating system you should call the following function

```
P_Result   P_Open()
```

Step 2) it's also vital to set proper serial port by using

```
P_Result   P_SetPortNumber   (unsigned short PortNumber)
```

where **PortNumber** – necessary serial port number.

Step 3) you should set necessary data accumulation time

```
P_Result   P_SetClbAccumulationTime
        (unsigned short AccumulationTime)
```

where **AccumulationTime** sets in seconds.

Step 4) the next thing is to set current coordinates by using

```
P_Result   P_SetClbCoordinates
        (float Latitude, float Longitude, float Altitude, float Date)
```

where **Latitude** and **Longitude** set in degrees, **Altitude** – in meters, Date – in years (e.g. May-20-2012 = 2012 + 5 / 12 + 20 / 365 = 2012.4714).

Step 5) you should also set proper reference azimuth by calling

```
P_Result   P_SetClbRefAzimuth(float RefAzimuth)
```

where **RefAzimuth** sets in degrees.

Step 6) after this moment you can start calibration and move to the main loop in step 7

```
P_Result   P_ClbStart(P_ClbType)
```

where **P_ClbType** for Zone 3D calibration is CLB_ZONE = 0x14.

Note: the Zone 3D calibration procedure involves pointing of the weapon on at least four corners and intermediate points of the firing zone. The maximum number of calibration points is 9.

Step 7) once you have started Zone 3D calibration in order to get current calibration status you should call the following function from time to time

```
P_Result   P_GetClbStatus(unsigned short *Status)
```

where **Status** contains information about actions available currently (full explanation of status bits see in the end of this document).

In this stage received Status is equal to:

- **"00000001 00100010"** – if you used this function right after starting calibration);
- **"00000001 10100010"** or **"00000001 10100011"** – if you have already made at least 4 calibration points;
- **"00000001 10000010"** or **"00000001 10000011"** – if you have already made 9 calibration points.

This means that calibration is currently in progress and that you are expected:

- to call P_ClbNext() function for data accumulating in the next position (if bit(5) == 1);
- to call P_ClbAccept() function for saving calibration results (if bit(7) == 1);
- or to call P_ClbExit() function to terminate calibration procedure.

Step 8) to start data accumulating you should set the device in the necessary position and call the following functions

```
P_Result  P_SetClbBias
        (P_ClbBiasType BiasType, float BiasValue)
```

where **BiasType** should be set whether to BT_DEGS = 0x00 (for setting **BiasValue** in degrees) or to BT_MILS = 0x01 (for setting **BiasValue** in mills); **BiasValue** – is an azimuth shift relative to the reference azimuth set in step 5.

Then call

```
P_Result  P_ClbNext()
```

which starts data accumulation and moves to step 10. Note, that P_ClbNext() function is available only if you haven't made 9 calibration points yet.

Step 10) after calling P_ClbNext() function received Status is equal to **"00000000 00001010"**, which means that data are currently accumulating. Note, that you shouldn't move the device until this process is done (bit(3) == 0).

After data are accumulated you should move to step 7.

Step 11) if you have already made at least 4 calibration points you can accept calibration procedure results by using

```
P_Result  P_ClbAccept()
```

which terminates further calibration, saves results and moves to step 13.

Step 12) you can also terminate the whole calibration procedure by using

```
P_Result   P_ClbExit()
```

which terminates further calibration without saving its results and moves to step 13.

Step 13) after the calibration is done you are expected to finalize working with Pointer_SDK by using

```
P_Result   P_Close()
```

This function closes Pointer_SDK and returns resources to the operating system.

Table A4: Status Bit Description

Bit Number	Description
0	If == 1 then current calibration is successful
1	If == 1 then calibration is in progress
2	If == 1 then initial alignment is in progress
3	If == 1 then data are accumulating.
4	If == 1 then accumulated data are calculating.
5	If == 1 then function P_ClbNext is available
6	If == 1 then function P_ClbStop is available
7	If == 1 then function P_ClbAccept is available
8	If == 1 then function P_ClbExit is available