|  |
| --- |
| **CONFIDENTIAL** |
| **Calibration library and scripts integration guide and examples**  Venue. Survey  **Revision: 1.1** |

|  |
| --- |
| **Document Code** |

|  |
| --- |
|  |

|  |
| --- |
| Trusted Positioning Inc.  A TDK Group Company  #405, 1000 Veterans Place NW  Calgary, AB Canada T3B 4M1 |

|  |  |
| --- | --- |
| REVISION: 1.1 - ISSUED FOR USE | |
|  |  |
| **PERMIT STAMP** | **ENGINEER’S STAMP** |
|  | |

A picture containing shape

Description automatically generated

# Table of Contents

[Table of Contents 3](#_Toc72865558)

[Version Log 4](#_Toc72865559)

[Figures 5](#_Toc72865560)

[Tables 5](#_Toc72865561)

[Equations 5](#_Toc72865562)

[Acronyms 6](#_Toc72865563)

[Calibration Library 7](#_Toc72865564)

[Calibration Library Location 7](#_Toc72865565)

[Integration Guide 7](#_Toc72865566)

[Calibration Library Usage Example (C++) 9](#_Toc72865567)

[Example of Android Application 10](#_Toc72865568)

[How to apply calibration parameters to measurements of magnetometer and accelerometer 10](#_Toc72865569)

[A list of parameters to provide in FPBL 11](#_Toc72865570)

[Calibration process guide 12](#_Toc72865571)

[Related equipment and software preparation and usage 12](#_Toc72865572)

[Gimbal frame servo motors. Gear leveling 12](#_Toc72865573)

[Maestro board. Wiring 13](#_Toc72865574)

[Maestro board. Port settings 13](#_Toc72865575)

[Maestro board. Calibration script upload 14](#_Toc72865576)

[Maestro board. Calibration process 15](#_Toc72865577)

[Gimbal. Set gimbal to survey position 16](#_Toc72865578)

[Calibration script executing example 16](#_Toc72865579)

[Appendices 18](#_Toc72865580)

[Appendix A. General calibration process description 18](#_Toc72865581)

[Appendix B. Interaction with calibration library and calibration data flows 19](#_Toc72865582)

## Version Log

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date  (D/M/Y/ | Author | Checker | Update Description |
| 0.1 | 15/05/2021 | Y. Kotik |  | Initial version |
| 1.0 | 20/05/2021 | D. Churikov |  | Formatting |
| 1.1 | 25/05/2021 | D. Churikov |  | Gimbal script integration guide added |

## Figures

**No table of figures entries found.**

## Tables

**No table of figures entries found.**

## Equations

**No table of figures entries found.**

## Acronyms

|  |  |
| --- | --- |
| Abbreviation | Definition |
|  |  |
|  |  |

# Calibration Library

## Calibration Library Location

The library is located in Gift repository, branch: **master\_calibration\_lib**

Path is: **\Gift\Libraries\CalibrationLib\Source\** - for C++ source files

**\Gift\Libraries\CalibrationLib\jni\** -for JNI files

C++ interface file for SensorsCalibrator class is here: **\Gift\Libraries\CalibrationLib\Source\ISensorsCalibrator.h**

And the following file contains Java interface:

**\Gift\Libraries\CalibrationLib\java\src\com\tdk\sensorcalibrator\SensorCalibrator.java**

The following path contains a very simple Android application that can be used as an example of how to call library:

**\Gift\Applications\Calibration\_console\Android\app\**

## Integration Guide

In addition to the instruction below please refer to sections “General calibration process description” and “Interaction with calibration library and calibration data flows” of “Calibration Library Algorithm Description” document.

In order to use library user should do the following in his Java application:

1. Create SensorCalibrator object:

public SensorCalibrator()

1. Call startCalibration() method:

public native void startCalibration();

1. Call ClearSensorsData() method (it does nothing the first time, but is important if calibration failed and a repeated calibration is needed):

public native void ClearSensorsData();

1. Call AddMagneticData() method if magnetometer is being calibrated, or call AddAccelData() if accelerometer is being calibrated and pass the data arrays as input:

public native int AddMagneticData(

double[] time,

double[] x,

double[] y,

double[] z,

double[] temperature,

int Len

);

public native int AddAccelData(

double[] time,

double[] x,

double[] y,

double[] z,

double[] temperature,

int Len

);

**Note:** this method can be called multiple times to add more data before the next step happens, however it is important to provide all data in the ascending order with regards to timestamp.

This means that for each added array element the following is true: **time[i+1] > time[i]**

1. Call EstimateMagneticCalibrationParams() for magnetometer or EstimateAccelCalibrationParams() for accelerometer:

public native int EstimateMagneticCalibrationParams(SensorCalibrationParams calibrationParams);

public native int EstimateAccelCalibrationParams(SensorCalibrationParams calibrationParams);

1. Check return status. If status is 0, then calibration parameters can be applied and calibration is finished. If the status has any negative value, then calibration has failed for one of the possible reasons (insufficient data, bad data, unstable temperature, etc.) and calibration process should be repeated starting with a call to ClearSensorsData().

## Calibration Library Usage Example (C++)

There is a console application in the same branch **master\_calibration\_lib**. Location is:

**\Gift\Applications\Calibration\_console\**

This simple application serves as an example of how to use the library. It takes one input parameter – path to file, which contains the data in csv format. The output result of calibration goes to std::cout.

The following two test examples and the expected output for each of them are as follows:

1. Build the calibration library and the calibration console app, then run the app with the following input:

Calibration\_console.exe //cayyc-proj01/compute02/FPL\_DATA/test\_data/Robotic\_survey/Calibration\_with\_gimbal/good.mag/session02/test03\_it1\_calibration\_on\_robot\_imitation/2021\_04\_12\_15\_20\_37\_mag.csv > output.txt

The following output is expected:

status: 0

calib\_level: 4

accuracy: 3.85096

DOP: 0.189323

biases: 104.959 -80.3265 152.914

matrix: 0.979526 0.0191098 -0.00125516 0.0191098 1.00183 -0.00743634 -0.00125516 -0.00743634 1.01298

1. Build the calibration library and the calibration console app, then run the app with the following input:

Calibration\_console.exe //cayyc-proj01/compute02/FPL\_DATA/test\_data/Robotic\_survey/Calibration\_with\_gimbal/faults.mag/session01/it1/2021\_04\_30\_19\_07\_00\_mag.csv > output.txt

The following output is expected:

status: -7

calib\_level: 0

accuracy: 1.29009

DOP: 0.43457

biases: 112.831 -80.7497 150.553

matrix: 0.974832 0.0204489 -8.32032e-05 0.0204489 0.994867 -0.00737382 -8.32032e-05 -0.00737382 1.00398

Note that in the first example Calibration::ReturnStatus value returned by **EstimateAccelCalibrationParams** method is 0, which means STATUS\_SUCCESS, while in the second example status is -7, which is STATUS\_INSUFFICIENT\_DATA. Any status other than STATUS\_SUCCES means that bias estimation failed calibration process needs to be repeated. User of the library is expected to call **ClearSensorsData** method and repeat the calibration.

In the first scenario bias estimation succeeded and it means that bias values and ellipticity and scale matrix can be applied to the sensor data to be used in the survey.

## Example of Android Application

The application is located here:

**\Gift\Applications\Calibration\_console\Android\app\**

This simple application reads data from text files, sends it to Calibration Library and prints the calibration output. It was used to test JNI.

## How to apply calibration parameters to measurements of magnetometer and accelerometer

After magnetometer calibration process has been successfully completed with return status 0, calibration parameters can be applied to each magnetometer measurement as follows:

Where:

is 3x1 vector of calibrated magnetometer measurements,

is 3x1 vector of raw (uncalibrated) magnetometer measurements,

is 3x3 matrix of soft iron/scale/non-orthogonality compensation determined by method EstimateMagneticCalibrationParams,

, , are hard iron (bias) compensation parameters determined by method EstimateMagneticCalibrationParams.

Correction of accelerometer measurements are provided in the same way:

Where:

is 3x1 vector of calibrated accelerometer measurements,

is 3x1 vector of raw (uncalibrated) accelerometer measurements,

is 3x3 matrix of scale/non-orthogonality compensation determined by method EstimateAccelCalibrationParams,

, , are bias compensation parameters determined by method EstimateAccelCalibrationParams.

## A list of parameters to provide in FPBL

|  |  |  |
| --- | --- | --- |
|  | **Means** | **Data to be provided** |
| mandatory | Irl.dat, Magnetometer data entity (0x0062) | Calibrated mag data  raw mag data  calibration accuracy |
| desirable | mag\_bias.txt file | calibration level  mag bias (hard iron)  mag bias covariance matrix  soft iron matrix |

# Calibration process guide

## Related equipment and software preparation and usage

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Item** | **When** | **Action** |
| 1 | Gimbal frame servo motors | Gimbal assembling | Gear leveling |
| 2 | Mini Maestro 12 board | Gimbal assembling | Wiring |
| 3 | Mini Maestro 12 board | Gimbal assembling | Port settings |
| 4 | Mini Maestro 12 board | Gimbal assembling | Calibration script upload |
| 5 | Mini Maestro 12 board | Calibration | Call calibration script |
| 6 | Gimbal | After calibration | Set gimbal to survey position |

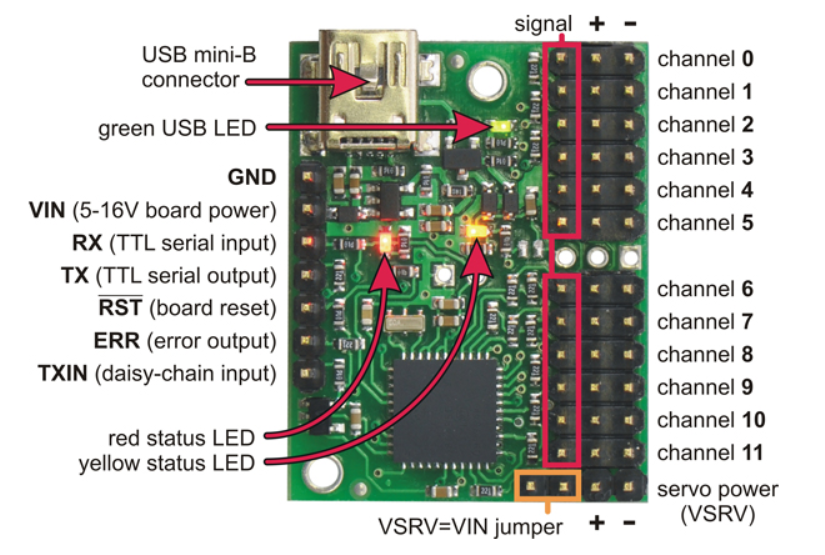
## Gimbal frame servo motors. Gear leveling

During the assembling frame servo motors must be connected to gears such way that the gimbal frames stay horizontal in motors positions as defined below:

|  |  |  |
| --- | --- | --- |
| **Item** | **Motor position** | **Tolerance in gimbal frames leveling around horizon** |
| Internal gimbal frame servo motor | 1920 | 2 deg |
| Internal gimbal frame servo motor | 5300 | 2 deg |

## Maestro board. Wiring

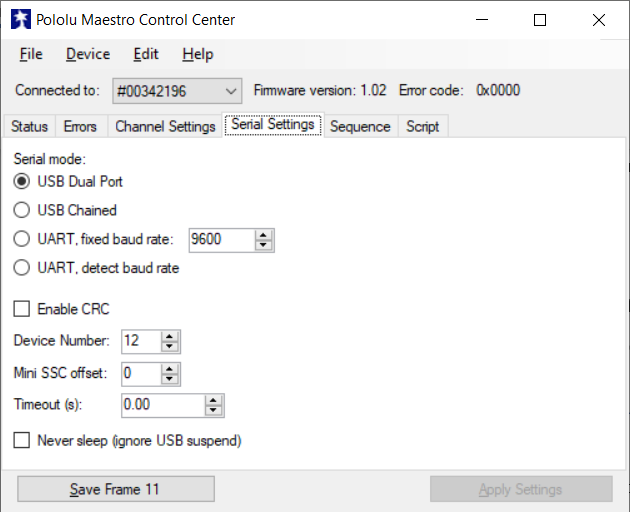
1. Connect control cable Internal gimbal frame servo motor to channel 0
2. Connect control cable External gimbal frame servo motor to channel 1
3. Connect power wires to servo power
4. Connect USB cable to USB connector



## Maestro board. Port settings

Follow to the instruction below

1. Connect the board to PC
2. Run Pololu Maestro Control Center and connect to the board
3. Go to Serial Settings tab
4. Set Serial Mode as USB Dual Port
5. Push Apply Setting button

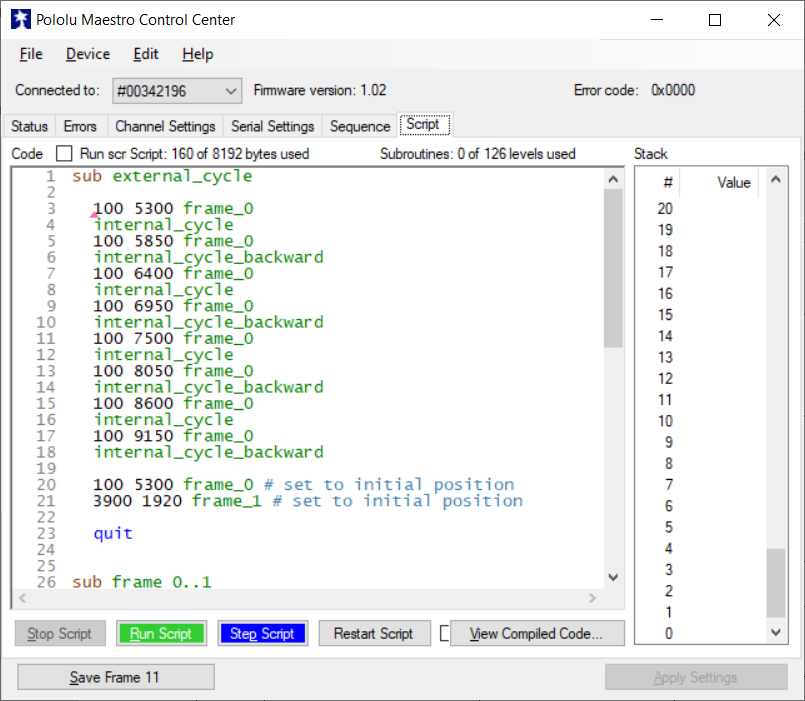


## Maestro board. Calibration script upload

Calibration script location: Gift\Libraries\CalibrationLib\scripts\calibration\_scripts\_for\_maestro\_board\calibration\_script\_to\_upload.txt

Follow to the instruction below to upload script in the board

1. Connect the board to PC
2. Run Pololu Maestro Control Center and connect to the board
3. Load the script with File->OpenSettingsFile menu item.
4. Go to Script tab
5. Push Apply Setting button
6. Push Rut Script button and check that script is working properly: internal frame’s rotated +/-90 deg, external frame’s rotated 45deg after 3 rotations of internal frame



## Maestro board. Calibration process

During the calibration process rotation of tag shall be provided in accordance with calibration program. The robot must complete 8 turns by 45 degrees as it is defined in “Interaction with calibration library and calibration data flows” section of “Calibration Library Algorithm Description” document. This is also available in Appendix B.

Master application must communicate with Maestro board to call calibration script. Communication with Maestro boar is provided via virtual COM port by Maestro Command Protocol.

Please see here for details about the protocol: <https://www.pololu.com/docs/0J40/5.c>

Parameters of virtual COM port by default are as follows:

|  |  |
| --- | --- |
| **Setting** | **Value** |
| Baud rate | 9600 |

See also serial settings here: <https://www.pololu.com/docs/0J40/5.a>

Please refer to sections “General calibration process description” and “Interaction with calibration library and calibration data flows” of “Calibration Library Algorithm Description” document for the information about calibration steps and data flows. These items are also available in Appendix.

The following commands are used to control and check state of the motors.

|  |  |  |
| --- | --- | --- |
| **Command** | **Command code in protocol** | **Parameters** |
| Restart Script at Subroutine | 0xA7 | subroutine\_number = 0 |
| Get Script Status | 0xAE |  |
| Get Errors | 0xA1 |  |

Refer for the Maestro Command description <https://www.pololu.com/docs/0J40/5.e> and <https://www.pololu.com/docs/0J40/5.f>

## Gimbal. Set gimbal to survey position

Calibration script sets gimbal frames to survey position automatically in the end of the calibration procedure.

## Calibration script executing example

There is a Windows console application located in Gift\Applications\Calibration\_process\_example\.

This simple application serves as an example of how to communicate with the Maestro board, runs Calibration script, waits for script ending and runs it again to simulate calibration process on the robot.

A cross-platform C example of communication with Maestro board is available here: <https://www.pololu.com/docs/0J40/5.h.1>

# Appendices

## Appendix A. General calibration process description



## Appendix B. Interaction with calibration library and calibration data flows



|  |  |  |
| --- | --- | --- |
| **Calibration parameter** | **Value** | **Description** |
| dH | 45 deg | Robot rotation angle around vertical axis |
| iteration\_number | 8 | Calibration iterations number |