**Background**

This document gives detailed explanation on the SDK structure and API functions. We assume that you have

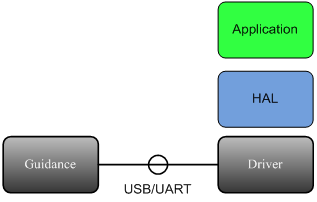
* a Guidance system,
* a computer with OpenCV installed,

and you are:

* familiar with Linux programming,
* or familiar with Windows programming and Microsoft Visual Studio.

**Introduction**

This section introduces the structure of the Guidance SDK. The SDK is divided into three layers:



* **Application:** This layer processes data from the HAL layer. It is written by developers.
* **HAL:** Hardware Abstraction Layer. This layer packs/parses the data to/from the Driver layer. It is implemented by the sample code (for UART) or SDK library (for USB), e.g. *libDJI\_guidance.so*.
* **Driver:** This layer receives data from the Guidance system through USB/UART. It is implemented by OS or 3rd party libraries, e.g. *libusb*.

**Interface**

The Guidance SDK supports two communication protocols: USB & UART.

**1. USB**

The supported data types are Velocity Data, Obstacle Distance Data, IMU Data, Ultrasonic Data, Greyscale Image, and Depth Image.

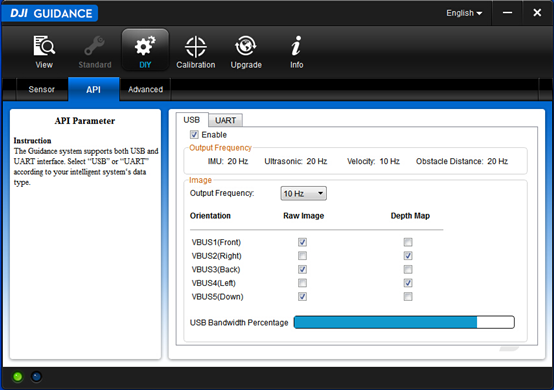
There are two ways to subscribe the data through USB.

1. Guidance Assistant Software

User can use Guidance assistant software to subscribe the data in "DIY->API->USB" tab.

* + Connect Guidance with PC using USB cable, power on the Guidance
  + Choose the "Enable" check box
  + Choose the data according your requirement

**Notes:** The available bandwidth is subject to the selection of image data and the output frequency. The selection of subscribed image data and output frequency will be saved and take effect when the Guidance system is turned off and on again.



1. Guidance API

User can subscribe the data using Guidance API. Identity these API functions that are named with "select".

**Notes:** If user subscribes the image data and output frequency using Guidance API functions, it will only temporarily override the data selection that is made in the Guidance Assistant software when the Guidance system is still powered on. However, the data selection that is made through the Guidance API will not permanently change the data subsections options stored in the Guidance system, unless you de-select the "Enable" option in the "USB" tab.

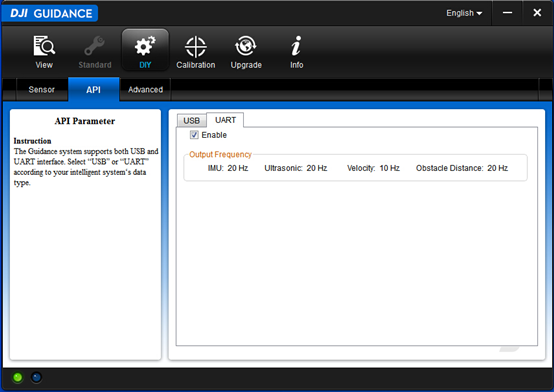
**2. UART**

The output data types of UART are Velocity Data, Obstacle Distance Data, IMU Data, and Ultrasonic Data. Image data are not output via UART due to the bandwidth limit.

**Note:** Guidance UART only supports **115200** baud rate.

1. Subscribe Data

You may only use Guidance assistant software to subscribe UART data. Enable this selection from "DIY->API->UART" page. Same as USB, the configuration will be saved in Guidance Core, unless you de-select the "Enable" option in the "UART" tab.



1. Protocol Description

Protocol Frame Format:

| **SOF** | **LEN** | **VER** | **RES** | **SEQ** | **CRC16** | **DATA** | **CRC32** |
| --- | --- | --- | --- | --- | --- | --- | --- |

Protocol Frame Explanation:

| **Field** | **Byte Index** | **Size（bit）** | **Description** |
| --- | --- | --- | --- |
| SOF | 0 | 8 | Frame start number, fixed to be 0xAA |
| LEN | 1 | 10 | Frame length, maximum length is 1023 bytes |
| VER | 1 | 6 | Version of the protocol |
| RES | 5 | 40 | Reserved bits, fixed to be 0 |
| SEQ | 8 | 16 | Frame sequence number |
| CRC16 | 10 | 16 | Frame header CRC16 checksum |
| DATA | 12 | --① | Frame data, maximum length 1007 bytes |
| CRC32 | --② | 32 | Frame CRC32 checksum |

① Frame data size can vary, 1007 is the maximum length.

② The index of this field depends on the length of the data field.

Data Field Format:

| **COMMAND SET** | **COMMAND ID** | **COMMAND DATA** |
| --- | --- | --- |

Data Field Explanation:

| **Data Field** | **Byte Index** | **Size（byte）** | **Description** |
| --- | --- | --- | --- |
| COMMAND SET | 0 | 1 | Always 0x00 |
| COMMAND ID | 1 | 1 | e\_image: 0x00; e\_imu: 0x01; e\_ultrasonic: 0x02; e\_velocity: 0x03; e\_obstacle\_distance: 0x04 |
| COMMAND DATA | 2 | -- | Data body |

**Data Types**

Each of the supported data types is described below.

* [**Error Code**](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#e-sdk-err-code) enumerates possible error codes. When error occurs, usually an error code will be given, and the developer can reference this enum to find the error type.
* [**Velocity Data:**](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#velocity) velocity in body frame. The unit is **millimeter/second** and the frequency is 10 Hz.
* [**Obstacle Distance Data:**](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#obstacle-distance) obstacle distance from five Guidance Sensors. The unit is **centimeter** and the frequency is 20 Hz.
* [**IMU Data:**](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#imu-data) IMU data including accelerometer (in unit of acceleration of gravity **g**) and gyroscope (in quaternion format) data. The frequency is 20 Hz.
* [**Motion Data:**](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#motion-data) Pose and velocity data including quaternion orientation, position in the global frame, velocity in the global frame.
* [**Ultrasonic Data:**](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#ultrasonic-data) Outputs ultrasonic data from five Guidance Sensors, including obstacle distance (in unit of **meter**) and reliability of the data. The frequency is 20 Hz.
* [**Greyscale Image:**](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#image-data) Outputs Greyscale images for five directions. The image size is 320\*240 bytes for individual sensor. The default frequency is 20 Hz and can be scaled down using API functions.
* [**Depth Image:**](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#image-data) Outputs depth images for five directions. The image size is 320\*240\*2 bytes for each direction. The default frequency is 20 Hz and can be scaled down using API functions.
* [**Disparity Image:**](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#image-data) Outputs disparity images for five directions. This data is useful when developers want to further refine the disaprity images using functions like speckle filter. The image size is 320\*240\*2 bytes for each direction. The default frequency is 20 Hz and can be scaled down using API functions.

**Data Structures**

**e\_sdk\_err\_code**

**Description:** Define error code of SDK.

|  |
| --- |
| enum e\_sdk\_err\_code  {  e\_timeout = -7, // time out  e\_libusb\_io\_err = -1, // libusb IO error  e\_OK = 0, // Succeed with no error  e\_load\_libusb\_err=1, // Load libusb library error  e\_sdk\_not\_inited=2, // SDK software is not ready  e\_hardware\_not\_ready=3, // Guidance hardware is not ready  e\_disparity\_not\_allowed=4, // Disparity or depth image is not allowed  e\_image\_frequency\_not\_allowed=5, // Image frequency must be one of the enum type e\_image\_data\_frequecy  e\_config\_not\_ready=6, // Config is not ready  e\_online\_flag\_not\_ready=7, // Online flag is not ready  e\_stereo\_cali\_not\_ready = 8,// Stereo calibration parameters are not ready  e\_max\_sdk\_err = 100 // maximum number of possible SDK errors  }; |

**Explanation:**

1. e\_timeout: time out during USB transfer.
2. e\_libusb\_io\_err: IO error returned by libusb library. This can be caused by physical connection problem of USB.
3. e\_OK: Succeed with no error.
4. e\_load\_libusb\_err: Load libusb library error. This is caused by the inappropriate libusb library.
5. e\_sdk\_not\_inited: SDK software is not ready.
6. e\_hardware\_not\_ready: Guidance hardware is not ready.
7. e\_disparity\_not\_allowed: If your Guidance is working in standard mode with obstacle sensing function activated, disparity or depth image is not allowed to select. The reason is, obstacle sensing has its own way to select disparity images.
8. e\_image\_frequency\_not\_allowed: Image frequency must be one of the enum type e\_image\_data\_frequecy.
9. e\_config\_not\_ready: Configuration data is not ready. When Guidance is powered on, it takes several seconds (sometimes longer) to initiate, including loading configuration data (including other data) into memory, and sending to application layer (i.e. the SDK software). If the users start SDK application before configuration data is ready, this error will be thrown. Configuration data includes: Guidance working mode, Guidance Sensor online status, stereo calibration parameters, and so on.
10. e\_online\_flag\_not\_ready: Online flag is not ready. Guidance system allows users to use any number of sensors, from 1 to 5. We use an array of online status to indicate which sensor are online. If users subscribe data from a sensor that is not online, no data will be returned.
11. e\_stereo\_cali\_not\_ready: Stereo calibration parameters are not ready. The calibration parameters are useful for 3D applications. As the images are already rectified, no distortion coefficients are provided, but only coordinates of the principal point cu, cv, focal length focal, and baseline baseline.

**e\_vbus\_index**

**Description:** Define logical direction of vbus, i.e. the direction of selected Guidance Sensor. Note that they are only defined by the VBUS ports on Guidance Core, not by the Guidance Sensors.

The comment of each element indicates the default direction when Guidance is installed on Matrice 100. However the developers can install Guidance in any manner on any device, thus the directions might also be different.

|  |
| --- |
| enum e\_vbus\_index  {  e\_vbus1 = 1, // front on M100  e\_vbus2 = 2, // right on M100  e\_vbus3 = 3, // rear on M100  e\_vbus4 = 4, // left on M100  e\_vbus5 = 0 // down on M100  }; |

**e\_image\_data\_frequecy**

**Description:** Define frequency of image data. The supported frequencies are: 5Hz, 10Hz, 20Hz. With more images selected, smaller frequency should be selected.

|  |
| --- |
| enum e\_image\_data\_frequecy  {  e\_frequecy\_5 = 0, // frequecy of image data: 5Hz  e\_frequecy\_10 = 1, // frequecy of image data: 10Hz  e\_frequecy\_20 = 2 // frequecy of image data: 20Hz  }; |

**e\_guidance\_event**

**Description:** Define event type of callback

|  |
| --- |
| enum e\_guidance\_event  {  e\_image = 0, // called back when image comes  e\_imu, // called back when imu comes  e\_ultrasonic, // called back when ultrasonic comes  e\_velocity, // called back when velocity data comes  e\_obstacle\_distance, // called back when obstacle data comes  e\_motion, // called back when global position comes  e\_event\_num  }; |

**image\_data**

**Description:** Define image data structure. For each direction of stereo camera pair, the depth image aligns with the left greyscale image.

|  |
| --- |
| typedef struct \_image\_data  {  unsigned int frame\_index; // frame index  unsigned int time\_stamp; // time stamp of image captured in ms  char \*m\_greyscale\_image\_left[CAMERA\_PAIR\_NUM]; // greyscale image of left camera  char \*m\_greyscale\_image\_right[CAMERA\_PAIR\_NUM]; // greyscale image of right camera  char \*m\_depth\_image[CAMERA\_PAIR\_NUM]; // depth image in \*128 meters  char \*m\_disparity\_image[CAMERA\_PAIR\_NUM]; // disparity image in \*16 pixels  }image\_data; |

**Explanation:**

1. m\_greyscale\_image\_left and m\_greyscale\_image\_right are both 320 hight, 240 width, 8 bit grayscale images.
2. m\_depth\_image is 320 hight, 240 width, 16 bit depth image. Every 2 bytes describes the depth of a single point (in big-endian format), with the lower 7 bits being fraction and higher 9 bits being integer.
3. m\_disparity\_image is 320 hight, 240 width, 16 bit depth image. Every 2 bytes describes the disparity of a single point (in big-endian format), with the lower 4 bits being fraction and higher 12 bits being integer.

**ultrasonic\_data**

**Description:** Define ultrasonic data structure. ultrasonic is the distance between Guidance Sensor and the nearest object detected by ultrasonic sensor. The Unit is mm. reliability is the reliability of this distance measurement, with 1 meaning reliable and 0 unreliable. **Note:** Due to noise in the distance measurement, it is recommended to filter the data before use.

|  |
| --- |
| typedef struct \_ultrasonic\_data  {  unsigned int frame\_index; // correspondent frame index  unsigned int time\_stamp; // time stamp of correspondent image captured in ms  short ultrasonic[CAMERA\_PAIR\_NUM]; // distance in mm. -1 means invalid.  unsigned short reliability[CAMERA\_PAIR\_NUM]; // reliability of the distance data  }ultrasonic\_data; |

**velocity**

**Description:** Define velocity in body frame coordinate. Unit is mm/s.

|  |
| --- |
| typedef struct \_velocity  {  unsigned int frame\_index; // correspondent frame index  unsigned int time\_stamp; // time stamp of correspondent image captured in ms  short vx; // velocity of x in mm/s  short vy; // velocity of y in mm/s  short vz; // velocity of z in mm/s  }velocity; |

**obstacle\_distance**

**Description:** Define obstacle distance calculated by fusing vision and ultrasonic sensors. Unit is cm.

|  |
| --- |
| typedef struct \_obstacle\_distance  {  unsigned int frame\_index; // correspondent frame index  unsigned int time\_stamp; // time stamp of correspondent image captured in ms  unsigned short distance[CAMERA\_PAIR\_NUM]; // distance of obstacle in cm  }obstacle\_distance; |

**imu data**

**Description:** Define IMU data structure. Unit of acceleration is m/s^2.

|  |
| --- |
| typedef struct \_imu  {  unsigned int frame\_index; // correspondent frame index  unsigned int time\_stamp; // time stamp of correspondent image captured in ms  float acc\_x; // acceleration of x in unit of m/s^2  float acc\_y; // acceleration of y in unit of m/s^2  float acc\_z; // acceleration of z in unit of m/s^2  float q[4]; // quaternion: [w,x,y,z]  }imu; |

**stereo\_cali**

**Description:** Calibration parameters of cameras. All values will be zero if the corresponding sensor is not online.

|  |
| --- |
| typedef struct \_stereo\_cali  {  float cu; // x position of focal center in units of pixels  float cv; // y position of focal center in units of pixels  float focal; // focal length in units of pixels  float baseline; // baseline of stereo cameras in units of meters  \_stereo\_cali() { }  \_stereo\_cali(float \_cu, float \_cv, float \_focal, float \_baseline)  {  cu = \_cu, cv = \_cv;  focal = \_focal, baseline = \_baseline;  }  }stereo\_cali; |

**exposure\_param**

**Description:** Parameters of camera exposure. When m\_expo\_time = m\_expected\_brightness=0, return to default AEC.

|  |
| --- |
| typedef struct \_exposure\_param  {  float m\_step; // adjustment step for auto exposure control (AEC). Default is 10.  float m\_exposure\_time; // constant exposure time in mini-seconds. Range is 0.1~20. Default is 7.25.  unsigned int m\_expected\_brightness;// constant expected brightness for AEC. Range is 50~200. Default is 85.  unsigned int m\_is\_auto\_exposure; // 1: auto exposure; 0: constant exposure  int m\_camera\_pair\_index; // index of Guidance Sensor  \_exposure\_param(){  m\_step = 10;  m\_exposure\_time = 7.68;  m\_expected\_brightness = 85;  m\_is\_auto\_exposure = 1;  m\_camera\_pair\_index = 1;  }  }exposure\_param; |

**motion data**

**Description:** Define global motion data. Unit is m for position and m/s for velocity.

|  |
| --- |
| typedef struct \_motion  {  unsigned int frame\_index;  unsigned int time\_stamp;  int corresponding\_imu\_index;  float q0;  float q1;  float q2;  float q3;  int attitude\_status; // 0:invalid; 1:valid  float position\_in\_global\_x; // position in global frame: x  float position\_in\_global\_y; // position in global frame: y  float position\_in\_global\_z; // position in global frame: z  int position\_status; // lower 3 bits are confidence. 0:invalid; 1:valid  float velocity\_in\_global\_x; // velocity in global frame: x  float velocity\_in\_global\_y; // velocity in global frame: y  float velocity\_in\_global\_z; // velocity in global frame: z  int velocity\_status; // lower 3 bits are confidence. 0:invalid; 1:valid  float reserve\_float[8];  int reserve\_int[4];  float uncertainty\_location[3];// uncertainty of position  float uncertainty\_velocity[3];// uncertainty of velocity  } motion; |

**API**

**Overview**

The Guidance API provides configuration and control methods for Guidance with C interface. Here is an overview of the key methods available in this API.

Please reference the protocol of Section 2.1.2 and also the example code of uart\_example when using UART transfer type.

* initialization
  + [reset\_config](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#reset-config)
  + [init\_transfer](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#init-transfer)
* subscribe data
  + [select\_imu](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#select-imu)
  + [select\_ultrasonic](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#select-ultrasonic)
  + [select\_velocity](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#select-velocity)
  + [select\_obstacle\_distance](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#select-obstacle-distance)
  + [set\_image\_frequecy](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#set-image-frequecy)
  + [select\_depth\_image](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#select-depth-image)
  + [select\_disparity\_image](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#select-disparity-image)
  + [select\_greyscale\_image](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#select-greyscale-image)
  + [select\_motion](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#select-motion)
* set callback and exposure
  + [set\_sdk\_event\_handler](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#set-sdk-event-handler)
  + [set\_exposure\_param](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#set-exposure-param)
* get data
  + [get\_online\_status](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#get-online-status)
  + [get\_stereo\_cali](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#get-stereo-cali)
  + [get\_device\_type](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#get-device-type)
  + [get\_image\_size](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#get-image-size)
* transfer control
  + [start\_transfer](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#start-transfer)
  + [stop\_transfer](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#stop-transfer)
  + [release\_transfer](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#release-transfer)
  + [wait\_for\_board\_ready](https://developer.dji.com/guidance-sdk/documentation/introduction/index.html#wait-for-board-ready)

**Method**

**user\_call\_back**

* **Description:** Callback function prototype. The developer must write his/her own callback function in this form. In order to achieve best performance, it is suggested not performing any time-consuming processing in the callback function, but only copying the data out. Otherwise the transfer frequency might be slowed down.
* **Parameters:** event\_type use it to identify the data:image,imu,ultrasonic,velocity or obstacle distance
* **Parameters:** data\_len length of the input data
* **Parameters:** data data read from Guidance.
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| typedef int (\*user\_call\_back)( int event\_type, int data\_len, char \*data ); |

**reset\_config**

* **Description:** Clear subscribed configure, if you want to subscribe data different from last time.
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| SDK\_API int reset\_config( void ); |

**init\_transfer**

* **Description:** Initialize Guidance and create data transfer thread.
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| SDK\_API int init\_transfer( void ); |

**select\_imu**

* **Description:** Subscribe IMU data. In standard mode, IMU data can only be output when Guidance is connected to DJI N1 flight controller. While in DIY mode, IMU data can always be output without connecting to a flight controller.
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| SDK\_API void select\_imu( void ); |

**select\_ultrasonic**

* **Description:** Subscribe ultrasonic data.
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| SDK\_API void select\_ultrasonic( void ); |

**select\_velocity**

* **Description:** subscribe velocity data, i.e. velocity of Guidance in body coordinate system.
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| SDK\_API void select\_velocity( void ); |

**select\_obstacle\_distance**

* **Description:** Subscribe obstacle distance.
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| SDK\_API void select\_obstacle\_distance( void ); |

**select\_greyscale\_image**

* **Description:** Subscribe rectified greyscale image.
* **Parameters:** camera\_pair\_index index of camera pair selected
* **Parameters:** is\_left whether the image data selected is left
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| SDK\_API int select\_greyscale\_image( e\_vbus\_index camera\_pair\_index, bool is\_left ); |

**select\_depth\_image**

* **Description:** Subscribe depth image.
* **Parameters:** camera\_pair\_index index of camera pair selected
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| SDK\_API int select\_depth\_image( e\_vbus\_index camera\_pair\_index ); |

**Example:**

|  |
| --- |
| #include "DJI\_guidance.h"  #include "DJI\_utility.h"  #include "opencv2/opencv.hpp"  #include <stdio.h>  #include <string>  e\_vbus\_index sensor\_id = e\_vbus1;  Mat g\_depth;  int my\_callback(int data\_type, int data\_len, char \*content)  {  g\_lock.enter();  if (e\_image == data\_type && NULL != content)  {  image\_data\* data = (image\_data\* )content;  if ( data->m\_depth\_image[sensor\_id] ){  g\_depth = Mat::zeros(HEIGHT,WIDTH,CV\_16SC1);  memcpy( g\_depth.data, data->m\_depth\_image[sensor\_id], IMAGE\_SIZE \* 2 );  }  }  g\_lock.leave();  g\_event.set\_event();  return 0;  }  int main(int argc, const char\*\* argv)  {  reset\_config(); // clear all data subscription  int err\_code = init\_transfer(); //wait for board ready and init transfer thread  err\_code = select\_depth\_image( sensor\_id );    err\_code = set\_sdk\_event\_handler( my\_callback );  err\_code = start\_transfer();    while(1)  {  g\_event.wait\_event();  if(!g\_depth.empty()){  Mat depth8(HEIGHT,WIDTH,CV\_8UC1);  g\_depth.convertTo(depth8, CV\_8UC1);  imshow(string("depth\_")+char('0'+sensor\_id), depth8);  printf("Depth at point (%d,%d) is %f meters!\n", HEIGHT/2, WIDTH/2, float(g\_depth.at<short>( HEIGHT/2,WIDTH/2))/128);  }  }  err\_code = stop\_transfer();  //make sure the ack packet from GUIDANCE is received  sleep( 1000000 );  err\_code = release\_transfer();  } |

**select\_disparity\_image**

* **Description:** Subscribe disparity image, which can be filtered with functions such as filterSpeckles.
* **Parameters:** camera\_pair\_index index of camera pair selected
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| SDK\_API int select\_disparity\_image( e\_vbus\_index camera\_pair\_index ); |

**Example:**

|  |
| --- |
| #include "DJI\_guidance.h"  #include "DJI\_utility.h"  #include "opencv2/opencv.hpp"  #include <stdio.h>  #include <string>  e\_vbus\_index sensor\_id = e\_vbus1;  Mat g\_disparity;  int my\_callback(int data\_type, int data\_len, char \*content)  {  g\_lock.enter();  if (e\_image == data\_type && NULL != content)  {  image\_data\* data = (image\_data\* )content;  if ( data->m\_disparity\_image[sensor\_id] ){  g\_disparity = Mat::zeros(HEIGHT,WIDTH,CV\_16SC1);  memcpy( g\_disparity.data, data->m\_disparity\_image[sensor\_id], IMAGE\_SIZE \* 2 );  }  }  g\_lock.leave();  g\_event.set\_event();  return 0;  }  int main(int argc, const char\*\* argv)  {  reset\_config(); // clear all data subscription  int err\_code = init\_transfer(); //wait for board ready and init transfer thread  err\_code = select\_disparity\_image( sensor\_id );    err\_code = set\_sdk\_event\_handler( my\_callback );  err\_code = start\_transfer();    while(1)  {  g\_event.wait\_event();  if(!g\_disparity.empty()){  Mat disp8(HEIGHT,WIDTH,CV\_8UC1);  g\_disparity.convertTo(disp8, CV\_8UC1);  imshow(string("disparity\_")+char('0'+sensor\_id), disp8);  printf("Disparity at point (%d,%d) is %f pixels!\n", HEIGHT/2, WIDTH/2, float(g\_disparity.at<short>( HEIGHT/2,WIDTH/2))/16);  }  }  err\_code = stop\_transfer();  //make sure the ack packet from GUIDANCE is received  sleep( 1000000 );  err\_code = release\_transfer();  } |

**select\_motion**

* **Description:** Subscribe global motion data, i.e. velocity and position of Guidance in global coordinate system.
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| SDK\_API void select\_motion( void ); |

**set\_image\_frequecy**

* **Description:** Set frequecy of image transfer. **Note**: The bandwidth of USB is limited. If you subscribe too many images (greyscale image or depth image), the frequency should be set relatively small, otherwise the SDK cannot guarantee the continuity of image transfer.
* **Parameters:** frequecy frequecy of image transfer
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| SDK\_API int set\_image\_frequecy( e\_image\_data\_frequecy frequecy ); |

**start\_transfer**

* **Description:** Inform Guidance to start data transfer.
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| SDK\_API int start\_transfer( void ); |

**stop\_transfer**

* **Description:** Inform Guidance to stop data transfer.
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| SDK\_API int stop\_transfer( void ); |

**release\_transfer**

* **Description:** Release the data transfer thread.
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| SDK\_API int release\_transfer( void ); |

**set\_sdk\_event\_handler**

* **Description:** Set callback function handler. When data from Guidance comes, it will be called by data transfer thread.
* **Parameters:** handler function pointer to callback function.
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| SDK\_API int set\_sdk\_event\_handler( user\_call\_back handler ); |

**get\_stereo\_cali**

* **Description:** Get stereo calibration parameters.
* **Parameters:** stereo\_cali\_pram Array of calibration parameters for all sensors.
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| SDK\_API int get\_stereo\_cali( stereo\_cali stereo\_cali\_pram[CAMERA\_PAIR\_NUM]); |

**get\_online\_status**

* **Description:** Get the online status of Guidance sensors.
* **Parameters:** online\_status Array of online status for all sensors.
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| SDK\_API int get\_online\_status(int online\_status[CAMERA\_PAIR\_NUM]); |

**get\_device\_type**

* **Description:** Get the type of devices. Currently only support one type of device: Guidance.
* **Parameters:** device\_type Device type.
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| SDK\_API int get\_device\_type(e\_device\_type\* device\_type); |

**get\_image\_size**

* **Description:** Get the size of image data.
* **Parameters:** width Image width.
* **Parameters:** height Image height.
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| SDK\_API int get\_image\_size(int\* width, int\* height); |

**wait\_for\_board\_ready**

* **Description:** Wait for board ready signal. This function waits 20 seconds for Guidance board to get started. If 20 seconds pass and the board is still not ready, return a timeout error code. The users usually do not need to use this function, as it is already called in init\_transfer.
* **Return:** error code. Zero if succeed, otherwise e\_timout.

|  |
| --- |
| SDK\_API int wait\_for\_board\_ready(); |

**set\_exposure\_param**

* **Description:** Set exposure mode and parameters.
* **Parameters:** param pointer of exposure parameter struct.
* **Return:** error code. Non-zero if error occurs.

|  |
| --- |
| SDK\_API int set\_exposure\_param( exposure\_param \*param ); |